“Who is General Failure and why is he reading my disk?”

—Anonymous

CHAPTER 7

Input/Output and Storage Systems

7.1 INTRODUCTION

One could easily argue that computers are more useful to us as appliances for information storage and retrieval than they are as instruments of computation. Indeed, without having some means of getting data into the computer and information out of it, we have little use at all for a CPU and memory. We interact with these components only through the I/O devices connected to them.

With personal systems, for example, a keyboard and mouse are the primary user input devices. A standard monitor is an output-only device that presents results to the user. Although most printers provide device status information to the host system to which they are connected, they are still considered output-only devices. Disk drives are called input/output devices because data can be written to and read from them. I/O devices also exchange control and status information with the host system. We observe that the term **I/O** is commonly used both as an adjective and as a noun: Computers have I/O devices connected to them, and to achieve good performance, one endeavors to keep disk I/O to a minimum.

After reading this chapter, you will understand the details of how input, output, and I/O devices interact with their host systems, and the various ways in which I/O is controlled. We also discuss the internals of mass storage devices and some of the ways in which they are put to work in large-scale computer systems. Enterprise-class storage systems incorporate many of the ideas presented in this chapter, but they also rely on data network infrastructures. We therefore defer our discussion of storage systems until Chapter 13.

7.2 I/O AND PERFORMANCE

We expect our computer systems to be able to efficiently store and retrieve data, and to quickly carry out the commands we give them. When processing time exceeds user "think time," we complain that the computer is "slow." Sometimes this slowness can have a substantial productivity impact, measured in hard currency. More often than not, the root cause of the problem is not in the processor or the memory but in how the system processes its input and output (I/O).

I/O is more than just file storage and retrieval. A poorly functioning I/O system can have a ripple effect, dragging down the entire computer system. In the preceding chapter, we described virtual memory, that is, how systems page blocks of memory to disk to make room for more user processes in main memory. If the disk system is sluggish, process execution slows down, causing backlogs in CPU and disk queues. The easy solution to the problem is to simply throw more resources at the system. Buy more main storage. Buy a faster processor. If we're in a particularly Draconian frame of mind, we could simply limit the number of concurrent processes!

Such measures are wasteful, if not plain irresponsible. If we really understand what's happening in a computer system, we can make the best use of the resources available, adding costly resources only when absolutely necessary. There are a number of tools we can use to determine the most effective way performance can be improved. Amdahl's Law is one of them.

7.3 AMDAHL' S LAW

Each time a (particular) microprocessor company announces its latest and greatest CPU, headlines sprout across the globe heralding this latest leap forward in technology. Cyberphiles the world over would agree that such advances are laudable and deserving of fanfare. However, when similar advances are made in I/O technology, the story is apt to appear on page 29 of some obscure trade magazine. Under the glare of media hype, it is easy to lose sight of the integrated nature of computer systems. A 40% speedup for one component certainly will not make the entire system 40% faster, despite media implications to the contrary.

In 1967, Gene Amdahl recognized the interrelationship of all components with the overall efficiency of a computer system. He quantified his observations in a formula, which is now known as Amdahl's Law. In essence, Amdahl's Law states that the overall **speedup** of a computer system depends on both the speedup in a particular component and how much that component is used by the system. In symbols:

$$S = \frac{1}{(1 - f) + f/k}$$

where

S is the overall system speedup;
f is the fraction of work performed by the faster component; and
k is the speedup of a new component.

Let's say that most of your daytime processes spend 70% of their time running in the CPU and 30% waiting for service from the disk. Suppose also that someone is trying to sell you a processor array upgrade that is 50% faster than what you have and costs $10,000. The day before, someone called offering you a set of disk drives for $7,000. These new disks promise to be 150% faster than your existing disks. You know that the system performance is starting to degrade, so you need to do something. Which would you choose to yield the best performance improvement for the least amount of money?

For the processor option, we have:

$$f = .70,\ k = 1.5,\ \text{so } S = \frac{1}{(1 - 0.7) + 0.7/5} = 1.30$$

We therefore appreciate a total speedup of 1.3 times, or 30%, with the new processor for $10,000.

