

8

INPUT, OUTPUT, AND COMMUNICATION

In the earlier chapters, we considered how the CPU interacts with data that is accessed internal to the CPU, or is accessed within the main memory, which may be extended to a hard magnetic disk through virtual memory. While the access speeds at the different levels of the memory hierarchy vary dramatically, for the most part, the CPU sees the same response rate from one access to the next. The situation when accessing input/output (I/O) devices is very different.

- The speeds of I/O data transfers can range from extremely slow, such as reading data entered from a keyboard, to so fast that the CPU may not be able to keep up, as may be the case with data streaming from a fast disk drive, or real time graphics being written to a video monitor.
- I/O activities are **asynchronous**, that is, not synchronized to the CPU clock, as are memory data transfers. Additional signals, called **handshaking** signals, may need to be incorporated on the I/O bus to coordinate when the device is ready to have data read from it or written to it.
- The quality of the data may be suspect. For example, line noise during data transfers using the public telephone network, or errors caused by media defects on disk drives mean that error detection and correction strategies may be needed to ensure data integrity.
- Many I/O devices are mechanical, and are in general more prone to failure than the CPU and main memory. A data transfer may be interrupted due to mechanical failure, or special conditions such as a printer being out of paper, for example.
- I/O software modules, referred to as **device drivers**, must be written in such a way as to compensate for the properties mentioned above.

In this chapter we will first discuss the nature of the I/O devices themselves,

beginning with mass storage devices and then input and output devices. Following that we discuss the nature of the communications process with these devices, and we conclude with a treatment of error detection and correction techniques.

8.1 Mass Storage

In Chapter 7, we saw that computer memory is organized as a hierarchy, in which the fastest method of storing information (registers) is expensive and not very dense, and the slowest methods of storing information (tapes, disks, *etc.*) are inexpensive and are very dense. Registers and random access memories require continuous power to retain their stored data, whereas **media** such as magnetic tapes and magnetic disks retain information indefinitely after the power is removed, which is known as **indefinite persistence**. This type of storage is said to be **non-volatile**. There are many kinds of non-volatile storage, and only a few of the more common methods are described below. We start with one of the more prevalent forms: the **magnetic disk**.

8.1.1 MAGNETIC DISKS

A magnetic disk is a device for storing information that supports a large storage density and a relatively fast access time. A **moving head** magnetic disk is composed of a stack of one or more **platters** that are spaced several millimeters apart and are connected to a **spindle**, as shown in Figure 8-1. Each platter has two **surfaces** made of aluminum or glass (which expands less than aluminum as it heats up), which are coated with extremely small particles of a magnetic material such as iron oxide, which is the essence of rust. This is why disk platters, floppy diskettes, audio tapes, and other magnetic media are brown. Binary 1's and 0's are stored by magnetizing small areas of the material.

A single **head** is dedicated to each surface. Six heads are used in the example shown in Figure 8-1, for the six surfaces. The top surface of the top platter and the bottom surface of the bottom platter are sometimes not used on multi-platter disks because they are more susceptible to contamination than the inner surfaces. The heads are attached to a common **arm** (also known as a **comb**) which moves in and out to reach different portions of the surfaces.

In a "hard disk drive," as it is called, the platters rotate at a constant speed of typically 3600 to 10,000 revolutions per minute (RPM). The heads read or write data by magnetizing the magnetic material as it passes under the heads when writing, or by sensing the magnetic fields when reading. Only a single head is

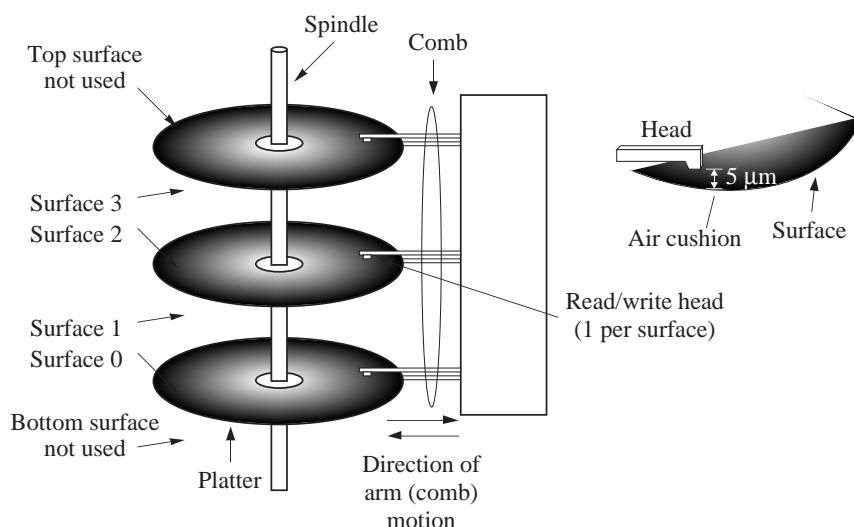


Figure 8-1 A magnetic disk with three platters.

used for reading or writing at any time, so data is stored in serial fashion even though the heads can theoretically be used to read or write several bits in parallel. One reason that the parallel mode of operation is not normally used is that heads can become misaligned, which corrupts the way that data is read or written. A single surface is relatively insensitive to the alignment of the corresponding head because the head position is always accurately known with respect to reference markings on the disk.

Data encoding

Only the transitions between magnetized areas are sensed when reading a disk, and so runs of 1's or 0's will not be detected unless a method of encoding is used that embeds timing information into the data to identify the breaks between bits. **Manchester encoding** is one method that addresses this problem, and another method is **modified frequency modulation (MFM)**. For comparison, Figure 8-2a shows an ASCII 'F' character encoded in the **non-return-to-zero (NRZ)** format, which is the way information is encoded inside of a CPU. Figure 8-2b shows the same character encoded in the Manchester code. In Manchester encoding there is a transition between high and low signals on every bit, resulting in a transition at every bit time. A transition from low to high indicates a 1, whereas a transition from high to low indicates a 0. These transitions are used to recover the timing information.

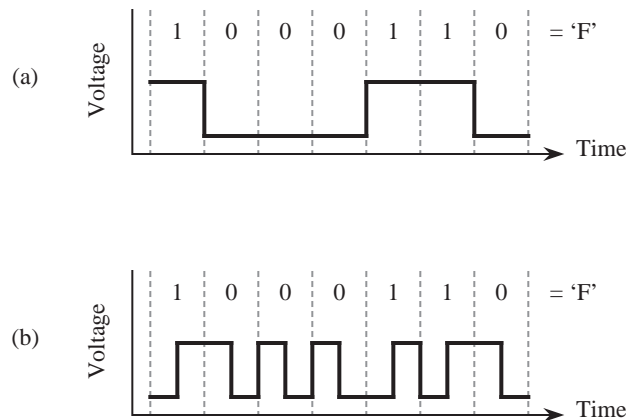


Figure 8-2 (a) Straight amplitude (NRZ) encoding of ASCII 'F'; (b) Manchester encoding of ASCII 'F'.

A single surface contains several hundred concentric **tracks**, which in turn are composed of **sectors** of typically 512 bytes in size, stored serially, as shown in Figure 8-3. The sectors are spaced apart by **inter-sector gaps**, and the tracks are

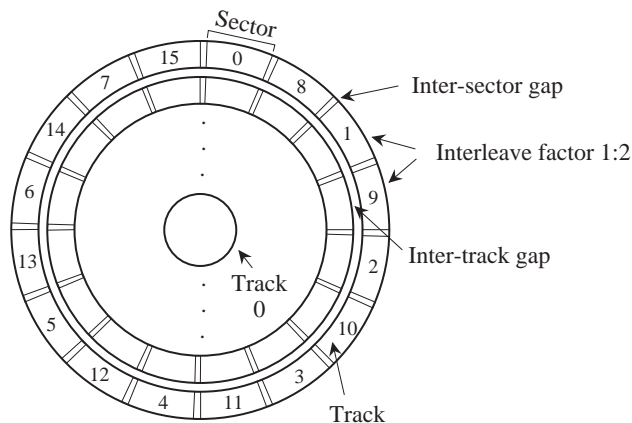


Figure 8-3 Organization of a disk platter with a 1:2 interleave factor.

spaced apart by **inter-track gaps**, which simplify positioning of the head. A set of corresponding tracks on all of the surfaces forms a **cylinder**. For instance, track 0 on each of surfaces 0, 1, 2, 3, 4, and 5 in Figure 8-1 collectively form cylinder 0. The number of bytes per sector is generally invariant across the entire platter.

In modern disk drives the number tracks per sector may vary in **zones**, where a

zone is a group of tracks having the same number of sectors per track. Zones near the center of the platter where bits are spaced closely together have fewer sectors, while zones near the outside periphery of the platter, where bits are spaced farther apart, have more sectors per track. This technique for increasing the capacity of a disk drive is known as **zone-bit recording**.

Disk drive capacities and speeds

If the disk is assumed to have only a single zone, its storage capacity, C , can be computed from the number of bytes per sector, N , the number of sectors per track, S , the number of tracks per surface, T , and the number of platter surfaces that have data encoded in them, P , with the formula:

$$C = N \times S \times T \times P$$

A high-capacity disk drive may have $N = 512$ bytes, $S = 1,000$ sectors per track, $T = 5,000$ tracks per platter, and $P = 8$ platters. The total capacity of this drive is $C = 512 \times 1000 \times 5000 \times 8 / 2^{30}$ or 19 GB.

Maximum data transfer speed is governed by three factors: the time to move the head to the desired track, referred to as the head **seek time**, the time for the desired sector to appear under the read/write head, known as the **rotational latency**, and the time to transfer the sector from the disk platter once the sector is positioned under the head, known as the **transfer time**. Transfers to and from a disk are always carried out in complete sectors. Partial sectors are never read or written.

Head seek time is the largest contributor to overall access time of a disk. Manufacturers usually specify an average seek time, which is roughly the time required for the head to travel half the distance across the disk. The rationale for this definition is that there is no way to know, *a priori*, which track the data is on, or where the head is positioned when the disk access request is made. Thus it is assumed that the head will, on average, be required to travel over half the surface before arriving at the correct track. On modern disk drives average seek time is approximately 10 ms.

Once the head is positioned at the correct track, it is again impossible to know ahead of time how long it will take for the desired sector to appear under the head. Therefore the average rotational latency is taken to be 1/2 the time of one

complete revolution, which is on the order of 4-8 ms. The sector transfer time is just the time for one complete revolution divided by the number of sectors per track. If large amounts of data are to be transferred, then after a complete track is transferred, the head must move to the next track. The parameter of interest here is the track-to-track access time, which is approximately 2 ms (notice that the time for the head to travel past multiple tracks is much less than 2 ms per track). An important parameter related to the sector transfer time is the **burst rate**, the rate at which data streams on or off the disk once the read/write operation has started. The burst rate equals the disk speed in revolutions per second \times the capacity per track. This is not necessarily the same as the transfer rate, because there is a setup time needed to position the head and synchronize timing for each sector.

The maximum transfer rate computed from the factors above may not be realized in practice. The limiting factor may be the speed of the bus interconnecting the disk drive and its interface, or it may be the time required by the CPU to transfer the data between the disk and main memory. For example, disks that operate with the **Small Computer Systems Interface** (SCSI) standards have a transfer rate between the disk and a host computer of from 5 to 40 MB/second, which may be slower than the transfer rate between the head and the internal buffer on the disk. Disk drives invariably contain internal buffers that help match the speed of the disk with the speed of transfer from the disk unit to the host computer.

Disk drives are delicate mechanisms.

The strength of a magnetic field drops off as the square of the distance from the source of the field, and for that reason, it is important for the head of the disk to travel as close to the surface as possible. The distance between the head and the platter can be as small as 5 μm . The engineering and assembly of a disk do not have to adhere to such a tight tolerance – the head assembly is aerodynamically designed so that the spinning motion of the disk creates a cushion of air that maintains a distance between the heads and the platters. Particles in the air contained within the disk unit that are larger than 5 μm can come between the head assembly and the platter, which results in a **head crash**.

Smoke particles from cigarette ash are 10 μm or larger, and so smoking should not take place when disks are exposed to the environment. Disks are usually assembled into sealed units in **clean rooms**, so that virtually no large particles are introduced during assembly. Unfortunately, materials used in manufacturing

(such as glue) that are internal to the unit can deteriorate over time and can generate particles large enough to cause a head crash. For this reason, sealed disks (formerly called **Winchester** disks) contain filters that remove particles generated within the unit and that prevent particulate matter from entering the drive from the external environment.

Floppy disks

A **floppy disk**, or **diskette**, contains a flexible plastic platter coated with a magnetic material like iron oxide. Although only a single side is used on one surface of a floppy disk in many systems, both sides of the disks are coated with the same material in order to prevent warping. Access time is generally slower than a hard disk because a flexible disk cannot spin as quickly as a hard disk. The rotational speed of a typical floppy disk mechanism is only 300 RPM, and may be varied as the head moves from track to track to optimize data transfer rates. Such slow rotational speeds mean that access times of floppy drives are 250-300 ms, roughly 10 times slower than hard drives. Capacities vary, but range up to 1.44 MB.

Floppies are inexpensive because they can be removed from the drive mechanism and because of their small size. The head comes in physical contact with the floppy disk but this does not result in a head crash. It does, however, place wear on the head and on the media. For this reason, floppies only spin when they are being accessed.

When floppies were first introduced, they were encased in flexible, thin plastic enclosures, which gave rise to their name. The flexible platters are currently encased in rigid plastic and are referred to as “diskettes.”

Several high-capacity floppy-like disk drives have made their appearance in recent years. The Iomega Zip drive has a capacity of 100 MB, and access times that are about twice those of hard drives, and the larger Iomega Jaz drive has a capacity of 2GB, with similar access times.

Another floppy drive recently introduced by Imation Corp., the SuperDisk, has floppy-like disks with 120MB capacity, and in addition can read and write ordinary 1.44 MB floppy disks.

Disk file systems

A **file** is a collection of sectors that are linked together to form a single logical entity. A file that is stored on a disk can be organized in a number of ways. The most efficient method is to store a file in consecutive sectors so that the seek time and the rotational latency are minimized. A disk normally stores more than one file, and it is generally difficult to predict the maximum file size. Fixed file sizes are appropriate for some applications, though. For instance, satellite images may all have the same size in any one sampling.

An alternative method for organizing files is to assign sectors to a file on demand, as needed. With this method, files can grow to arbitrary sizes, but there may be many head movements involved in reading or writing a file. After a disk system has been in use for a period of time, the files on it may become **fragmented**, that is, the sectors that make up the files are scattered over the disk surfaces. Several vendors produce optimizers that will defragment a disk, reorganizing it so that each file is again stored on contiguous sectors and tracks.

A related facet in disk organization is **interleaving**. If the CPU and interface circuitry between the disk unit and the CPU all keep pace with the internal rate of the disk, then there may still be a hidden performance problem. After a sector is read and buffered, it is transferred to the CPU. If the CPU then requests the next contiguous sector, then it may be too late to read the sector without waiting for another revolution. If the sectors are interleaved, for example if a file is stored on alternate sectors, say 2, 4, 6, *etc.*, then the time required for the intermediate sectors to pass under the head may be enough time to set up the next transfer. In this scenario, two or more revolutions of the disk are required to read an entire track, but this is less than the revolution per sector that would otherwise be needed. If a single sector time is not long enough to set up the next read, than a greater interleave factor can be used, such as 1:3 or 1:4. In Figure 8-3, an interleave factor of 1:2 is used.

An operating system has the responsibility for allocating blocks (sectors) to a growing file, and to read the blocks that make up a file, and so it needs to know where to find the blocks. The **master control block** (MCB) is a reserved section of a disk that keeps track of the makeup of the rest of the disk. The MCB is normally stored in the same place on every disk for a particular type of computer system, such as the innermost track. In this way, an operating system does not have to guess at the size of a disk; it only needs to read the MCB in the innermost track.

Figure 8-4 shows one version of an MCB. Not all systems keep all of this infor-

Preamble	No. surfaces on disk = 4							
	No. tracks/surface = 814							
	No. sectors/track = 32							
	No. bytes/sector = 512							
	Interleave factor = 1:3							
Starting sector, or sector list								
Files	Filename	Surface	Track	Sector	Creation Date	Last Modified	Owner	Protec-tions
	xyz.p	1	10	5	11/14/93 10:30:57	11/14/93 19:30:57	16	RWX by Owner
		1	12	7				
		2	23	4				
	ab.c	1	10	8	8/18/93 16:03:12	1/21/94 14:45:03	20	RX - All W-Owner
		3	95	2				
		2	12	0				
			:					
			:					
	Free blocks		1	1	0			
		1	1	1				
		1	2	5				
			:					
Bad blocks		1	1	3				
		2	5	7				
			:					
			:					

R = Read
W = Write
X = Execute

Figure 8-4 Simplified example of an MCB.

mation in the MCB, but it has to be kept *somewhere*, and some of it may even be kept in part of the file itself. There are four major components to the MCB. The Preamble section specifies information relating to the physical layout of the disk, such as the number of surfaces, number of sectors per surface, *etc.* The Files section cross references file names with the list of sectors of which they are composed, and file attributes such as the file creation date, last modification date, the identification of the owner, and protections. Only the starting sector is needed for a fixed file size disk, otherwise, a list of all of the sectors that make up a file is maintained.