For the disk option, we have:

$$f = .30,\ k = 2.5,\ \text{so } S = \frac{1}{(1 - 0.3) + 0.3/2.5} \approx 1.22$$

The disk upgrade gives us a speedup of 1.22 times, or 22%, for $7,000.

All things being equal, it is a close decision. Each 1% of performance improvement resulting from the processor upgrade costs about $333. Each 1% with the disk upgrade costs about $318. This makes the disk upgrade a slightly better choice, based solely on dollars spent per performance improvement percentage point. Certainly, other factors would influence your decision. For example, if your disks are nearing the end of their expected lives, or if you're running out of disk space, you might consider the disk upgrade even if it were to cost more than the processor upgrade.

What Do We Really Mean by "Speedup"?

Amdahl's Law uses the variable *K* to represent the speedup of a particular component. But what do we really mean by "speedup"?

There are many different ways to discuss the notion of "speedup." For example, one person may say A is twice as fast as B; another may say A is 100% faster than B. It is important to understand the difference if you want to use Amdahl's Law.

It is a common misconception to believe that if A is twice as fast as B that A is 200% faster than B. However, this is not accurate. It is easy to see that if A is twice as fast as B, that a multiple of 2 is involved. For example, if Bob runs a race in 15 seconds, but it takes Sue 30 seconds, clearly Bob is twice as fast as Sue. If Bob is running an average of 4mph, then Sue must be running an average of 2mph. The error occurs when converting it to "percentage" terminology. Bob's speed does not represent a 200% increase over Sue's speed; it is only a 100% increase. This becomes clearer when we look at the definition for "% faster."

$$\text{A is N\% faster than B if } \frac{\textit{time B}}{\textit{time A}} = 1 + \frac{N}{100}$$

Bob is 100% faster than Sue because 30/15 = 1 + 100/100 (*N* must be 100). The ratio of Bob's time to Sue's time (30/15) represents the speedup (Bob is 2 times faster than Sue). It is this notion of speedup that must be used for *k* in Amdahl's' equation; it is also this notion of speedup that results from applying Amdahl's Law.

Suppose we wish to use Amdahl's Law to find the overall speedup for a system, assuming we replace the CPU. We know that the CPU is used 80% of the time and that the new CPU we are considering is 50% faster than the current one in the system. Amdahl's Law requires us to know the speedup of the newer component. The variable *k* in this case is not 50 or 0.5; instead, it is 1.5 (because the newer CPU is 1.5 times faster than the old one):

$$\frac{\textit{time old CPU}}{\textit{time new CPU}} = 1 + \frac{50}{100} = 1.5$$

So, applying Amdahl's Law, we get:

$$S = \frac{1}{(1 - 0.8) + 8/1.5} = 1.36$$

which means we have an overall speedup of 1.36; with the new CPU, the system will be 1.36 times faster. The new system is 36% faster.

In addition to being used for hardware speedup, Amdahl's Law can be used in programming. It is well known that, on average, a program spends a majority of its time in a small percentage of its code. Programmers will often focus on increasing the performance of that small segment of code. They can use Amdahl's Law to determine the overall effect on the program's running time.

Suppose you have written a program and determined that 80% of the time is spent in one segment of code. You examine the code and determine that you can decrease the running time in that segment of code by half (i.e., a speedup of 2). If we apply Amdahl's Law, we see that the overall effect on the entire program is:

$$S = \frac{1}{(1 - .8) + .8/2} = 1.67$$

which means the program, as a whole, will run 1.67 times faster with the new code.

Consider one last example. As a programmer, you have an option of making a segment of code that is used 10% of the time 100 times faster. You estimate the cost at one month (time and wages) to rewrite the code. You could also make it 1,000,000 times faster but estimate that it will cost you 6 months. What should you do? If we use Amdahl's Law, we see that a speedup of 100 times yields:

$$S = \frac{1}{(1 - .1) + .1/100} = 1.1098$$

so the overall program is sped up 1.1 times (roughly 11%). If we spend 6 months to increase the performance 1,000,000 times, the overall program execution speedup is:

$$S = \frac{1}{(1 - .1) + 1/1.000.000} = 1.1111$$

The speedup is minimal at best. So it is clear that spending the additional time and wages to speed up that section of code is not worth the effort. Programmers working on parallel programs often apply Amdahl's Law to determine the benefit of parallelizing code.