The Free blocks section lists the positions of blocks that are free to be used for new or growing files. The Bad blocks section lists positions of blocks that are free but produce **checksums** (see Section 8.5) that indicate errors. The bad blocks are thus unused.

As a file grows in size, the operating system reads the MCB to find a free block, and then updates the MCB accordingly. Unfortunately, this generates a great deal of head movement since the MCB and free blocks are rarely (if ever) on the same track. A solution that is used in practice is to copy the MCB to main memory and make updates there, and then periodically update the MCB on the disk, which is known as **syncing** the disk.

A problem with having two copies of the MCB, one on the disk and one in main memory, is that if a computer is shut down before the main memory version of the MCB is synced to the disk, then the integrity of the disk is destroyed. The normal shutdown procedure for personal computers and other machines syncs the disks, so it is important to shut down a computer this way rather than simply shutting off the power. In the event that a disk is not properly synced, there is usually enough circumstantial information for a disk recovery program to restore the integrity of the disk, often with the help of a user. (Note: See problem 8.10 at the end of the chapter for an alternative MCB organization that makes recovery easier.)

8.1.2 MAGNETIC TAPE

A magnetic tape unit typically has a single read / write head, but may have separate heads for reading and writing. A spool of plastic (Mylar) tape with a magnetic coating passes the head, which magnetizes the tape when writing or senses stored data when reading. Magnetic tape is an inexpensive means for storing large amounts of data, but access to any particular portion is slow because all of the prior sections of the tape must pass the head before the target section can be accessed.

Information is stored on a tape in two-dimensional fashion, as shown in Figure 8-5. Bits are stored across the width of the tape in **frames** and along the length of the tape in **records**. A file is made up of a collection of (typically contiguous) records. A record is the smallest amount of data that can be read from or written to a tape. The reason for this is physical rather than logical. A tape is normally motionless. When we want to write a record to the tape, then a motor starts the tape moving, which takes a finite amount of time. Once the tape is up to speed, the record is written, and the motion of the tape is then stopped, which again takes a finite amount of time. The starting and stopping times consume sections of the tape, which are known as **inter-record gaps**.

A tape is suitable for storing large amounts of data, such as backups of disks or

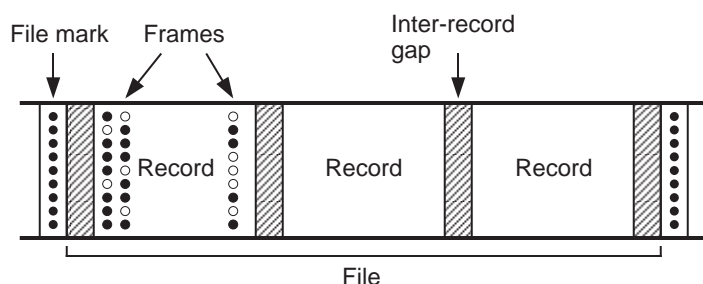


Figure 8-5 A portion of a magnetic tape (adapted from [Hamacher, 1990]).

scanned images, but is not suitable for random access reading and writing. There are two reasons for this. First, the sequential access can require a great deal of time if the head is not positioned near the target section of tape. The second reason is caused when records are overwritten in the middle of the tape, which is not generally an operation that is allowed in tape systems. Although individual records are the same size, the inter-record gaps eventually creep into the records (this is called **jitter**) because starting and stopping is not precise.

A **physical record** may be subdivided into an integral number of **logical records**. For example, a physical record that is 4096 bytes in length may be composed of four logical records that are each 1024 bytes in length. Access to logical records is managed by an operating system, so that the user has the perspective that the logical record size relates directly to a physical record size, when in fact, only physical records are read from or written to the tape. There are thus no inter-record gaps between logical records.

Another organization is to use variable length records. A special mark is placed at the beginning of each record so that there is no confusion as to where records begin.

8.1.3 MAGNETIC DRUMS

Although they are nearly obsolete today, magnetic drum units have traditionally been faster than magnetic disks. The reason for the speed advantage of drums is that there is one stationary head per track, which means that there is no head movement component in the access time. The rotation rate of a drum can be much higher than a disk as a result of a narrow cylindrical shape, and rotational delay is thus reduced.

The configuration of a drum is shown in Figure 8-6. The outer surface of the

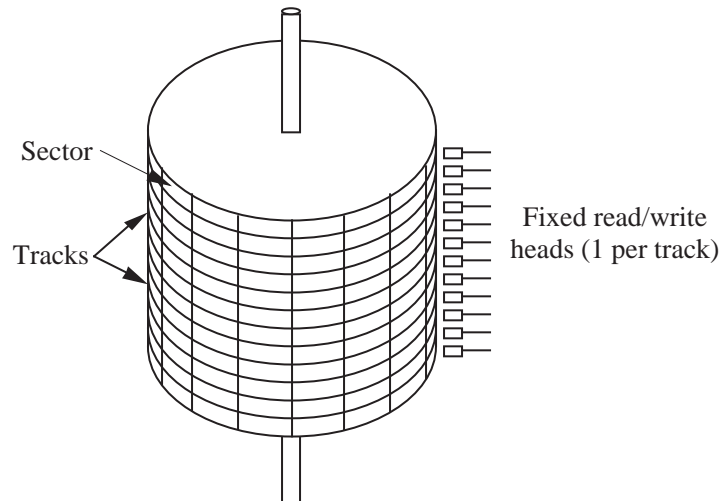


Figure 8-6 A magnetic drum unit.

drum is divided into a number of tracks. The top and bottom of the drum are not used for storage, and the interior of the drum is not used for storage, so there is less capacity per unit volume for a drum unit than there is for a disk unit.

The transfer time for a sector on a drum is determined by the rotational delay and the length of a sector. Since there is no head movement, there is no seek time to consider. Nowadays, **fixed head disks** are configured in a similar manner to drums with one head per track, but are considerably less expensive per stored bit than drums since the surfaces of platters are used rather than the outside of a drum.

8.1.4 OPTICAL DISKS

Several new technologies take advantage of optics to store and retrieve data. Both the **Compact Disc (CD)** and the newer **Digital Versatile Disc (DVD)** employ light to read data encoded on a reflective surface.

The Compact Disc

The CD was introduced in 1983 as a medium for playback of music. CDs have the capacity to store 74 minutes of audio, in digital stereo (2-channel) format.

The audio is sampled at $2 \times 44,000$ 16-bit samples per second, or nearly 700 MB capacity. Since the introduction of the CD in 1983, CD technology has improved in terms of price, density, and reliability, which led to the development of **CD ROMs** (CD read only memories) for computers, which also have the same 700 MB capacity. Their low cost, only a few cents each when produced in volume, coupled with good reliability and high capacity, have made CD ROMs the medium of choice for distributing commercial software, replacing floppy disks.

CD ROMs are “read only” because they are stamped from a master disk similar to the way that audio CDs are created. A CD ROM disk consists of aluminum coated plastic, which reflects light differently for **lands** or **pits**, which are smooth or pitted areas, respectively, that are created in the stamping process. The master is created with high accuracy using a high power laser. The pressed (stamped) disks are less accurate, and so a complex error correction scheme is used which is known as a **crossinterleaved Reed Solomon** error correcting code. Errors are also reduced by assigning 1’s to pit-land and land-pit transitions, with runs of 0’s assigned to smooth areas, rather than assigning 0’s and 1’s to lands and pits, as in Manchester encoding.

Unlike a magnetic disk in which all of the sectors on concentric tracks are lined up like a sliced pie (where the disk rotation uses **constant angular velocity**), a CD is arranged in a spiral format (using **constant linear velocity**) as shown in Figure 8-7. The pits are laid down on this spiral with equal spacing from one end

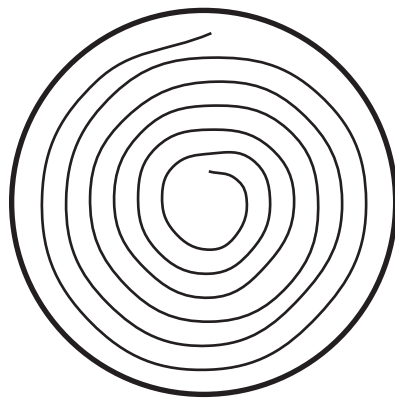


Figure 8-7 Spiral storage format for a CD.

of the disk to the other. The speed of rotation, approximately the same 30 RPM

as the floppy disk, is adjusted so that the disk moves more slowly when the head is at the edge than when it is at the center. Thus CD ROMs suffer from the same long access time as floppy disks because of the high rotational latency. CD ROM drives are available with rotational speeds up to 24 \times , or 24 times the rotational speed of an audio CD, with a resulting decrease in average access time.

CD ROM technology is appropriate for distributing large amounts of data inexpensively when there are many copies to be made, because the cost of creating a master and pressing the copies is distributed over the cost of each copy. If only a few copies are made, then the cost of each disk is high because CDs cannot be economically pressed in small quantities. CDs also cannot be written after they are pressed. A newer technology that addresses these problem is the **write once read many** (WORM) optical disk, in which a low intensity laser in the CD controller writes onto the optical disk (but only once for each bit location). The writing process is normally slower than the reading process, and the controller and media are more expensive than for CD ROMs.

The Digital Versatile Disc

A newer version of optical disk storage is the **Digital Versatile Disc**, or DVD. There are industry standards for DVD-Audio, DVD-Video, and DVD-ROM and DVD-RAM data storage. When a single side of the DVD is used, its storage capacity can be up to 4.7 GB. The DVD standards also include the capability of storing data on both sides in two layers on each side, for a total capacity of 17 GB. The DVD technology is an evolutionary step up from the CD, rather than being an entirely new technology, and in fact the DVD player is backwardly compatible—it can be used to play CDs and CD ROMs as well as DVDs.

8.2 Input Devices

Disk units, tape units, and drum units are all input/output (I/O) devices, and they share a common use for mass storage. In this section, we look at a few devices that are used exclusively for input of data. We start with one of the most prevalent devices – the **keyboard**.

8.2.1 KEYBOARDS

Keyboards are used for manual input to a computer. A keyboard layout using the ECMA-23 Standard (2nd ed.) is shown in Figure 8-8. The “QWERTY” layout (for the upper left row of letters D01 – D06) conforms to the traditional layout

	99	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
F																				
E		⌵	1	2	#	4	%	&	7	()	0	=	⌵						
D			Q	W	E	R	T	Y	U	I	O	P	@	}			7	8	9	
C			A	S	D	F	G	H	J	K	L	;	:	}			-	4	5	6
B			Z	X	C	V	B	N	M	<	>	?	/				1	2	3	
A																	0	00	.	SP
Z																				

Figure 8-8 Keyboard layout for the ECMA-23 Standard (2nd ed.). Shift keys are frequently placed in the B row.

used in typewriters. Frequently used letters are placed far apart so that the typist is slowed and jams in mechanical typewriters are reduced. Although jams are not a problem with electronic keyboards, the traditional layout prevails.

When a character is typed, a bit pattern is created that is transmitted to a host computer. For 7-bit ASCII characters, only 128 bit patterns can be used, but many keyboards that expand on the basic ECMA-23 standard use additional modifier keys (shift, escape, and control) and so a seven-bit pattern is no longer large enough. A number of alternatives are possible, but one method that has gained acceptance is to provide one bit pattern for each modifier key and other bit patterns for the remaining keys.

Other modifications to the ECMA-23 keyboard include the addition of function keys (in row F, for example), and the addition of special keys such as tab, delete, and carriage return. A modification that places frequently used keys together was developed for the **Dvorak keyboard** as shown in Figure 8-9. Despite the perfor-

"	,	.	P	Y	F	G	C	R	L	?	}
,	,	.								/	{
A	O	E	U	I	D	H	T	N	S	-	-
:	Q	J	K	X	B	M	W	V	Z		
:											

Figure 8-9 The Dvorak keyboard layout.

mance advantage of the Dvorak keyboard, it has not gained wide acceptance.

8.2.2 BIT PADS

A **digitizing tablet**, or **bit pad**, is an input device that consists of a flat surface and a **stylus** or **puck** as illustrated in Figure 8-10. The tablet has an embedded

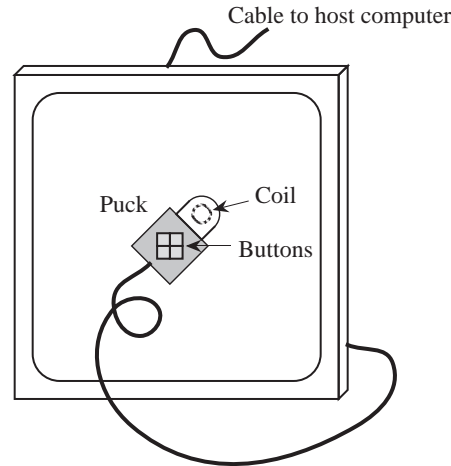


Figure 8-10 A bit pad with a puck.

two-dimensional mesh of wires that detects an induced current created by the puck as it is moved about the tablet. The bit pad transmits X-Y (horizontal-vertical) positions and the state of the buttons on the puck (or stylus) either continuously, or for an event such as a key click or a movement, depending on the control method. Bit pads are commonly used for entering data from maps, photographs, charts, or graphs.

8.2.3 MICE AND TRACKBALLS

A **mouse** is a hand-held input device that consists of a rubber ball on the bottom and one or more buttons on the top as illustrated in the left side of Figure 8-11. As the mouse is moved, the ball rotates proportional to the distance moved. Potentiometers within the mouse sense the direction of motion and the distance traveled, which are reported to the host along with the state of the buttons. Two button events are usually distinguished: one for the key-down position and one for the key-up position.

A **trackball** can be thought of as a mouse turned upside down. The trackball unit is held stationary while the ball is manually rotated. The configuration of a trackball is shown in the right side of Figure 8-11.

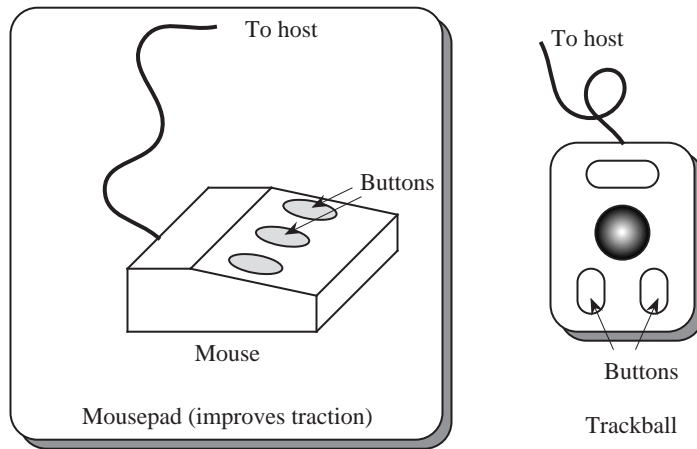


Figure 8-11 A three-button mouse (left) and a three-button trackball (right).

An **optical mouse** replaces the ball with a **light emitting diode (LED)** and uses a special reflective mousepad that consists of alternating reflective and absorptive horizontal and vertical stripes. Motion is sensed through transitions between reflective and absorptive areas. The optical mouse does not accumulate dirt as readily as the ball mouse, and can be used in a vertical position or even in a weightless environment. The natural rotation of the wrist and elbow, however, do not match the straight horizontal and vertical stripes of the optical mousepad, and so some familiarity is required by the user in order to use the device effectively.

8.2.4 LIGHTPENS AND TOUCH SCREENS

Two devices that are typically used for selecting objects are **lightpens** and **touch screens**. A lightpen does not actually produce light, but senses light from a video screen as illustrated in Figure 8-12. An electron beam excites a phosphor coating

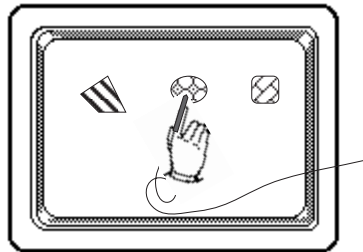


Figure 8-12 A user selecting an object with a lightpen.

on the back of the display surface. The phosphor glows and then dims as it returns to its natural state. Each individual spot is refreshed at a rate of 30 – 60 Hz, so that a user perceives a continuous image.

When a dim spot is refreshed, it becomes brighter, and this change in intensity signals the location of the beam at a particular time. If the lightpen is positioned at a location where the phosphor is refreshed, then the position of the electron beam locates the position of the pen. Since the lightpen detects intensity, it can only distinguish among illuminated areas. Dark areas of the screen all appear the same since there is no change in intensity over time.

A touch screen comes in two forms, photonic and electrical. An illustration of the photonic version is shown in Figure 8-13. A matrix of beams covers the

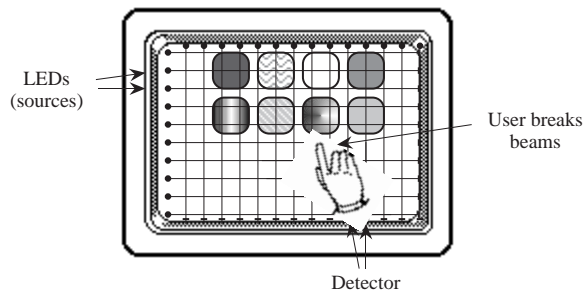


Figure 8-13 A user selecting an object on a touch screen.

screen in the horizontal and vertical dimensions. If the beams are interrupted (by a finger for example) then the position is determined by the interrupted beams.

In an alternative version of the touch screen, the display is covered with a touch sensitive surface. The user must make contact with the screen in order to register a selection.

8.2.5 JOYSTICKS

A **joystick** indicates horizontal and vertical position by the distance a rod that protrudes from the base is moved (see Figure 8-14). Joysticks are commonly used in video games, and for indicating position in graphics systems. Potentiometers within the base of the joystick translate X-Y position information into voltages, which can then be encoded in binary for input to a digital system. In a spring-loaded joystick, the rod returns to the center position when released. If

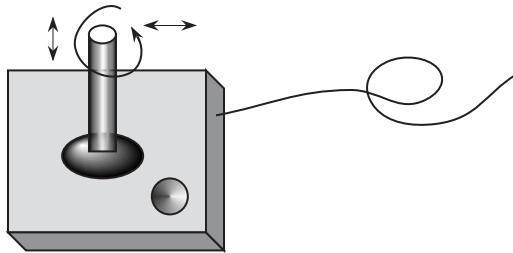


Figure 8-14 A joystick with a selection button and a rotatable rod.

the rod can be rotated, then an additional dimension can be indicated, such as height.

8.3 Output Devices

There are many types of output devices. In the sections below, we explore two common output devices: the **laser printer** and the **video display**.

8.3.1 LASER PRINTERS

A laser printer consists of a charged drum in which a laser discharges selected areas according to a bit mapped representation of a page to be printed. As the drum advances for each scan line, the charged areas pick up electrostatically sensitive toner powder. The drum continues to advance, and the toner is transferred to the paper, which is heated to fix the toner on the page. The drum is cleaned of any residual toner and the process repeats for the next page. A schematic diagram of the process is shown in Figure 8-15.

Since the toner is a form of plastic, rather than ink, it is not absorbed into the page but is melted onto the surface. For this reason, a folded sheet of laser printed paper will display cracks in the toner along the fold, and the toner is sometimes unintentionally transferred to other materials if exposed to heat or pressure (as from a stack of books).

Whereas older printers could print only ASCII characters, or occasionally crude graphics, the laser printer is capable of printing arbitrary graphical information. Several languages have been developed for communicating information from computer to printer. One of the most common is the **Adobe PostScript** language. PostScript is a stack-based language that is capable of describing objects as diverse as ASCII characters, high level shapes such as circles and rectangles, and low-level bit maps. It can be used to describe foreground and background colors,

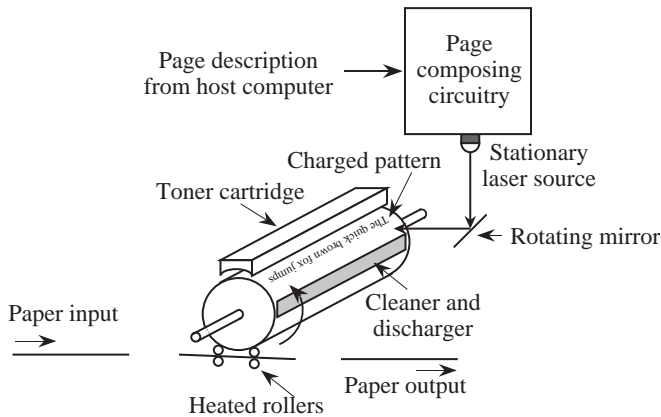


Figure 8-15 Schematic of a laser printer (adapted from [Tanenbaum, 1999]).

and fill colors with which to fill objects.

8.3.2 VIDEO DISPLAYS

A video display, or **monitor**, consists of a luminescent display device such as a cathode ray tube (CRT) or a liquid crystal panel, and controlling circuitry. In a CRT, vertical and horizontal deflection plates steer an electron beam that sweeps the display screen in **raster** fashion (one line at a time, from left to right, starting at the top).

A configuration for a CRT is shown in Figure 8-16. An electron gun generates a

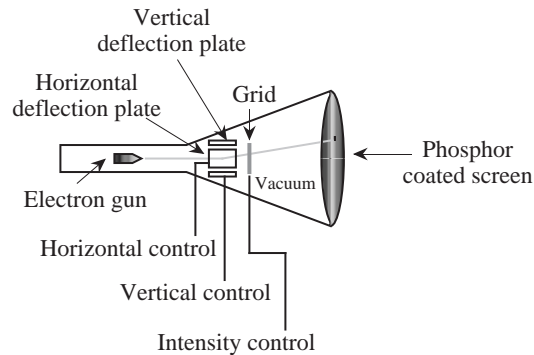


Figure 8-16 A CRT with a single electron gun.

stream of electrons that is imaged onto a phosphor coated screen at positions

controlled by voltages on the vertical and horizontal deflection plates. Electrons are negatively charged, and so a positive voltage on the grid accelerates electrons toward the screen and a negative voltage repels electrons away from the screen. The color produced on the screen is determined by the characteristics of the phosphor. For a color CRT, there are usually three different phosphor types (red, green, and blue) that are interleaved in a regular pattern, and three guns, which produce three beams that are simultaneously deflected on the screen.

A simple display controller for a CRT is shown in Figure 8-17. The writing of

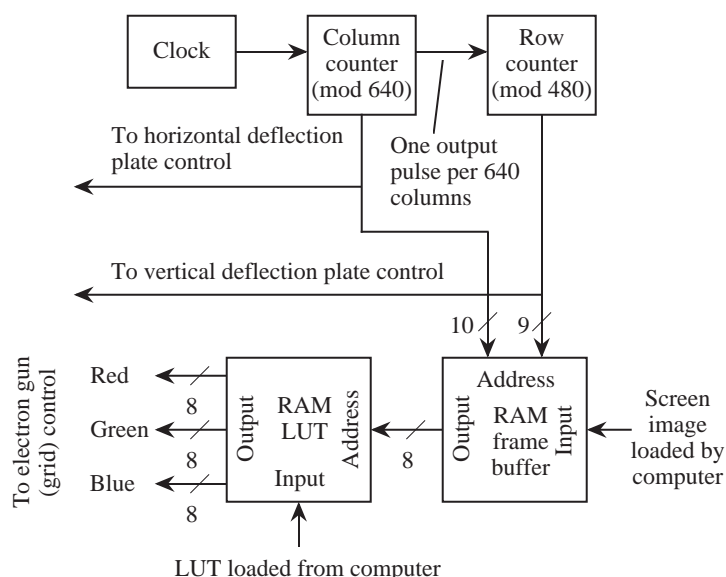


Figure 8-17 Display controller for a 640×480 color monitor (adapted from [Hamacher *et al.*, 1990]).

information on the screen is controlled by the “dot clock,” which generates a continuous stream of alternating 1’s and 0’s at a rate that corresponds to the update time for a single spot on the screen. A single spot is called a **picture element**, or **pixel**. The display controller in Figure 8-17 is for a screen that is 640 pixels wide by 480 pixels high. A column counter is incremented from 0 to 639 for each row, then repeats, and a row counter is incremented from 0 to 479, which then repeats. The row and column addresses index into the **frame buffer**, or “display RAM” that holds the bit patterns corresponding to the image that is to be displayed on the screen. The contents of the frame buffer are transferred to the screen from 30 to 100 times per second. This technique of mapping a RAM area to the screen is referred to as **memory-mapped video**. Each pixel on the screen may be represented by from 1 to 12 or more bits in the frame buffer.

When there is only a single bit per pixel, the pixel can only be on or off, black or white; multiple bits per pixel allow a pixel to have varying colors, or shades of gray.

Each pixel in the display controller of Figure 8-17 is represented by eight bits in the frame buffer, which means that one out of 2^8 , or 256 different intensities can be used for each pixel. In a simple configuration the eight bits can be partitioned for the red, green, and blue (R, G, and B) inputs to the CRT as three bits for red, three bits for green, and two bits for blue. An alternative is to pass the eight-bit pixel value to a color lookup table (LUT) in which the eight-bit value is translated into 256 different 24-bit colors. Eight bits of the 24-bit output are then used for each of the red, green, and blue guns. A total of 2^{24} different colors can be displayed, but only 2^8 of the 2^{24} can be displayed at any one time since the LUT has only 2^8 entries. The LUT can be reloaded as necessary to select different subsets of the 2^{24} colors. For example, in order to display a gray scale image (no color), we must have $R=G=B$ and so a ramp from 0 to 255 is stored for each of the red, green, and blue guns.

The human eye is relatively slow when compared with the speed of an electronic device, and cannot perceive a break in motion that happens at a rate of about 25 Hz or greater. A computer screen only needs to be updated 25 or 30 times a second in order for an observer to perceive a continuous image. Whereas video monitors for computer applications can have any scan rate that the designer of the monitor and video interface card wish, in television applications the scan rate must be standardized. In Europe, a rate of 25 Hz is used for standard television, and a rate of 30 Hz is used in North America. The phosphor types used in the screens do not have a long persistence, and so scan lines are updated alternately in order to reduce flicker. The screen is thus updated at a 50 Hz rate in Europe and at a 60 Hz rate in North America, but only alternating lines are updated on each sweep. For high resolution graphics, the entire screen may be updated at a 50 Hz or 60 Hz rate, rather than just the alternating lines. Many observers believe that the European choice of 50 Hz was a bad one, because many viewers can detect the 50 Hz as a flicker in dim lighting or when viewed at the periphery of vision.

On the other hand, the Europeans point to the United States **NTSC** video transmission standard as being inferior to their **PAL** system, referring to the NTSC system as standing for “Never The Same Color,” because of its poorer ability to maintain consistent color from frame to frame.

The data rates between computer and video monitor can be quite high. Consider that a 24-bit monitor with 1024×768 pixel resolution and a refresh rate of 60 Hz will require a **bandwidth** (that is, the amount of information that can be carried over a given period of time) of $3 \text{ bytes/pixel} \times (1024 \times 768) \text{ pixels} \times 60 \text{ Hz}$, or roughly 140 MB per second. Fortunately, the hardware described above maps the frame buffer onto the screen without CPU intervention, but it is still up to the CPU to output pixels to the frame buffer as the image on the screen changes.

8.4 Communication

“Communication” is the process of transferring information from a source to a destination, and “telecommunications” is the process of communicating at a distance. Communication systems range from busses within an integrated circuit to the public telephone system, and radio and television. Wide-area telecommunication systems have become very complex, with all combinations of voice, video, and data being transferred by wire, optical fiber, radio, and microwaves. The routes that communication takes may traverse cross-country, under water, through local radio cells, and via satellite. Data that originates as analog voice signals may be converted to digital data streams for efficient routing over long distances, and then converted back to an analog signal, without the knowledge of those communicating.

In this chapter we focus on the relatively short range communications between entities located at distances ranging from millimeters to a kilometer or so. An example of the former is the interactions between a CPU and main memory, and an example of the latter is a **local area network** (LAN). The LAN is used to interconnect computers located within a kilometer or so of one another. In Chapter 10 we extend our discussion to **wide area networks** (WANs) as typified by the Internet.

In the next sections we discuss communications from the viewpoints of communications at the CPU and motherboard level, and then branch out to the local area network.

8.4.1 BUSES

Unlike the long distance telecommunications network, in which there may be many senders and receivers over a large geographical distance, a computer has only a small number of devices that are geographically very local, within a few

millimeters to a few meters of each other. In a worst case scenario, all devices need to simultaneously communicate with every other device, in which $N^2/2$ links are needed for N devices. The number of links becomes prohibitively large for even small values of N , but fortunately, as for long distance telecommunication, not all devices need to simultaneously communicate.

A **bus** is a common pathway that connects a number of devices. An example of a bus can be found on the **motherboard** (the main circuit board that contains the central processing unit) of a personal computer, as illustrated in Figure 8-18. A

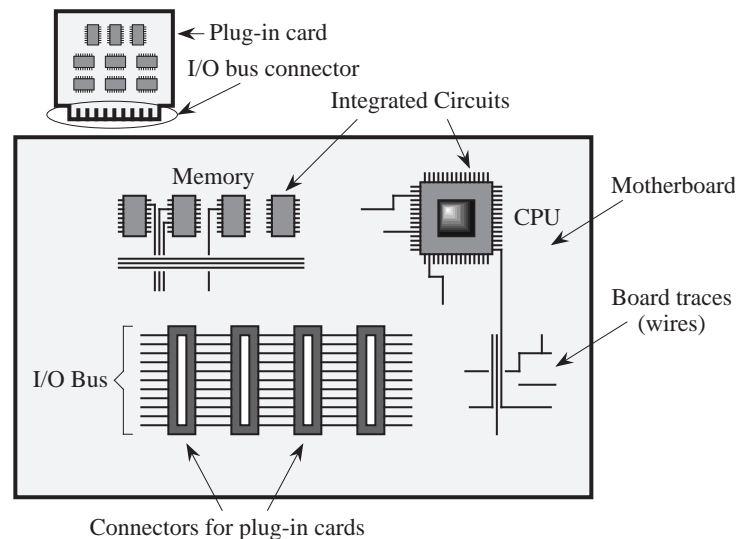


Figure 8-18 A motherboard of a personal computer (top view).

typical motherboard contains **integrated circuits** (ICs) such as the CPU chip and memory chips, board **traces** (wires) that connect the chips, and a number of busses for chips or devices that need to communicate with each other. In the illustration, an I/O bus is used for a number of cards that plug into the connectors, perpendicular to the motherboard in this example configuration.

Bus Structure, Protocol, and Control

A bus consists of the physical parts, like connectors and wires, and a **bus protocol**. The wires can be partitioned into separate groups for control, address, data, and power as illustrated in Figure 8-19. A single bus may have a few different power lines, and the example shown in Figure 8-19 has lines for ground (GND) at 0 V, +5 V, and -15 V.

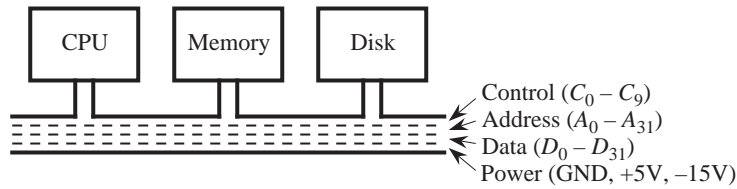


Figure 8-19 Simplified illustration of a bus.

The devices share a common set of wires, and only one device may send data at any one time. All devices simultaneously listen, but normally only one device receives. Only one device can be a **bus master**, and the remaining devices are then considered to be **slaves**. The master controls the bus, and can be either a sender or a receiver.

An advantage of using a bus is to eliminate the need for connecting every device with every other device, which avoids the wiring complexity that would quickly dominate the cost of such a system. Disadvantages of using a bus include the slowdown introduced by the master/slave configuration, the time involved in implementing a protocol (see below), and the lack of scalability to large sizes due to fan-out and timing constraints.

A bus can be classified as one of two types: **synchronous** or **asynchronous**. For a synchronous bus, one of the devices that is connected to the bus contains a crystal oscillator (a clock) that sends out a sequence of 1's and 0's at timed intervals as illustrated in Figure 8-20. The illustration shows a train of pulses that repeat at

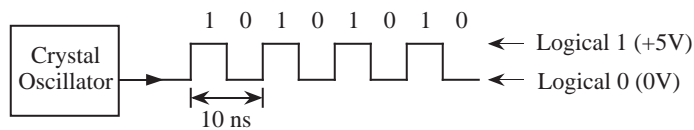


Figure 8-20 A 100 MHz bus clock.

10 ns intervals, which corresponds to a clock rate of 100 MHz. Ideally, the clock would be a perfect square wave (instantaneous rise and fall times) as shown in the figure. In practice, the rise and fall times are approximated by a rounded, trapezoidal shape.

Bus Clocking

For a synchronous bus, which is discussed below, a clock signal is used to syn-

chronize bus operations. This bus clock is generally derived from the master system clock, but it may be slowed down with respect to the master clock, especially in higher-speed CPUs. For example, one model of the Power Macintosh G3 computer has a system clock speed of 333 MHz, but a bus clock speed of 66 MHz, which is slower by a factor of 5. This is because memory access times are so much longer than typical CPU clock speeds. Typical cache memory has an access time of around 20 ns, compared to a 3 ns clock period for the processor described above.

In addition to the bus clock running at a slower speed than the processor, several bus clock cycles are usually required to effect a bus transaction, referred to as a **bus cycle**. Typical bus cycles run from two to five bus clock periods in duration.