Amdahl's Law is used in many areas of computer hardware and software, including programming in general, memory hierarchy design, hardware replacement, processor set design, instruction set design, and parallel programming. However, Amdahl's Law is important to *any* activity subject to the notion of diminishing returns; even business managers are applying Amdahl's Law when developing and comparing various business processes.

Before you make that disk decision, however, you need to know your options. The sections that follow will help you to gain an understanding of general I/O architecture, with special emphasis on disk I/O. Disk I/O follows closely behind the CPU and memory in determining the overall effectiveness of a computer system.

7.4 I/O ARCHITECTURES

We will define input/output as a subsystem of components that moves coded data between external devices and a host system, consisting of a CPU and main memory. I/O subsystems include, but are not limited to:

- Blocks of main memory that are devoted to I/O functions
- Buses that provide the means of moving data into and out of the system
- Control modules in the host and in peripheral devices
- Interfaces to external components such as keyboards and disks
- Cabling or communications links between the host system and its peripherals

Figure 7.1 shows how all of these components can fit together to form an integrated I/O subsystem. The I/O modules take care of moving data between main memory and a particular device interface. Interfaces are designed specifically to communicate with certain types of devices, such as keyboards, disks, or printers. Interfaces handle the details of making sure that devices are ready for the next batch of data, or that the host is ready to receive the next batch of data coming in from the peripheral device.

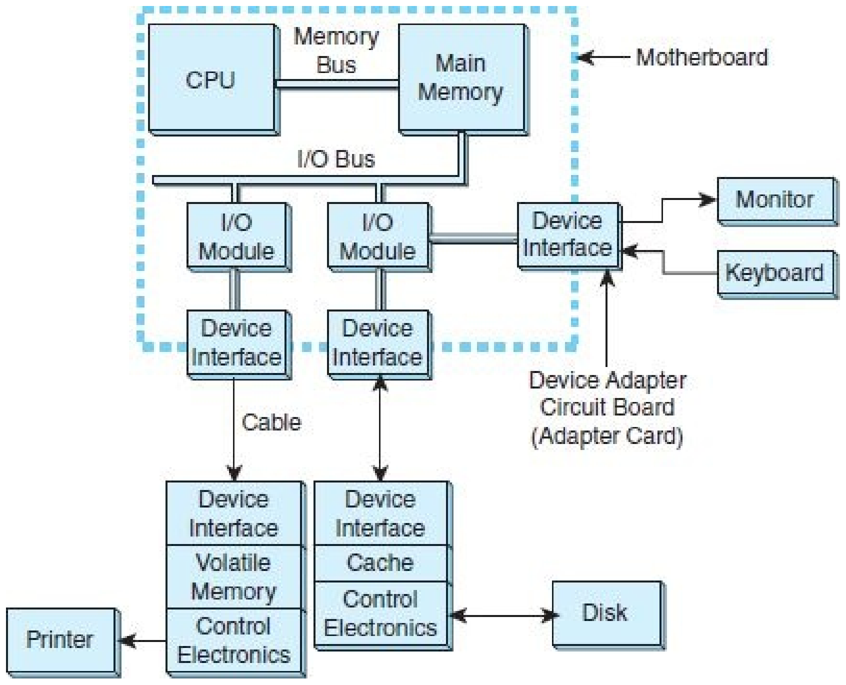


FIGURE 7.1 A Model I/O Configuration

The exact form and meaning of the signals exchanged between a sender and a receiver is called a **protocol**. Protocols include command signals, such as “Printer reset”; status signals, such as “Tape ready”; or data-passing signals, such as “Here are the bytes you requested.” In most data-exchanging protocols, the receiver must acknowledge the commands and data sent to it or indicate that it is ready to receive data. This type of protocol exchange is called a **handshake**.