The Synchronous Bus

As an example of how communication takes place over a synchronous bus, consider the timing diagram shown in Figure 8-21 which is for a synchronous read

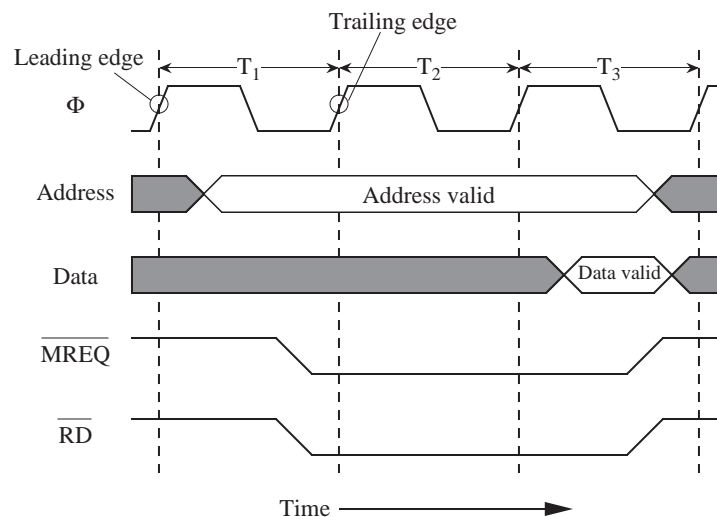


Figure 8-21 Timing diagram for a synchronous memory read (adapted from [Tanenbaum, 1999]).

of a word of memory by a CPU. At some point early in time interval T_1 , while the clock is high, the CPU places the address of the location it wants to read onto the address lines of the bus. At some later time during T_1 , after the voltages on the address lines have become stable, or “settled,” the \overline{MREQ} and \overline{RD} lines are asserted by the CPU. \overline{MREQ} informs the memory that it is selected for the

transfer (as opposed to another device, like a disk). The \overline{RD} line informs the selected device to perform a read operation. The overbars on \overline{MREQ} and \overline{RD} indicate that a 0 must be placed on these lines in order to assert them.

The read time of memory is typically slower than the bus speed, and so all of time interval T_2 is spent performing the read, as well as part of T_3 . The CPU assumes a fixed read time of three bus clocks, and so the data is taken from the bus by the CPU during the third cycle. The CPU then releases the bus by de-asserting \overline{MREQ} and \overline{RD} in T_3 . The shaded areas of the data and address portions of the timing diagram indicate that these signals are either invalid or unimportant at those times. The open areas, such as for the data lines during T_3 , indicate valid signals. Open and shaded areas are used with crossed lines at either end to indicate that the levels of the individual lines may be different.

The Asynchronous Bus

If we replace the memory on a synchronous bus with a faster memory, then the memory access time will not improve because the bus clock is unchanged. If we increase the speed of the bus clock to match the faster speed of the memory, then slower devices that use the bus clock may not work properly.

An asynchronous bus solves this problem, but is more complex, because there is no bus clock. A master on an asynchronous bus puts everything that it needs on the bus (address, data, control), and then asserts \overline{MSYN} (master synchronization). The slave then performs its job as quickly as it can, and then asserts \overline{SSYN} (slave synchronization) when it is finished. The master then de-asserts \overline{MSYN} , which signals the slave to de-assert \overline{SSYN} . In this way, a fast master/slave combination responds more quickly than a slow master/slave combination.

As an example of how communication takes place over an asynchronous bus, consider the timing diagram shown in Figure 8-22. In order for a CPU to read a word from memory, it places an address on the bus, followed by asserting \overline{MREQ} and \overline{RD} . After these lines settle, the CPU asserts \overline{MSYN} . This event triggers the memory to perform a read operation, which results in \overline{SSYN} eventually being asserted by the memory. This is indicated by the **cause-and-effect** arrow between \overline{MSYN} and \overline{SSYN} shown in Figure 8-22. This method of synchronization is referred to as a “full handshake.” In this particular implementation of a full handshake, asserting \overline{MSYN} initiates the transfer, followed by the slave asserting \overline{SSYN} , followed by the CPU de-asserting \overline{MSYN} , followed by

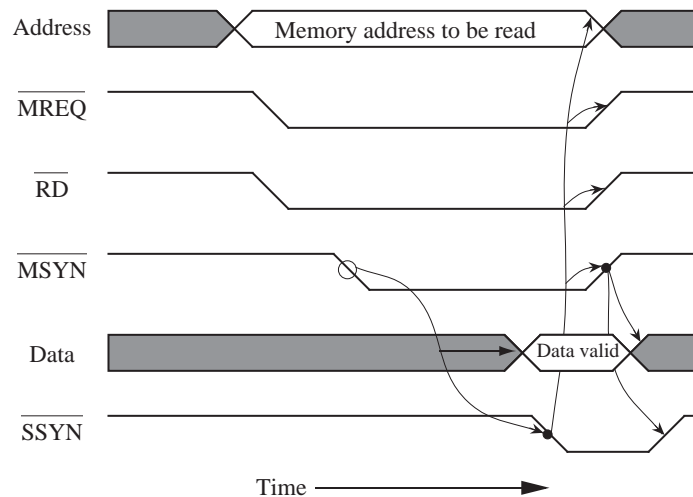


Figure 8-22 Timing diagram for asynchronous memory read (adapted from [Tanenbaum, 1999]).

the memory de-asserting \overline{SSYN} . Notice the absence of a bus clock signal.

Asynchronous busses are more difficult to debug than synchronous busses when there is a problem, and interfaces for asynchronous busses are more difficult to make. For these reasons, synchronous busses are very common, particularly in personal computers.

Bus Arbitration—Masters and Slaves

Suppose now that more than one device wants to be a bus master at the same time. How is a decision made as to who will be the bus master? This is the **bus arbitration** problem, and there are two basic schemes: **centralized** and **decentralized** (distributed). Figure 8-23 illustrates three organizations for these two schemes. In Figure 8-23a, a centralized arbitration scheme is used. Devices 0 through n are all attached to the same bus (not shown), and they also share a **bus request** line that goes into an **arbiter**. When a device wants to be a bus master, it asserts the bus request line. When the arbiter sees the bus request, it determines if a **bus grant** can be issued (it may be the case that the current bus master will not allow itself to be interrupted). If a bus grant can be issued, then the arbiter asserts the bus grant line. The bus grant line is **daisy chained** from one device to the next. The first device that sees the asserted bus grant and also wants to be the bus master takes control of the bus and does not propagate the bus grant to higher numbered devices. If a device does not want the bus, then it simply passes the

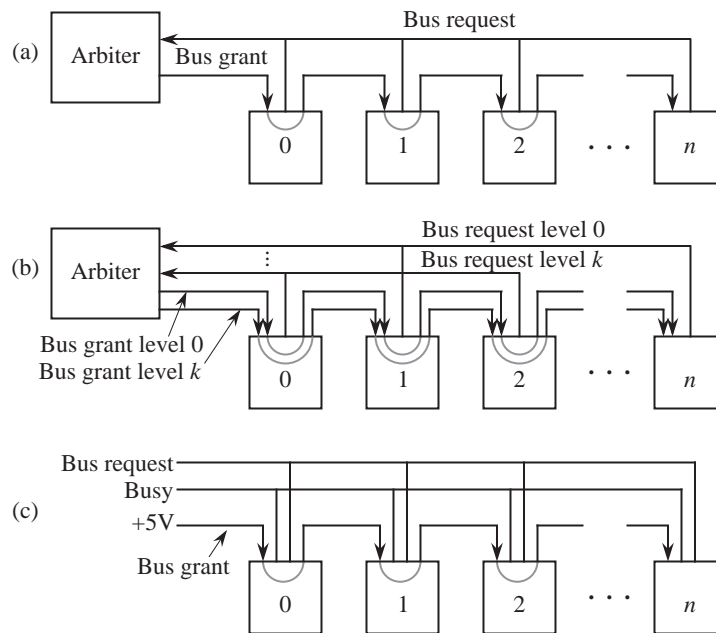


Figure 8-23 (a) Simple centralized bus arbitration; (b) centralized arbitration with priority levels; (c) decentralized bus arbitration. (Adapted from [Tanenbaum, 1999]).

bus grant to the next device. In this way, devices that are electrically closer to the arbiter have higher priorities than devices that are farther away.

Sometimes an absolute priority ordering is not appropriate, and a number of bus request/bus grant lines are used as shown in Figure 8-23(b). Lower numbered bus request lines have higher priorities than higher numbered bus request lines. In order to raise the priority of a device that is far from the arbiter, it can be assigned to a lower numbered bus request line. Priorities are assigned within a group on the same bus request level by electrical proximity to the arbiter.

In a third approach, a decentralized bus arbitration scheme is used as illustrated in Figure 8-23(c). Notice the lack of a central arbiter. A device that wants to become a bus master first asserts the bus request line, and then it checks if the bus is busy. If the busy line is not asserted, then the device sends a 0 to the next higher numbered device on the daisy chain, asserts the busy line, and de-asserts the bus request line. If the bus is busy, or if a device does not want the bus, then it simply propagates the bus grant to the next device.

Arbitration needs to be a fast operation, and for that reason, a centralized scheme

will only work well for a small number of devices (up to about eight). For a large number of devices, a decentralized scheme is more appropriate.

Given a system that makes use of one of these arbitration schemes, imagine a situation in which n card slots are used, and then card m is removed, where $m < n$. What happens? Since each bus request line is directly connected to all devices in a group, and the bus grant line is passed through each device in a group, a bus request from a device with an index greater than m will never see an asserted bus grant line, which can result in a system crash. This can be a frustrating problem to identify, because a system can run indefinitely with no problems, until the higher numbered device is accessed.

When a card is removed, higher cards should be repositioned to fill in the missing slot, or a dummy card that continues the bus grant line should be inserted in place of the removed card. Fast devices (like disk controllers) should be given higher priority than slow devices (like terminals), and should thus be placed close to the arbiter in a centralized scheme, or close to the beginning of the Bus grant line in a decentralized scheme.

8.4.2 COMMUNICATION BETWEEN PROCESSORS AND MEMORIES

Computer systems have a wide range of communication tasks. The CPU must communicate with memory, and with a wide range of I/O devices, ranging from extremely slow devices such as keyboards, to high-speed devices like disk drives and network interfaces. In fact, there may be multiple CPUs that communicate with one another either directly or through a shared memory, in a typical configuration.

Three methods for managing input and output are **programmed I/O** (also known as **polling**), **interrupt driven I/O**, and **direct memory access** (DMA).

Programmed I/O

Consider reading a block of data from a disk. In programmed I/O, the CPU polls each device to see if it needs servicing. In a restaurant analogy, the host would approach the patron and ask if the patron is ready.

The operations that take place for programmed I/O are shown in the flowchart in Figure 8-24. The CPU first checks the status of the disk by reading a special

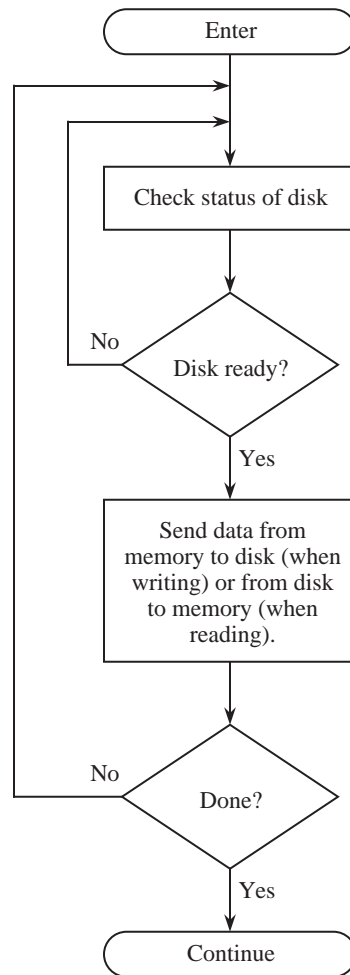


Figure 8-24 Programmed I/O flowchart for a disk transfer.

register that can be accessed in the memory space, or by issuing a special I/O instruction if this is how the architecture implements I/O. If the disk is not ready to be read or written, then the process loops back and checks the status continuously until the disk is ready. This is referred to as a **busy-wait**. When the disk is finally ready, then a transfer of data is made between the disk and the CPU.

After the transfer is completed, the CPU checks to see if there is another communication request for the disk. If there is, then the process repeats, otherwise the CPU continues with another task.

In programmed I/O the CPU wastes time polling devices. Another problem is that high priority devices are not checked until the CPU is finished with its current I/O task, which may have a low priority. Programmed I/O is simple to implement, however, and so it has advantages in some applications.

Interrupt-driven I/O

With interrupt driven I/O, the CPU does not access a device until it needs servicing, and so it does not get caught up in busy-waits. In interrupt-driven I/O, the device requests service through a special interrupt request line that goes directly to the CPU. The restaurant analogy would have the patron politely tapping silverware on a water glass, thus interrupting the waiter when service is required.

A flowchart for interrupt driven I/O is shown in Figure 8-25. The CPU issues a request to the disk for reading or for writing, and then immediately resumes execution of another process. At some later time, when the disk is ready, it interrupts the CPU. The CPU then invokes an **interrupt service routine** (ISR) for the disk, and returns to normal execution when the interrupt service routine completes its task. The ISR is similar in structure to the procedure presented in Chapter 4, except that interrupts occur asynchronously with respect to the process being executed by the CPU: an interrupt can occur at any time during program execution.

There are times when a process being executed by the CPU should not be interrupted because some critical operation is taking place. For this reason, instruction sets include instructions to disable and enable interrupts under programmed control. (The waiter can ignore the patron at times.) Whether or not interrupts are accepted is generally determined by the state of the Interrupt Flag (IF) which is part of the Processor Status Register. Furthermore, in most systems priorities are assigned to the interrupts, either enforced by the processor or by a **peripheral interrupt controller** (PIC). (The waiter may attend to the head table first.) At the top priority level in many systems, there is a **non-maskable interrupt** (NMI) which, as the name implies, cannot be disabled. (The waiter will in all cases pay attention to the fire alarm!) The NMI is used for handling potentially catastrophic events such as power failures, and more ordinary but crucially uninteruptible operations such as file system updates.

At the time when an interrupt occurs (which is sometimes loosely referred to as a **trap**, even though traps usually have a different meaning, as explained in Chap-

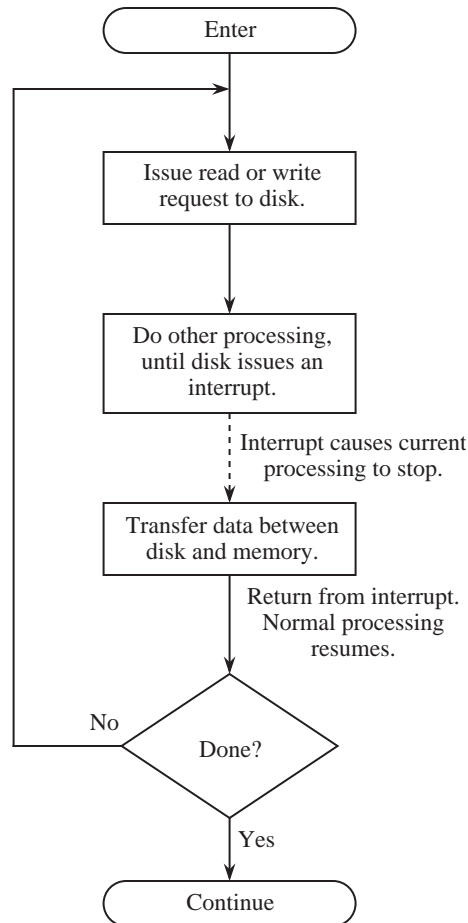


Figure 8-25 Interrupt driven I/O flowchart for a disk transfer.

ter 6), the Processor Status Register and the Program Counter (`%psr` and `%pc` for the ARC) are automatically pushed onto the stack, and the Program Counter is loaded with the address of the appropriate interrupt service routine. The processor status register is pushed onto the stack because it contains the interrupt flag (IF), and the processor must disable interrupts for at least the duration of the first instruction of the ISR. (Why?) Execution of the interrupt routine then begins. When the interrupt service routine finishes, execution of the interrupted program then resumes.

The ARC `jmp1` instruction (see Chapter 5) will not work properly for resuming execution of the interrupted routine, because in addition to restoring the pro-

gram counter contents, the processor status register must be restored. Instead, the `rett` (return from trap) instruction is invoked, which reverses the interrupt process and restores the `%psr` and `%pc` registers to their values prior to the interrupt. In the ARC architecture, `rett` is an arithmetic format instruction with `op3 = 111001`, and an unused `rd` field (all zeros).

Direct Memory Access (DMA)

Although interrupt driven I/O frees the CPU until the device requires service, the CPU is still responsible for making the actual data transfer. Figure 8-26 high-

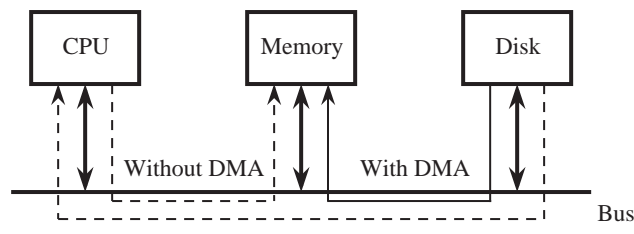


Figure 8-26 DMA transfer from disk to memory bypasses the CPU.

lights the problem. In order to transfer a block of data between the memory and the disk using either programmed I/O or interrupt driven I/O, every word travels over the bus twice: first to the CPU, then again over the bus to its destination.

A DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus improve the speed of communication over the bus. In keeping with the restaurant analogy, the host serves everyone at one table before serving anyone at another table. DMA services are usually provided by a DMA controller, which is itself a specialized processor whose specialty is transferring data directly to or from I/O devices and memory. Most DMA controllers can also be programmed to make memory-to-memory block moves. A DMA device thus takes over the job of the CPU during a transfer. In setting up the transfer, the CPU programs the DMA device with the starting address in main memory, the starting address in the device, and the length of the block to be transferred.

Figure 8-27 illustrates the DMA process for a disk transfer. The CPU sets up the DMA device and then signals the device to start the transfer. While the transfer is taking place, the CPU continues execution of another process. When the DMA transfer is completed, the device informs the CPU through an interrupt. A sys-

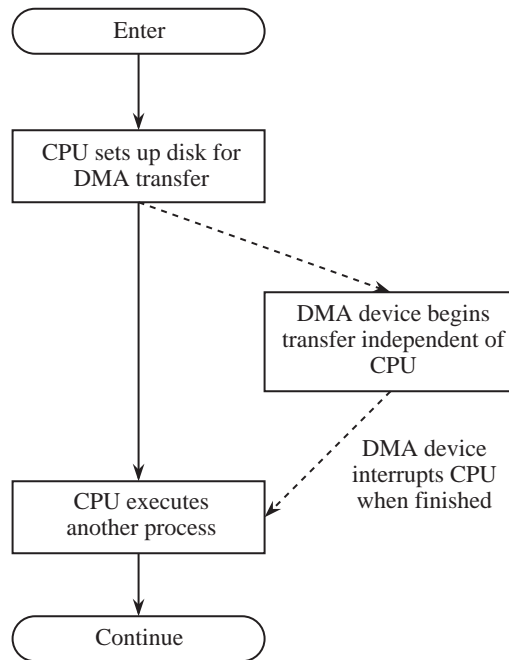


Figure 8-27 DMA flowchart for a disk transfer.

tem that implements DMA thus also implements interrupts as well.

If the DMA device transfers a large block of data without relinquishing the bus, the CPU may become starved for instructions or data, and thus its work is halted until the DMA transfer has completed. In order to alleviate this problem, DMA controllers usually have a “cycle-stealing” mode. In **cycle-stealing DMA** the controller acquires the bus, transfers a single byte or word, and then relinquishes the bus. This allows other devices, and in particular the CPU, to share the bus during DMA transfers. In the restaurant analogy, a patron can request a check while the host is serving another table.

8.4.3 I/O CHANNELS

The DMA concept is an efficient method of transferring blocks of data over a bus, but there is a need for a more sophisticated approach for complex systems. There are a number of reasons for not connecting I/O devices directly to the system bus:

- The devices might have complex operating characteristics, and the CPU should

be insulated from this complexity.

- Peripherals might be slow, and since the system bus is fast, overall performance is degraded if direct access to the system bus is allowed by all devices.
- Peripherals sometimes use different data formats and word lengths than the CPU (such as serial *vs.* parallel, byte *vs.* word, *etc.*)

I/O for complex systems can be handled through an **I/O channel**, or **I/O module**, that interfaces peripheral devices to the system bus. An I/O channel is a high level controller that can execute a computer program, which is its distinguishing characteristic. This program might seek a head across a disk, or collect characters from a number of keyboards into a block and transmit the block using DMA.

There are two types of channels, as illustrated in Figure 8-28. A **selector** channel

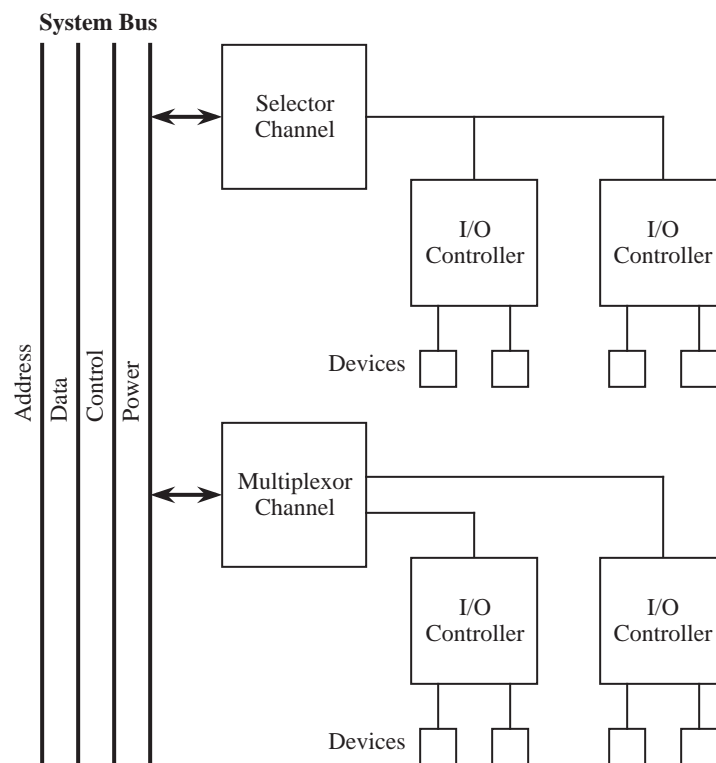


Figure 8-28 A selector channel and a multiplexor channel.

controls several devices, but handles transfers for a single device at a time. A selector channel is typically used for high speed devices like hard disks. A **multiplexor** channel handles transfers for several devices at a time. A multiplexor channel comes in two forms: a **byte multiplexor**, which interleaves bytes from a number of low speed devices into a single stream, or a **block multiplexor**, which interleaves blocks from a number of high speed devices into a single stream.

For both types of channels, concurrent operations can take place among devices and the rest of the system. For instance, a selector channel may perform a head seek operation while a multiplexor channel performs a block transfer over the system bus. Only a single block at a time can be transferred over the system bus, however.

8.4.4 MODEMS

People communicate over telephone lines by forming audible sounds that are converted to electrical signals, which are transmitted to a receiver where they are converted back to audible sounds. This does not mean that people always need to speak and hear in order to communicate over a telephone line: this audible medium of communication can also be used to transmit non-audible information that is converted to an audible form.

Figure 8-29 shows a configuration in which two computers communicate over a

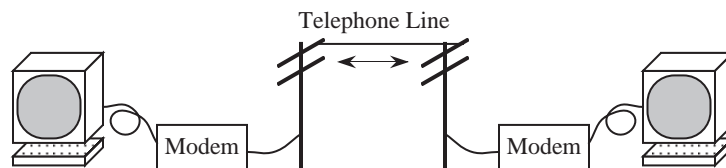


Figure 8-29 Communication over a telephone line with modems.

telephone line through the use of **modems** (which is a contraction of modulator / demodulator). A modem transforms an electrical signal from a computer into an audible form for transmission, and performs the reverse operation when receiving.

Modem communication over a telephone line is normally performed in serial fashion, a single bit at a time, in which the bits have an encoding that is appropriate for the transmission medium. There are a number of forms of **modulation** used in communication, which are encodings of data into the medium. Figure

8-30 illustrates three common forms of modulation.

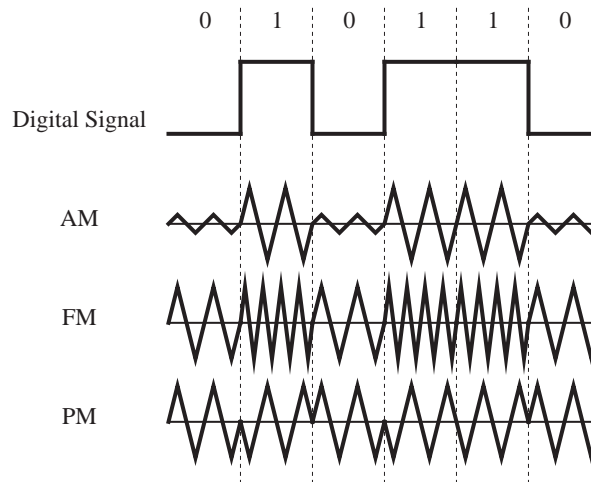


Figure 8-30 Three common forms of modulation.

Amplitude modulation (AM) uses the strength of the signal to encode 1's and 0's. AM lends itself to simple implementations that are inexpensive to build. However, since there is information in the amplitude of the signal, anything that changes the amplitude affects the signal. For an AM radio, a number of situations affect the amplitude of the signal (such as driving under a bridge or near electrical lines, lightning, *etc.*).

Frequency modulation (FM) is not nearly as sensitive to amplitude related problems because information is encoded in the frequency of the signal rather than in the amplitude. The FM signal on a radio is relatively static-free, and does not diminish as the receiver passes under a bridge.

Phase modulation (PM) is most typically used in modems, where four phases (90 degrees apart) double the data bandwidth by transmitting two bits at a time (which are referred to as **dibits**). The use of phase offers a degree of freedom in addition to frequency, and is appropriate when the number of available frequencies is restricted.

In **pulse code modulation** (PCM) an analog signal is sampled and converted into binary. Figure 8-31 shows the process of converting an analog signal into a PCM binary sequence. The original signal is sampled at twice the rate of the highest significant frequency, which produces values at discrete intervals. The

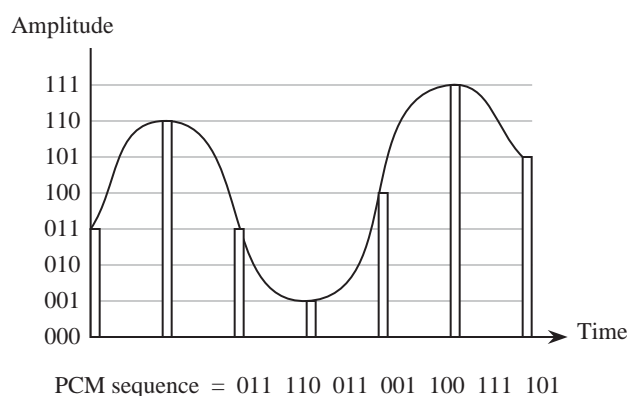


Figure 8-31 Conversion of an analog signal to a PCM binary sequence.

samples are encoded in binary and catenated to produce the PCM sequence.

PCM is a digital approach, and has all of the advantages of digital information systems. By using repeaters at regular intervals the signal can be perfectly restored. By decreasing the distance between repeaters, the effective bandwidth of a channel can be significantly increased. Analog signals, however, can at best be guessed and can only be approximately restored. There is no good way to make analog signals perfect in a noisy environment.

Shannon's result about the data rate of a noisy channel applies here:

$$\text{data rate} = \text{bandwidth} \times \log(1 + S/N)$$

where S is the signal and N is the noise. Since a digital signal can be made to use arbitrarily noisy channels (in which S/N is large) because of its noise immunity, higher data rates can be achieved over the same channel. This is one of the driving forces in the move to digital technology in the telecommunications industry. The transition to all-digital has also been driven by the rapid drop in the cost of digital circuitry.

8.4.5 LOCAL AREA NETWORKS

A **local area network** (LAN) is a communication medium that interconnects computers over a limited geographical distance of a few miles at most. A LAN allows a set of closely grouped computers and other devices to share common resources such as data, software applications, printers, and mass storage.

A LAN consists of hardware, software, and protocols. The hardware may be in the form of cables and interface circuitry. The software is typically embedded in an operating system, and is responsible for connecting a user to the network. The protocols are sets of rules that govern format, timing, sequencing, and error control. Protocols are important for ensuring that data is packaged for injection into the network and is extracted from the network properly. The data to be transmitted is decomposed into pieces, each of which is prepended with a **header** that contains information about parameters such as the destination, the source, error protection bits, and a time stamp. The data, which is often referred to as the **payload**, is combined with the header to form a **packet** that is injected into the network. A receiver goes through the reverse process of extracting the data from the packet.

The process of communicating over a network is normally carried out in a hierarchy of steps, each of which has its own protocol. The steps must be followed in sequence for transmission, and in the reverse sequence when receiving. This leads to the notion of a **protocol stack** which isolates the protocol being used within the hierarchy.

The OSI Model

The **Open System Interconnection** (OSI) model is a set of protocols established by the International Standards Organization (ISO) in an attempt to define and standardize data communications. The OSI model has been largely displaced by the Internet TCP/IP model (see Chapter 10) but still heavily influences network communication, particularly for telecommunication companies.

In the OSI model the communication process is divided into seven layers: **application**, **presentation**, **session**, **transport**, **network**, **data link**, and **physical** as summarized in Figure 8-32. As an aid in remembering the layers, the mnemonic is sometimes used: **A** **P**owered-down **S**ystem **T**ransmits **N**o **D**ata **P**ackets.

The OSI model does not give a single definition of how data communications actually take place. Instead, the OSI model serves as a reference for how the process should be divided and what protocols should be used at each layer. The concept is that equipment providers can select a protocol for each layer while ensuring compatibility with equipment from other providers that may use different protocols.

The highest level in the OSI model is the application layer, which provides an

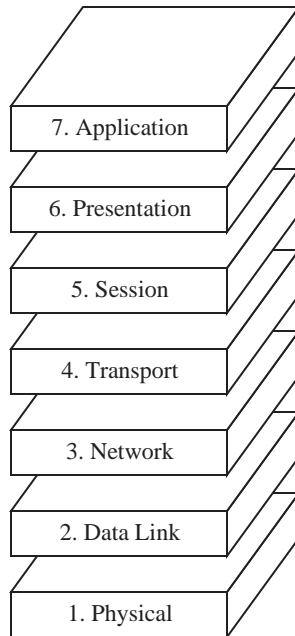


Figure 8-32 The seven layers of the OSI model.

interface for allowing applications to communicate with each other over the network. It offers high level support for applications that interact over the network such as database services for network database programs, message handling for **electronic mail** (e-mail) programs, and file handling for file transfer programs.

The presentation layer ensures that information is presented to communication applications in a common format. This is necessary because different systems may use different internal data formats. For instance, some systems use a big-endian internal format while others use a little-endian internal format. The function of the presentation layer is to insulate the applications from these differences.

The session layer establishes and terminates communication sessions between host processes. The session layer is responsible for maintaining the integrity of communication even if the layers below it lose data. It also synchronizes the exchange, and establishes reference points for continuing an interrupted communication.

The transport layer ensures reliable transmission from source to destination. It

allocates communication resources so that data is transferred both quickly and cost effectively. The session layer makes requests to the transport layer, which prioritizes the requests and makes trade-offs among speed, cost, and capacity. For example, a transmission may be split into several packets which are transmitted over a number of networks in order to obtain a faster communication time. Packets may thus arrive at the destination out of order, and it is the responsibility of the transport layer to ensure that the session layer receives data in the same order it is sent. The transport layer provides error recovery from source to destination, and also provides flow control (that is, it ensures that the speeds of the sender and receiver are matched).

The network layer routes data through intermediate systems and subnetworks. Unlike the upper layers, the network layer is aware of the network **topology**, which is the connectivity among the network components. The network layer informs the transport layer of the status of potential and existing connections in the network in terms of speed, reliability, and availability. The network layer is typically implemented with **routers**, which connect different networks that use the same transport protocol.

The data link layer manages the direct connections between components on a network. This layer is divided into the **logical link control** (LLC) which is independent of the network topology, and the **media access control** (MAC) which is specific to the topology. In some networks the physical connections between devices are not permanent, and it is the responsibility of the data link layer to inform the physical layer when to make connections. This layer deals in units of **frames** (single packets, or collections of packets that may be interleaved), which contain addresses, data, and control information.

The physical layer ensures that raw data is transmitted from a source to a destination over the physical medium. It transmits and repeats signals across network boundaries. The physical layer does *not* include the hardware itself, but includes methods of accessing the hardware.

Topologies

There are three primary LAN organizations, as illustrated in Figure 8-33. The **bus** topology is the simplest of the three. Components are connected to a bus system by simply plugging them into the single cable that runs through the network, or in the case of a wireless network, by simply emitting signals into a common medium. An advantage to this type of topology is that each component can

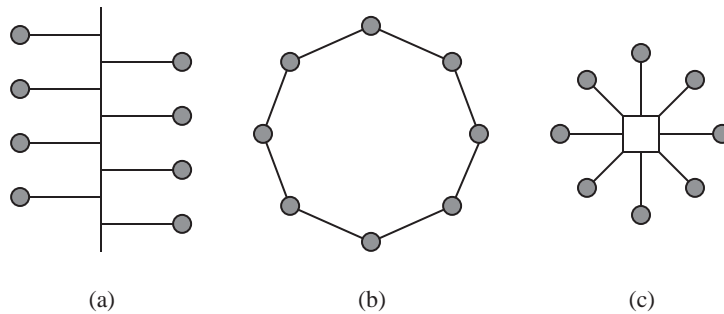


Figure 8-33 (a) bus; (b) ring; and (c) star network topologies.

communicate directly with any other component on the bus, and that it is relatively simple to add another component to the network. Control is distributed among the components, and so there is no single network component that serves as an intermediary, which reduces the initial cost of this type of network. Disadvantages of this topology include a limit on the length of the cable from the bus to each network component (for a wireline network) and that a break in the cable may be needed in order to add another component to the network, which disrupts the rest of the network. An example of a bus-based network is Ethernet.