External devices that handle large blocks of data (such as printers, and disk and tape drives) are often equipped with buffer memory. Buffers allow the host system to send large quantities of data to peripheral devices in the fastest manner possible, without having to wait until slow mechanical devices have actually written the data. Dedicated memory on disk drives is usually of the fast cache variety, whereas printers are usually provided with slower RAM.

Device control circuits take data to or from on-board buffers and ensure that it gets where it’s going. In the case of writing to disks, this involves making certain that the disk is positioned properly so that the data is written to a particular location. For printers, these circuits move the print head or laser beam to the next character position, fire the head, eject the paper, and so forth.

Disk and tape are forms of **durable storage**, so called because data recorded on them lasts longer than it would in volatile main memory. However, no storage method is permanent. The expected life of data on these media is approximately 5 to 30 years for magnetic media and as much as 100 years for optical media.

7.4.1 I/O Control Methods

Because of the great differences in control methods and transmission modes among various kinds of I/O devices, it is infeasible to try to connect them directly to the system bus. Instead, dedicated I/O modules serve as interfaces between the CPU and its peripherals. These modules perform many functions, including controlling device actions, buffering data, performing error detection, and communicating with the CPU. In this section, we are most interested in the method by which these I/O modules communicate with the CPU, thus controlling I/O. Computer systems employ any of five general I/O control methods, including **programmed I/O**, **interrupt-driven I/O**, **memory-mapped I/O**, **direct memory access**, and

channel-attached I/O. Although one method isn't necessarily better than another, the manner in which a computer controls its I/O greatly influences overall system design and performance. The objective is to know when the I/O method employed by a particular computer architecture is appropriate to how that system will be used.

Programmed I/O

The simplest way for a CPU to communicate with an I/O device is through **programmed I/O**, sometimes called **polled I/O** (or **port I/O**). The CPU continually monitors (**polls**) a control register associated with each I/O port. When a byte arrives in the port, a bit in the control register is also set. The CPU eventually polls the port and notices that the "data ready" control bit is set. The CPU resets the control bit, retrieves the byte, and processes it according to instructions programmed for that particular port. When the processing is complete, the CPU resumes polling the control registers as before.

The benefit of using this approach is that we have programmatic control over the behavior of each device. By modifying a few lines of code, we can adjust the number and types of devices in the system, as well as their polling priorities and intervals. Constant register polling, however, is a problem. The CPU is in a continual "busy wait" loop until it starts servicing an I/O request. It doesn't do any useful work until there is I/O to process. Another problem is in deciding how frequently to poll; some devices might need to be polled more frequently than others. Because of these limitations, programmed I/O is best suited for special-purpose systems such as automated teller machines and embedded systems that control or monitor environmental events.

Interrupt-Driven I/O

A more common and efficient control method is **interrupt-driven I/O**. Interrupt-driven I/O can be thought of as the converse of programmed I/O. Instead of the CPU continually asking its attached devices whether they have any input, the devices tell the CPU when they have data to send. The CPU proceeds with other tasks until a device requesting service sends an interrupt to the CPU. These interrupts are typically generated for every word of information that is transferred. In most interrupt-driven I/O implementations, this communication takes place through an intermediary interrupt controller. This circuit handles interrupt signals from all I/O devices in the system. Once this circuit recognizes an interrupt signal from any of its attached devices, it raises a single interrupt signal that activates a control line on the system bus. The control line typically feeds directly into a pin on the CPU chip. An example configuration is shown in Figure 7.2. Every peripheral device in the system has access to an interrupt request line. The interrupt control chip has an input for each interrupt line. Whenever an interrupt line is asserted, the controller decodes the interrupt and raises the Interrupt (INT) input on the CPU. When the CPU is ready to process the interrupt, it asserts the Interrupt Acknowledge (INTA) signal. Once the interrupt controller gets this acknowledgment, it can lower its INT signal. When two or more I/O interrupts occur simultaneously, the interrupt controller determines which one should take precedence, based on the time-criticality of the device requesting the I/O. Keyboard and mouse I/O are usually the least critical.