The **ring** topology uses a single cable, in which the ends are joined. Packets are passed around the ring through each network component until they reach their destinations. At the destinations, the packets are extracted from the network and are not passed farther along the ring. If a packet makes its way back to the originating system, then the transmission is unsuccessful, and so the packet is stopped and a new transmission can be attempted. An example of a ring-based LAN is IBM's Token Ring.

In a **star** topology, each component is connected to a central **hub** which serves as an intermediary for all communication over the network. In a simple configuration, the hub receives data from one component and forwards it to all of the other components, leaving it to the individual components to determine whether or not they are the intended target. In a more sophisticated configuration, the hub receives data and forwards it to a specific network component.

An advantage of a star topology is that most of the network service, troubleshooting, and wiring changes take place at the central hub. A disadvantage is that a problem with the hub affects the entire network. Another disadvantage is that geometrically, the star topology requires more cable than a bus or a ring because a separate cable connects each network component to the hub. An example of a

star-based network is ARCnet (although it is actually a bus-based network).

Data Transmission

Communication within a computer is synchronized by a common clock, and so the transmission of a 1 or a 0 is signalled by a high or low voltage that is sampled at a time determined by the clock. This scheme is simple, but does not work well over longer distances, as in a LAN. The problem is that there is no timing reference to signal the start or stop of a bit. When there is a long string of 1's or 0's, timing with respect to the sending and receiving clocks may drift because the clocks are not precisely synchronized. The distances over a LAN are too great to maintain both a global clock and high speed at the same time. LANs thus typically use the Manchester encoding scheme (see Section 8.1.1), in which timing is embedded in the data.

Manchester encoding is applied at the lowest level of transmission. At the next level, a data stream is decomposed into packets and frames that are transmitted over the network, not necessarily in order. The data link layer is responsible for decomposing a data stream into packets, forming packets into frames, and injecting frames into the network. When receiving frames, the data link layer extracts the packets and assembles them into a format that the higher level network layers can use. The size of a data packet is commonly on the order of a kilobyte, and requires a few microseconds for transmission at typical speeds and distances.

Ethernet is one of the most prevalent bus-based networks. Ethernet uses **carrier sense multiple access** with **collision detection** (CSMA/CD) for transmission. Under CSMA/CD, when a network component wants to transmit data, it first listens for a carrier. If there is a carrier present on the line, which is placed there by a transmitting device, then it transmits nothing and listens again after a random waiting period. The random waiting period is important in order to avoid a **deadlock** in which components that are trying to access the bus perpetually listen and wait in synchrony.

If there is no traffic on the line, then transmission can begin by sending a carrier on the line as well as the data. The source also listens for **collisions**, in which two or more components simultaneously transmit. A collision is detected by the presence of more than one carrier. Collisions can occur in a fully operational network as a result of the finite time it takes for a signal to travel the length of the bus. The propagation of signals on the bus is bounded by the speed of light over the length of the bus, which can be 500 m in a generic Ethernet installation. When a

collision occurs, the transmitting components wait for a random interval before retransmitting.

Transmitted data moves in both directions over the bus. Every component sees every packet of data, but only extracts those packets with corresponding destination addresses. After a packet is successfully delivered, the destination can generate an acknowledgment to the sender, typically at the transport layer. If the sender does not receive an acknowledgment after a fixed period of time (which must be greater than the round trip delay through the network), then it retransmits the message.

Collisions should occur infrequently in practice, and so the overhead of recovering from a collision is not very significant. A serious degradation in Ethernet performance does not occur until traffic increases to about 35% of network capacity.

Bridges, Routers, and Gateways

As networks grow in size, they can be subdivided into smaller networks that are interconnected. The smaller **subnetworks** operate almost entirely independently of each other, and can use different protocols and topologies.

If the subnetworks all use the same topology and the same protocols, then it may be the case that all that is needed to extend the network are **repeaters**. A repeater amplifies the signals on the network, which become attenuated in proportion to the distance traveled. The overall network is divided into subnetworks, in which each subnetwork operates somewhat independently with respect to the others. The subnetworks are not entirely independent because every subnetwork sees all of the traffic that occurs on the other subnetworks. A network with simple repeaters is not extensible to large sizes. Since noise is amplified as well as the signal, the noise will eventually dominate the signal if too many repeaters are used in succession.

A **bridge** does more than simply amplify voltage levels. A bridge restores the individual voltage levels to logical 1 or 0, which prevents noise from accumulating. Bridges have some level of intelligence, and can typically interpret the destination address of a packet and route it to the appropriate subnetwork. In this way, network traffic can be reduced, since the alternative method would be to blindly send each incoming packet to each subnetwork (as for a repeater based network).

Although bridges have some level of intelligence in that they sense the incoming bits and make routing decisions based on destination addresses, they are unaware of protocols. A **router** operates at a higher level, in the network layer. Routers typically connect logically separate networks that use the same transport protocol.

A **gateway** translates packets up through the application layer of the OSI model (layers 4 through 7). Gateways connect dissimilar networks by performing protocol conversions, message format conversions, and other high level functions.

8.5 Communication Errors and Error Correcting Codes

In all computer architectures, and especially in situations involving communications between computers, there is a finite chance that the data is received in error, due to noise in the communication channel. The data representations we have considered up to this point make use of the binary symbols 1 and 0. In reality, the binary symbols take on physical forms such as voltages or electric current. The physical form is subject to noise that is introduced from the environment, such as atmospheric phenomena, gamma rays, and power fluctuations, to name just a few. The noise can cause errors, also known as **faults**, in which a 0 is turned into a 1 or a 1 is turned into a 0.

Suppose that the ASCII character 'b' is transmitted from a sender to a receiver, and during transmission, an error occurs, so that the least significant bit is inverted. The correct bit pattern for ASCII 'b' is 1100010. The bit pattern that the receiver sees is 1100011, which corresponds to the character 'c.' There is no way for the receiver to know that an error occurred simply by looking at the received character. The problem is that all of the possible 2^7 ASCII bit patterns represent valid characters, and if any of the bit patterns is transformed into another through an error, then the resulting bit pattern appears to be valid.

It is possible for the sender to transmit additional "check bits" along with the data bits. The receiver can examine these check bits and under certain conditions not only detect errors, but correct them as well. Two methods of computing these additional bits are described below. We start by introducing some preliminary information and definitions.

8.5.1 BIT ERROR RATE DEFINED

There are many different ways that errors can be introduced into a computer sys-

tem, and those errors can take many different forms. For the moment, we will assume that the probability that a given bit is received in error is independent of the probability that other bits near it are received in error. In this case, we can define the **bit error rate** (BER) as the probability that a given bit is erroneous. Obviously this must be a very small number, and is usually less than 10^{-12} errors per bit examined for many networks. That means, loosely speaking, that as bits are examined, only one in every 10^{12} bits will be erroneous.

Inside the computer system typical BER's may run 10^{-18} or less. As a rough estimate, if the clock rate of the computer is 100 MHz, and 32 bits are manipulated during each clock period, then the number of errors per second for that portion of the computer will be $10^{-18} \times 100 \times 10^6 \times 32$ or 3.2×10^{-9} errors per second, approximately one erroneous bit once every 10 years.

On the other hand, if one is receiving a bit stream from a serial communications line at, say, 1 million bits per second, and the BER is 10^{-10} , then the number of errors per second will be $1 \times 10^6 \times 10^{-10}$ or 10^{-4} errors per second, approximately 10 errors per day.

8.5.2 ERROR DETECTION AND CORRECTION

One of the simplest and oldest methods of error detection was used to detect errors in transmitting and receiving characters in telegraphy. A **parity bit**, 1 or 0, was added to each character to make the total number of 1's in the character even or odd, as agreed upon by sender and receiver. In our example of transmitting the ASCII character 'b,' 1100010, assuming even parity, a 1 would be attached as a parity bit to make the total number of 1's even, resulting in the bit pattern 11000101 being transmitted. The receiver could then examine the bit pattern, and if there was an even number of 1's, the receiver could assume that the character was received without error. (This method fails if there is significant probability of two or more bits being received in error. In this case, other methods must be used, as discussed later in this section.) The intuition behind this approach is explored below.

Hamming Codes

If additional bits are added to the data then it is possible to not only detect errors, but to correct them as well. Some of the most popular error-correcting codes are based on the work of Richard Hamming of Bell Telephone Laboratories (now operated by Lucent Technologies).

We can detect single-bit errors in the ASCII code by adding a redundant bit to each **codeword** (character). The **Hamming distance** defines the logical distance between two valid codewords, as measured by the number of digits that differ between the codewords. If a single bit changes in an ASCII character, then the resulting bit pattern represents a different ASCII character. The corresponding Hamming distance for this code is 1. If we recode the ASCII table so that there is a Hamming distance of 2 between valid codewords, then two bits must change in order to convert one character into another. We can then detect a single-bit error because the corrupted word will lie between valid codewords.

One way to recode ASCII for a Hamming distance of two is to assign a parity bit, which takes on a value of 0 or 1 to make the total number of 1's in a codeword odd or even. If we use **even parity**, then the parity bit for the character 'a' is 1 since there are three 1's in the bit pattern for 'a': 1100001 and assigning a parity bit of 1 (to the left of the codeword here) makes the total number of 1's in the recoded 'a' even: 11100001. This is illustrated in Figure 8-34. Similarly, the par-

Bit position

P	6	5	4	3	2	1	0	
1	1	1	0	0	0	0	1	a
1	1	1	0	0	0	1	0	b
0	1	1	0	0	0	1	1	c
1	1	1	1	1	0	1	0	z
0	1	0	0	0	0	0	1	A

↑
7-bit ASCII character code
↑

Even parity bit
Character

Figure 8-34 Even parity bits are assigned to a few ASCII characters.

ity bit for 'c' is 0 which results in the recoded bit pattern: 01100011. If we use odd parity instead, then the parity bits take on the opposite values: 0 for 'a' and 1 for 'c,' which results in the recoded bit patterns 01100001 and 11100011, respectively.

The recoded ASCII table now has $2^8 = 256$ entries, of which half of the entries

(the ones with an odd number of 1's) represent invalid codewords. If an invalid codeword is received, then the receiver knows that an error occurred and can request a retransmission.

A retransmission may not always be practical, and for these cases it would be helpful to both detect and correct an error. The use of a parity bit will detect an error, but will not locate the position of an error. If the bit pattern 11100011 is received in a system that uses even parity, then the presence of an error is known because the parity of the received word is odd. There is not enough information from the parity bit alone to determine if the original pattern was 'a', 'b', or any of five other characters in the ASCII table. In fact, the original character might even be 'c' if the parity bit itself is in error.

In order to construct an error correcting code that is capable of detecting and correcting single-bit errors, we must add more redundancy than a single parity bit provides to the ASCII code by further extending the number of bits in each codeword. For instance, consider the bit pattern for 'a': 1100001. If we wish to detect and correct a single bit error in any position of the word, then we need to assign seven additional bit patterns to 'a' in which exactly one bit changes in the original 'a' codeword: 0100001, 1000001, 1110001, 1101001, 1100101, 1100011, and 1100000. We can do the same for 'b' and the remaining characters, but we must construct the code in such a way that no bit pattern is common to more than one ASCII character, otherwise we will have no means to unambiguously determine the original bit pattern.

A problem with using redundancy in this way is that we assign eight bit patterns to every character: one for the original bit pattern, and seven for the neighboring error patterns. Since there are 2^7 characters in the ASCII code, and since we need 2^3 bit patterns for every character, then we can only recode $2^7/2^3 = 2^4$ characters if we use only the original seven bits in the representation.

In order to recode all of the characters, we must add additional **redundant bits** (also referred to as **check bits**) to the codewords. Let us now determine how many bits we need. If we start with a k -bit word that we would like to recode, and we use r check bits, then the following relationship must hold:

$$2^k \times (k + r + 1) \leq 2^{k+r} \equiv k + r + 1 \leq 2^r \quad (8.1)$$

The reasoning behind this relationship is that for each of the 2^k original words, there are k bit patterns in which a single bit is corrupted in the original word,

plus r bit patterns in which one of the check bits is in error, plus the original uncorrupted bit pattern. Thus, our error correcting code will have a total of $2^k \times (k + r + 1)$ bit patterns. In order to support all of these bit patterns, there must be enough bit patterns generated by $k + r$ bits, thus 2^{k+r} must be greater than or equal to the number of bit patterns in the error correcting code. There are $k = 7$ bits in the ASCII code, and so we must now solve for r . If we try a few successive values, starting at 1, we find that $r = 4$ is the smallest value that satisfies relation 8.1. The resulting codewords will thus have $7 + 4 = 11$ bits.

We now consider how to recode the ASCII table into the 11-bit code. Our goal is to assign the redundant bits to the original words in such a way that any single-bit error can be identified. One way to make the assignment is shown in Figure 8-35. Each of the 11 bits in the recoded word are assigned a position in the

Check bits C8 C4 C2 C1				Bit position checked
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11

Figure 8-35 Check bits for a single error correcting ASCII code.

table indexed from 1 to 11, and the 4-bit binary representations of the integers 1 through 11 are shown next to each index. With this assignment, reading across each of the 11 rows of four check bits, there is a unique positioning of the 1 bits in each row, and so no two rows are the same. For example, the top row has a single 1 in position C1, but no other row has only a single 1 in position C1 (other rows have a 1 in position C1, but they also have 1's in the other check bit positions.)

Now, reading down each of the four check bit columns, the positions of the 1 bits tell us which bits, listed in the rightmost 'Bit position checked' column, are included in a group that must form even parity. For example, check bit C8 covers a group of 4 bits in positions 8, 9, 10, and 11, that collectively must form even parity. If this property is satisfied when the 11-bit word is transmitted, but an

error in transmission causes this group of bits to have odd parity at the receiver, then the receiver will know that there must be an error in either position 8, 9, 10, or 11. The exact position can be determined by observing the remaining check bits, as we will see.

In more detail, each bit in the 11-bit encoded word, which includes the check bits, is assigned to a unique combination of the four check bits C1, C2, C4, and C8. The combinations are computed as the binary representation of the position of the bit being checked, starting at position 1. C1 is thus in bit position 1, C2 is in position 2, C4 is in position 4, *etc.* The check bits can appear anywhere in the word, but normally appear in positions that correspond to powers of 2 in order to simplify the process of locating an error. This particular code is known as a **single error correcting (SEC)** code.

Since the positions of the 1's in each of the check bit combinations is unique, we can locate an error by simply observing which of the check bits are in error. Consider the format shown in Figure 8-36 for the ASCII character 'a'. The values of

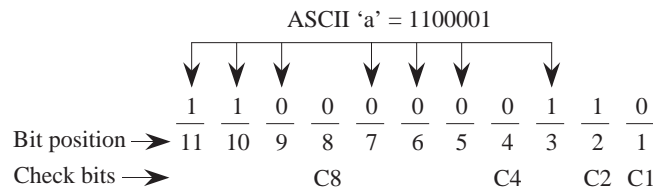


Figure 8-36 Format for a single error correcting ASCII code.

the check bits are determined according to the table shown in Figure 8-35. Check bit C1 = 0 creates even parity for the bit group {1, 3, 5, 7, 9, 11}. The members in this group are taken from the positions that have 1's in the C1 column in Figure 8-35. Check bit C2 = 1 creates even parity for the bit group {2, 3, 6, 7, 10, 11}. Similarly, check bit C4 = 0 creates even parity for the bit group {4, 5, 6, 7}. Finally, check bit C8 = 0 creates even parity for the bit group {8, 9, 10, 11}.

As an alternative to looking up members of a parity group in a table, in general, bit n of the coded word is checked by those check bits in positions b_1, b_2, \dots, b_j such that $b_1 + b_2 + \dots + b_j = n$. For example, bit 7 is checked by bits in positions 1, 2, and 4 because $1 + 2 + 4 = 7$.

Now suppose that a receiver sees the bit pattern 10010111001. Assuming that the SEC code for ASCII characters described above is used, what character was

opposing vertices. Any single bit error will locate an invalid codeword at a different vertex on the cube. Every error codeword has a closest valid codeword, which makes single error correction possible.

SECDED Encoding

If we now consider the case in which there are two errors, then we can see that the SEC code works for **double error detection (DED)**, but not for **double error correction (DEC)**. This is sometimes referred to as **SECDED** encoding. Since valid codewords are spaced at a Hamming distance of 3, two errors will locate an error codeword on the cube, and thus two errors can be detected. The original codeword cannot be determined unambiguously, however, since vertices that correspond to two errors from one codeword overlap vertices that correspond to a single error from another codeword. Thus, every SEC code is also a DED code, but every DED code is not necessarily a DEC code. In order to correct two errors, a Hamming distance of five must be maintained. In general, a Hamming distance of $p + 1$ must be maintained in order to detect p errors, and a Hamming distance of $2p + 1$ must be maintained to correct p errors.

8.5.3 VERTICAL REDUNDANCY CHECKING

The SEC code described in the previous section is used for detecting and correcting single bit errors in individual data words. Redundant bits are added to each data word, and each resulting codeword is treated independently. The recoding scheme is sometimes referred to as **horizontal** or **longitudinal redundancy checking (LRC)** because the width of the codeword is extended for the redundant bits.

An alternative approach is to use a **vertical redundancy checking (VRC)** code, in which a **checksum** word is added at the end of a group of words that are transmitted. In this case, parity is computed on a column by column basis, forming a checksum word that is appended to the message. The checksum word is computed and transmitted by the sender, and is recomputed and compared to the transmitted checksum word by the receiver. If an error is detected, then the receiver must request a retransmission since there is not enough redundancy to identify the position of an error. The VRC and LRC codes can be combined to improve error checking, as shown for the ASCII characters 'A' through 'H' in Figure 8-39.

In some situations, errors are bursty, and may corrupt several contiguous bits

<i>P</i>	<i>Code</i>	<i>Character</i>
0	1 0 0 0 0 0 1	A
0	1 0 0 0 0 1 0	B
1	1 0 0 0 0 1 1	C
0	1 0 0 0 1 0 0	D
1	1 0 0 0 1 0 1	E
1	1 0 0 0 1 1 0	F
0	1 0 0 0 1 1 1	G
0	1 0 0 1 0 0 0	H
1	0 0 0 1 0 0 0	Checksum

Figure 8-39 Combined LRC and VRC checking. Checksum bits form even parity for each column.

both horizontally and vertically. A more powerful scheme such as **cyclic redundancy checking** (CRC) is more appropriate for this situation, which is a variation of VRC checking in which the bits are grouped in a special way, as described in the next section.

8.5.4 CYCLIC REDUNDANCY CHECKING

Cyclic redundancy checking (CRC) is a more powerful error detection and correction scheme that operates in the presence of **burst errors**, which each begin and end with a bit error, with zero or more intervening corrupted bits. The two endpoint corrupted bits are included in the burst error. If the length of a burst error is B , then there must be B or more uncorrupted bits between burst errors.

CRCs use **polynomial codes**, in which a frame to be transmitted is divided by a polynomial, and the remainder is appended to the frame as a **frame check sequence** (FCS), commonly known as the **CRC digits**. The frame is transmitted (or stored) along with the CRC digits. After receiving the frame, the receiver then goes through the same computation, using the same polynomial, and if the remainders agree then there are no detectable errors. There can be undetectable errors, and the goal in creating a CRC code is to select a polynomial that covers the statistically likely errors for a given fault model.

The basic approach starts with a k -bit message to be transmitted, $M(x)$, which is appended with n 0's in which n is the degree of the **generator polynomial**, $G(x)$, with $k > n$. This extended form of $M(x)$ is divided by $G(x)$ using modulo 2 arithmetic (in which carries and borrows are discarded), and then the remainder, $R(x)$, which is no more than n bits wide, forms the CRC digits for $M(x)$.

As an example, consider a frame to be transmitted:

$$M(x) = 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1$$

and a generator polynomial $G(x) = x^4 + x + 1$. The **degree** of $G(x)$ (the highest exponent) is 4, and so we append 4 zeros to $M(x)$ to form the dividend of the computation.

The divisor is 10011, which corresponds to the coefficients in $G(x)$ written as:

$$G(x) = 1 \times x^4 + 0 \times x^3 + 0 \times x^2 + 1 \times x^1 + 1 \times x^0.$$

Notice that $G(x)$ has a degree of $n = 4$, and that there are $n + 1 = 5$ coefficients. The CRC digits are then computed as shown in Figure 8-40. The divisor

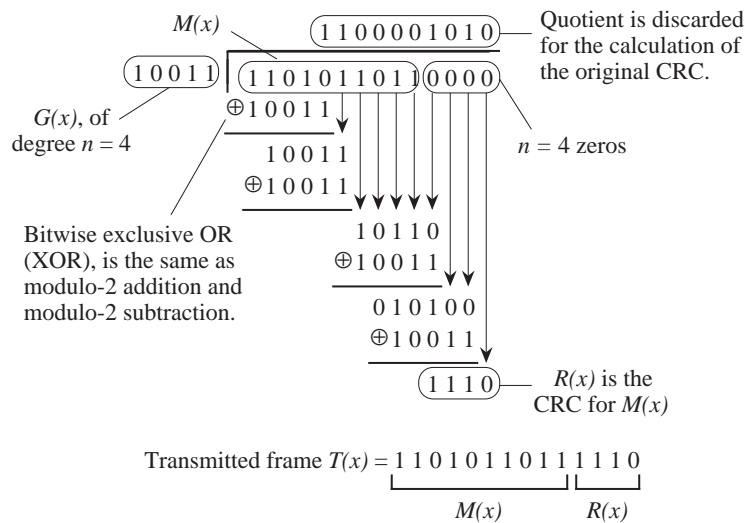


Figure 8-40 Calculation of the CRC digits.

(10011) is divided into the dividend, but the magnitudes of the divisor and dividend do not play a role in determining whether the divisor “goes into” the dividend at the location of a particular digit. All that matters is that the number of bits in the divisor (which has no leading zeros) matches the same number of bits in the dividend (which also must not have leading zeros at the position being checked.) Note that there are no borrows in modulo-2 subtraction, and that a bit-by-bit exclusive-OR (XOR) operation between the divisor and the dividend achieves the same result.

Now suppose that the transmitted frame $T(x) = M(x) + R(x)$ gets corrupted during transmission. The receiver needs to detect that this has happened. The receiver divides the received frame by $G(x)$, and all burst errors that do not include $G(x)$ as a factor will be caught because there will be a nonzero remainder for these cases. That is, as long as the 1's in 10011 do not coincide with the positions of errors in the received frame, all errors will be caught. In general, a polynomial code of degree n will catch all burst errors of length $\leq n$.

Common polynomials that give good error coverage include:

$$\text{CRC-16} = x^{16} + x^{15} + x^2 + 1$$

$$\text{CRC-CCITT} = x^{16} + x^{12} + x^5 + 1$$

$$\text{CRC-32} = x^{32} + x^{26} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

A deeper analysis of CRC codes is beyond the scope of this book, and the reader is referred to (Hamming, 1986) for further details.

EXAMPLE: ERROR CORRECTION

Consider how many check bits are needed for a double-error correcting ASCII code. There are $k = 7$ bits for each ASCII character, and we need to add r check bits to each codeword. For each of the 2^k ASCII characters there are $k + r$ possible one-bit error patterns, there are $\frac{(k+r)(k+r-1)}{2}$ possible two-bit error patterns, and there is one bit pattern for the uncorrupted codeword. There are 2^{k+r} possible bit patterns, and so the following relation must hold:

$$2^k \times \left[(k+r) + \frac{(k+r)(k+r-1)}{2} + 1 \right] \leq 2^{k+r}$$

\uparrow \nwarrow \uparrow \uparrow \swarrow
 Number of Number of Number of Uncorrupted Number of
 original one-bit two-bit codeword possible bit
 codewords errors errors patterns

Simplifying, using $k = 7$, yields: $r^2 + 15r + 58 \leq 2r + 1$ for which $r = 7$ is the smallest value that satisfies the relation.

Since a Hamming distance of $2p + 1$ must be maintained to correct p errors, the Hamming distance for this DEC code must be at least $2 \times 2 + 1 = 5$. If we use the same encoding for error detection instead, then we have $p + 1 = 5$, and since a Hamming distance of $p + 1$ must be maintained to detect p errors, then $p = 4$ errors can be detected. ■

EXAMPLE: TRANSFER TIME FOR A HARD DISK

Consider calculating the transfer time of a hard magnetic disk. For this example, assume that a disk rotates once every 16 ms. The seek time to move the head between adjacent tracks is 2 ms. There are 32 sectors per track that are stored in linear order (non-interleaved), from sector 0 to sector 31. The head sees the sectors in that order.

Assume the read/write head is positioned at the start of sector 1 on track 12. There is a memory buffer that is large enough to hold an entire track. Data is transferred between disk locations by reading the source data into memory, positioning the read/write head over the destination location, and writing the data to the destination.

How long will it take to transfer sector 1 on track 12 to sector 1 on track 13? How long will it take to transfer all of the sectors of track 12 to the corresponding sectors on track 13? Note that sectors do not have to be written in the same order they are read.

Solution:

The time to transfer a sector from one track to the next can be decomposed into its parts: the sector read time, the head movement time, the rotational delay, and the sector write time.

The time to read or write a sector is simply the time it takes for the sector to pass under the head, which is $(16 \text{ ms/track}) \times (1/32 \text{ tracks/sector}) = .5 \text{ ms/sector}$. For this example, the head movement time is only 2 ms because the head moves between adjacent tracks. After reading sector 1 on track 12, which takes .5 ms, an additional 15.5 ms of rotational delay is needed for the head to line up with sector 1 again. The head movement time of 2 ms overlaps the 15.5 ms of rotational delay, and so only the greater of the two times (15.5 ms) is used.

We sum the individual times and obtain: $.5 \text{ ms} + 15.5 \text{ ms} + .5 \text{ ms} = 16.5 \text{ ms}$ to transfer sector 1 on track 12 to sector 1 on track 13.

The time to transfer all of track 12 to track 13 is computed in a similar manner. The memory buffer can hold an entire track, and so the time to read or write an entire track is simply the rotational delay for a track, which is 16 ms. The head movement time is 2 ms, which is also the time for four sectors to pass under the head. Thus, after reading a track and moving the head, the head is now on track 13, at four sectors past the initial sector that was read on track 12.

Sectors can be written in a different order than they are read. Track 13 can thus be written with a four sector offset with respect to how track 12 was read. The time to write track 13 is 16 ms, and the time for the entire transfer then is: $16 \text{ ms} + 2 \text{ ms} + 16 \text{ ms} = 34 \text{ ms}$. Notice that the rotational delay is zero for this example because the head lands at the beginning of the first sector to be written. ■

8.6 Case Study: Communication on the Intel Pentium Architecture

The Intel Pentium processor family is Intel's current state-of-the-art implementation of their venerable x86 family, which began with the Intel 8086, released in 1978. The Pentium is itself a processor family, with versions that emphasize high speed, multiprocessor environments, graphics, low power, *etc.* In this section we examine the common features that underlie the Pentium system bus.

System clock, bus clock, and bus speeds

Interestingly, the system clock speed is set as a multiple of the bus clock. The value of the multiple is set by the processor whenever it is reset, according to the values on several of its pins. The possible values of the multiple vary across family members. For example, the Pentium Pro, a family member adapted for multiple CPU applications, can have multipliers ranging from 2 to 3-1/2. We mention again here that the reason for clocking the system bus at a slower rate than the CPU is that CPU operations can take place faster than memory access operations. A common bus clock frequency in Pentium systems is 66 MHz.

Address, data, memory, and I/O capabilities

The system bus effectively has 32 address lines, and can thus address up to 4 GB of main memory. Its data bus is 64 bits wide; thus the processor is capable of transferring an 8-byte quadword in one bus cycle. (Intel x86 words are 16-bits

long.) We say “effectively” because in fact the Pentium processor decodes the least significant three address lines, A_2 - A_0 , into eight “byte enable” lines, $BE0\#$ - $BE7\#$, prior to placing them on the system bus.¹ The values on these eight lines specify the byte, word, double word, or quad word that is to be transferred from the base address specified by A_{31} - A_3 .

Data words have soft-alignment

Data values have so-called **soft alignment**, meaning that words, double words, and quad words should be aligned on even word, double word, and quad word boundaries for maximum efficiency, but the processor can tolerate misaligned data items. The penalty for accessing misaligned words may be two bus cycles, which are required to access both halves of the datum.²

As a bow to the small address spaces of early family members, all Intel processors have separate address spaces for memory and I/O accesses. The address space to be selected is specified by the $M/I/O\#$ bus line. A high value on this line selects the 2 GB memory address space, and low specifies the I/O address space. Separate opcodes, IN and OUT, are used to access this space. It is the responsibility of all devices on the bus to sample the $M/I/O\#$ line at the beginning of each bus cycle to determine the address space to which the bus cycle is referring—memory or I/O. Figure 8-41 shows these address spaces graphically. I/O addresses in the x86 family are limited to 16 bits, allowing up to 64K I/O locations. Figure 8-41 shows the two address spaces.

Bus cycles in the Pentium family

The Pentium processor has a total of 18 different bus cycles, to serve different needs. These include the standard memory read and write bus cycles, the bus hold cycle, used to allow other devices to become the bus master, an interrupt acknowledge cycle, various “burst” cache access cycles, and a number of other special purpose bus cycles. In this Case Study we examine the read and write bus cycles, the “burst read” cycle, in which a burst of data can be transferred, and the bus hold/hold acknowledge cycle, which is used by devices that wish to become

1. The “#” symbol is Intel’s notation for a bus line that is active low.

2. Many systems require so-called hard alignment. Misaligned words are not allowed, and their detection causes a processor exception to be raised.

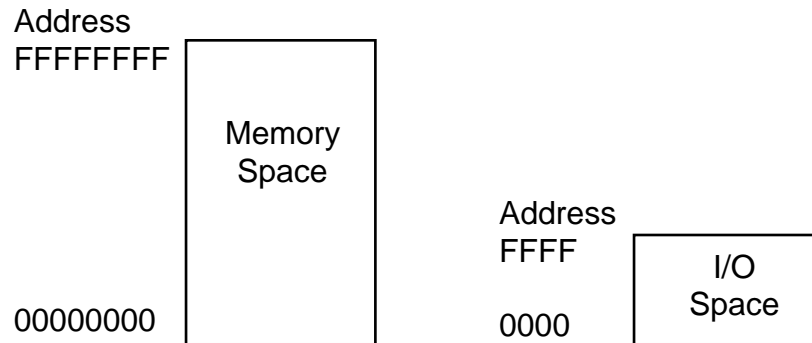


Figure 8-41 Intel memory and I/O address spaces.

the bus master.

Memory read and write bus cycles

The “standard” read and write cycles are shown in Figure 8-42. By convention,

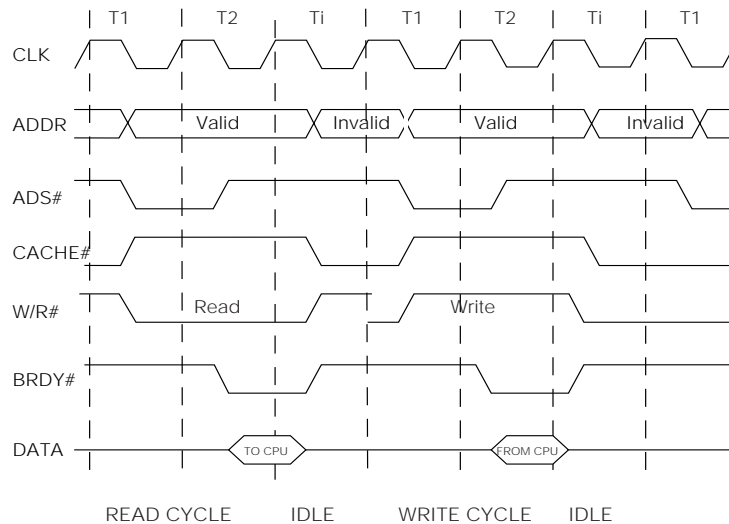


Figure 8-42 The standard Intel Pentium Read and Write bus cycles.

the states of the Intel bus are referred to as “T states,” where each T state is one clock cycle. There are three T states shown in the figure, T1, T2, and Ti, where Ti is the “idle” state, the state that occurs when the bus is not engaged in any specific activity, and when no requests to use the bus are pending. Recall that a “#” following a signal name indicates that a signal is active low, in keeping with Intel conventions.

Both read and write cycles require a minimum of two bus clocks, T1 and T2:

- The CPU signals the start of all new bus cycles by asserting the Address Status signal, ADS#. This signal both defines the start of a new bus cycle and signals to memory that a valid address is available on the address bus, ADDR. Note the transition of ADDR from invalid to valid as ADS# is asserted.
- The de-assertion of the cache load signal, CACHE#, indicates that the cycle will be composed of a single read or write, as opposed to a burst read or write, covered later in this section.
- During a read cycle the CPU asserts read, W/R#, simultaneously with the assertion of ADS#. This signals the memory module that it should latch the address and read a value at that address.
- Upon a read, the memory module asserts the Burst Ready, BRDY#, signal as it places the data, DATA, on the bus, indicating that there is valid data on the data pins. The CPU uses BRDY# as a signal to latch the data values.
- Since CACHE# is deasserted, the assertion of a single BRDY# signifies the end of the bus cycle.
- In the write cycle, the memory module asserts BRDY# when it is ready to accept the data placed on the bus by the CPU. Thus BRDY# acts as a handshake between memory and the CPU.
- If memory is too slow to accept or drive data within the limits of two clock cycles, it can insert “wait” states by not asserting BRDY# until it is ready to respond.

The burst Read bus cycle

Because of the critical need to supply the CPU with instructions and data from memory that is inherently slower than the CPU, Intel designed the burst read and write cycles. These cycles read and write four eight-byte quad words in a burst, from consecutive addresses. Figure 8-43 shows the Pentium burst read cycle.

The burst read cycle is initiated by the processor placing an address on the address lines and asserting ADS# as before, but now, by asserting the CACHE# line the processor signals the beginning of a burst read cycle. In response the memory asserts BRDY# and places a sequence of four 8-byte quad words on the data bus, one quad word per clock, keeping BRDY# asserted until the entire

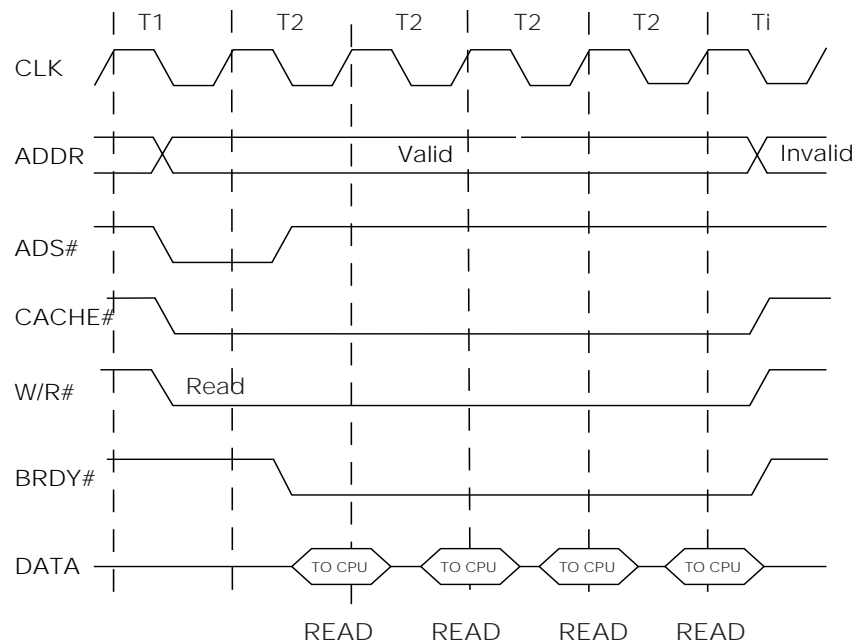


Figure 8-43 The Intel Pentium burst read bus cycle.

transfer is complete.

There is an analogous cycle for burst writes. There is also a mechanism for coping with slower memory by slowing the burst transfer rate from one per clock to one per two clocks.

Bus hold for request by bus master

There are two bus signals for use by devices requesting to become bus master: hold (HOLD) and hold acknowledge (HLDA). Figure 8-44 shows how the transactions work. The figure assumes that the processor is in the midst of a read cycle when the HOLD request signal arrives. The processor completes the current (read) cycle, and inserts two idle cycles, Ti. During the falling edge of the second Ti cycle the processor floats all of its lines and asserts HLDA. It keeps HLDA asserted for two clocks. At the end of the second clock cycle the device asserting HLDA “owns” the bus, and it may begin a new bus operation at the following cycle, as shown at the far right end of the figure. In systems of any complexity there will be a separate bus controller chip to mediate among the several devices that may wish to become the bus master.

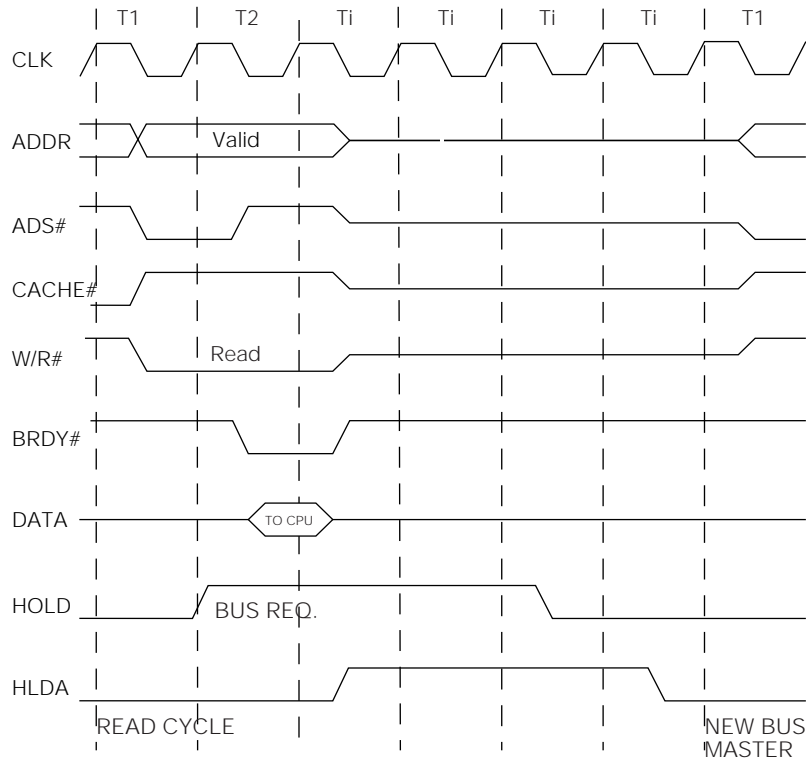


Figure 8-44 The Intel Pentium Hold-Hold Acknowledge bus cycle.

Data transfer rates

Let us compute the data transfer rates for the read and burst read bus cycles. In the first case, 8 bytes are transferred in two clock cycles. If the bus clock speed is 66 MHz, this is a maximum transfer rate of

$$\frac{8}{2} \times 66 \times 10^6$$

or 264 million bytes per second. In burst mode that rate increases to four 8-byte bursts in five clock cycles, for a transfer rate of

$$\frac{32}{5} \times 66 \times 10^6$$

or 422 million bytes per second. (Intel literature uses 4 cycles rather than 5 as the denominator, thus arriving at a burst rate of 528 million bytes per second. Take

your pick.)

At the 422 million byte rate, with a bus clock multiplier of 3-1/2, the data transfer rate to the CPU is

$$\frac{422 \times 10^6}{3.5 \times 66 \times 10^6}$$

or about 2 bytes per clock cycle. Thus under optimum, or ideal conditions, the CPU is probably just barely kept supplied with bytes. In the event of a branch instruction or other interruption in memory activity, the CPU will become starved for instructions and data.

The Intel Pentium is typical of modern processors. It has a number of specialized bus cycles that support multiprocessors, cache memory transfers, and other kinds of special situations. Refer to the Intel literature (see below) for more details.

■ SUMMARY

Mass storage devices come in a variety of forms. Examples of mass storage devices are hard disks and magnetic tape units. Optical storage provides greater density per unit area than magnetic storage, but is more expensive and does not offer the same degree of user writability. An example of an optical storage device is a CD ROM.

There is a wide range of other input/output devices. The few that we studied in this chapter that are not mass storage devices can be grouped into input devices and output devices. Examples of input devices are keyboards, bit pads, mice, trackballs, lightpens, touch screens, and joysticks. Examples of output devices are laser printers and video displays.

Input, output, and communication involve the transfer of information between transmitters and receivers. The transmitters, receivers, and methods of communication are often mismatched in terms of speed and in how information is represented, and so an important consideration is how to match input and output devices with a system using a particular method of communication.

A bus provides a fixed bandwidth that is shared among a number of devices. When a bus is used within a computer, communication is handled via programmed I/O, interrupt driven I/O, or DMA. In complex systems, a higher level

organization is used, which is known as an I/O channel.

One method of interconnecting systems is through the use of modems, and another method is through the use of LANs. A LAN operates over a limited geographical distance, and is generally self-contained. LANs provide greater bandwidth per channel than modems, but require a substantially greater investment in hardware and maintenance.

With the proliferation of personal communication devices and the expansion of telecommunications providers into LAN markets, the distinctions between modem based communication networks and LANs grow more obscure. The various mediums of communication among systems are frequently referred to collectively as "communication networks," without making distinctions among the specific forms of communication.

Error detection and correction are possible through redundancy, in which there are more bit patterns possible than the number of valid bit patterns. If the error bit patterns do not have a single closest valid codeword, then error detection is possible but error correction is not possible. If every error bit pattern is reachable from only one valid bit pattern, then error correction is also possible.

■ FURTHER READING

(Hamacher *et al.*, 1990) provides explanations of communication devices and a number of peripherals such as an alphanumeric CRT controller. (Tanenbaum, 1999) and (Stallings, 1996) also give readable explanations of peripheral devices. The material on synchronous and asynchronous busses, and bus arbitration, is influenced by a detailed presentation in (Tanenbaum, 1999). (Stallings, 1996) covers I/O channels.

(Needleman, 1990) and (Schnaidt, 1990) give a thorough treatment of local area networks according to the OSI model, and (Tanenbaum, 1996) is a good reference on communications in general.

(Tanenbaum, 1999) and (Stallings, 1993) give readable explanations of Hamming encoding. (Hamming, 1986) and (Peterson and Weldon, 1972) give more detailed treatments of error-correcting codes.

Intel data sheets and other literature, including the Pentium, Pentium II, and Pentium Pro hardware and programmer's manuals can be ordered from Intel Literature Sales, PO Box 7641, Mt. Prospect IL 60056-7641, or, in the U. S. and Canada, by calling (800) 548-4725.

Hamacher, V. C., Z. G. Vranesic, and S. G. Zaky, *Computer Organization*, 3/e, McGraw Hill, (1990).

Hamming, R. W., *Coding and Information Theory*, 2/e, Prentice-Hall, (1986).

Needleman, R., *Understanding Networks*, Simon and Schuster, New York, (1990).

Peterson, W. Wesley and E. J. Weldon, Jr., *Error-Correcting Codes*, 2/e, The MIT Press, (1972).

Schnaidt, P., *LAN Tutorial*, Miller Freeman Publications, California, (1990).

Stallings, W., *Data and Computer Communications*, 2/e, MacMillan Publishing, New York, (1988).

Stallings, W., *Computer Organization and Architecture: Designing for Performance*, 4/e, Prentice Hall, Upper Saddle River, (1996).

Tanenbaum, A., *Computer Networks*, 3/e, Prentice Hall, Upper Saddle River, (1996).

Tanenbaum, A., *Structured Computer Organization*, 4/e, Prentice Hall, Englewood Cliffs, (1999).

■ PROBLEMS

- 8.1 Show the Manchester encoding of the bit sequence: 10011101.
- 8.2 A disk that has 16 sectors per track uses an interleave factor of 1:4. What is the smallest number of revolutions of the disk required to read all of the sectors of a track in sequence.
- 8.3 A hard magnetic disk has two surfaces. The storage area on each surface

has an inner radius of 1 cm and an outer radius of 5 cm. Each track holds the same number of bits, even though each track differs in size from every other. The maximum storage density of the media is 10,000 bits/cm. The spacing between corresponding points on adjacent tracks is .1 mm, which includes the inter-track gap. Assume that the inter-sector gaps are negligible, and that a track exists on each edge of the storage area.

- (a) What is the maximum number of bits that can be stored on the disk?
- (b) What is the data transfer rate from the disk to the head in bits per second at a rotational speed of 3600 RPM?

8.4 A disk has 128 tracks of 32 sectors each, on each surface of eight platters. The disk spins at 3600 rpm, and takes 15 ms to move between adjacent tracks. What is the longest time needed to read an arbitrary sector located anywhere on the disk?

8.5 A 300 Mbyte (300×2^{20} bytes) disk has 815 cylinders, with 19 heads, a track-to-track speed of 7.5 m/s (that is, 7.5 meters per second), and a rotation rate of 3600 RPM. The fact that there are 19 heads means that there are 10 platters, and only 19 surfaces are used for storing data. Each sector holds the same amount of data, and each track has the same number of sectors. The transfer time between the disk and the CPU is 300 Kbytes/sec. The track-to-track spacing is .25 mm.

- (a) Compute the time to read a track (*not* the time to transmit the track to a host). Assume that interleaving is not used.
- (b) What is the minimum time required to read the entire disk pack to a CPU, given the best of all possible circumstances? Assume that the head of the first surface to be read is positioned at the beginning of the first sector of the first track, and that an entire cylinder is read before the arm is moved. Also assume that the disk unit can buffer an entire cylinder, but no more. During operation, the disk unit first fills its buffer, then it empties it to the CPU, and only then does it read more of the disk.

8.6 A fixed head disk has one head per track. The heads do not move, and thus, there is no head movement component in calculating the access time.

For this problem, calculate the time that it takes to copy one surface to another surface. This is an internal operation to the disk, and does not involve any communication with the host. There are 1000 cylinders, each track holds 10 sectors, and the rotation rate of the disk is 3000 RPM. The sectors all line up with each other. That is, within a cylinder, sector 0 on each track lines up with sector 0 on every other track, and within a surface, sector 0 in each track begins on the same line drawn from the center of the surface to the edge.

An internal buffer holds a single sector. When a sector is read from one track, it is held in the buffer until it is written onto another track. Only then can another sector be read. It is not possible to simultaneously read and write the buffer, and the buffer must be entirely loaded or entirely emptied – partial reads or writes are not allowed. Calculate the minimum time required to copy one surface to another, given the best starting conditions. The surfaces must be direct images of each other. That is, sector i in the source surface must be directly above or below sector i in the destination surface.

8.7 Compute the storage capacity of a 6250 byte per inch (BPI) tape that is 600 ft long and has a record size of 2048 bytes. The size of an inter-record gap is .5 in.

8.8 A bit mapped display is 1024 pixels wide by 1024 pixels high. The refresh rate is 60 Hz, which means that every pixel is rewritten to the screen 60 times a second, but only one pixel is written at any time. What is the maximum time allowed to write a single pixel?

8.9 How many bits need to be stored in the LUT in Figure 8-17? If the LUT is removed, and the RAM is changed to provide the 24-bit R, G, and B output directly, how many additional bits need to be stored in the RAM? Assume that the initial size of the RAM is $2^{10} \times 2^9 = 2^{19}$ words \times 8 bits/word.

8.10 The MCB as presented in Section 8.2.1 keeps track of every sector on the disk. An alternative organization, which significantly reduces the size of the MCB, is to store blocks in **chains**. The idea is to store only the first block of a file in the MCB, and then store a pointer to the succeeding block at the end of the first block. Each succeeding block is linked in a similar manner.

(a) How does this approach affect the time to access the middle of a file?

(b) After a system crash, would a disk recovery be easier if only the first sector of a file is stored in the MCB, and the remaining list of sectors is stored in a header at the beginning of each file? How does this approach affect storage?

8.11 You are now the administrator for a computer system that is maintained by Mega Equipment Corporation (MEC). As part of routine maintenance, MEC realigns the heads on one of the disks, and now the disk cannot be read or written without producing errors. What went wrong? Would this happen with or without the use of a timing track?

8.12 Why must the CPU ensure that interrupts are disabled before handing control over to the ISR?

8.13 What is the Hamming distance for the ASCII SEC code discussed in Section 8.5.2?

8.14 Construct the SEC code for the ASCII character 'Q' using even parity.

8.15 For parts (a) through (d) below, use a SEC code with even parity.

a) How many check bits should be added to encode a six-bit word?

b) Construct the SEC code for the six-bit word: 1 0 1 1 0 0. When constructing the code, number the bits from right to left starting with 1 as for the method described in Section 8.5.2.

c) A receiver sees a two-bit SEC encoded word that looks like this: 1 1 1 0 0. What is the initial two-bit pattern?

d) The 12-bit word: 1 0 1 1 1 0 0 1 1 0 0 1 complete with an SEC code (even parity) is received. What 12-bit word was actually sent?

8.16 How many check bits are needed for a SEC code for an initial word size of 1024?

8.17 Construct a checksum word for EBCDIC characters 'V' through 'Z' using vertical redundancy checking with even parity. DO NOT use longitudinal redundancy checking. Show your work.

- 8.18** Compare the number of bits used for parity in the SEC code with the simple parity VRC code, for 1024 eight-bit characters:
- Compute the number of check bits generated using SEC only (horizontally).
 - Compute the number of checksum bits using VRC only.
- 8.19** The SEC code discussed in Section 8.5.2 can be turned into a double error detecting/SEC (DED/SEC) code by adding one more bit that creates even parity over the SEC code (which includes the parity bit being added.) Explain how double error detection works while also maintaining single error correction with this approach.
- 8.20** Compute the CRC for a message to be transmitted $M(x) = 101100110$ and a generator polynomial $G(x) = x^3 + x^2 + 1$.
- 8.21** What is the longest burst error that CRC-32 is sure to catch?