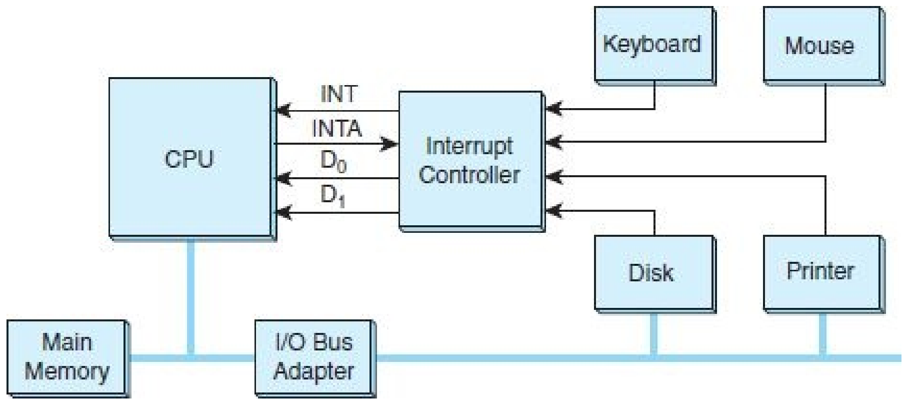


FIGURE 7.2 An I/O Subsystem Using Interrupts

System designers determine which devices should take precedence over the others when more than one device raises an interrupt simultaneously. The interrupt priorities are then hardwired into the I/O controller, making them practically impossible to change. Each computer that uses the same operating system and interrupt controller connects low-priority devices (such as a keyboard) to the same interrupt request line. The number of interrupt request lines is necessarily limited, and in some cases, the interrupt can be shared. Shared interrupts cause no problems when it is clear that no two devices will need the same interrupt at the same time. For example, a scanner and a printer can usually coexist peacefully using the same interrupt. This is not always the case with serial mice and modems, which often do try to share the same interrupt, causing bizarre behavior in both.

In Chapter 4, we described how interrupt processing changes the fetch–decode–execute cycle. We have reproduced Figure 4.12 in Figure 7.3, which shows how the CPU finishes execution of the current instruction and checks the status of its interrupt pins (not just I/O) at the beginning of every fetch–decode–execute cycle. Once the CPU acknowledges the interrupt, it saves its current state and processes the interrupt (as shown in Figure 7.4, also taken from Chapter 4).

Interrupt-driven I/O is similar to programmed I/O in that the service routines can be modified to accommodate hardware changes. Because vectors for the various types of hardware are usually kept in the same locations in systems running the same type and level of operating system, these vectors are easily changed to point to vendor-specific code. For example, if someone comes up with a new type of disk drive that is not yet supported by a popular operating system, the disk's manufacturer may provide a specialized **device driver** program to be kept in memory along with code for the standard devices. Installation of the device driver code involves updating the disk I/O vector to point to code particular to the disk drive.

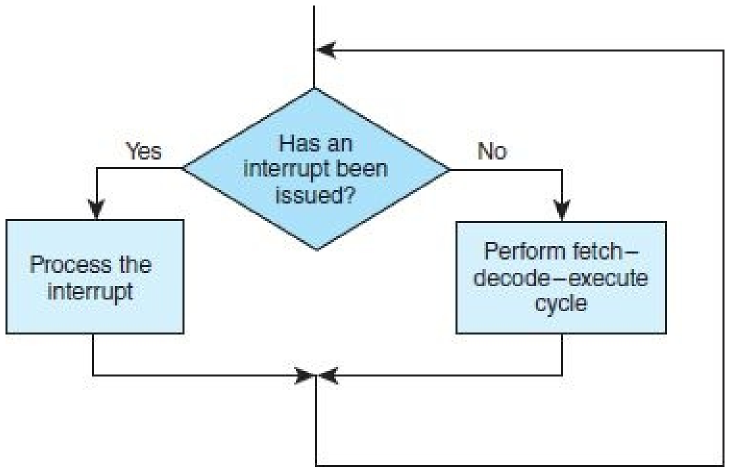

Has an interrupt been issued?
Yes
No
Process the interrupt
Perform fetch–decode–execute cycle

FIGURE 7.3 The Fetch–Decode–Interrupt Cycle with Interrupt Checking

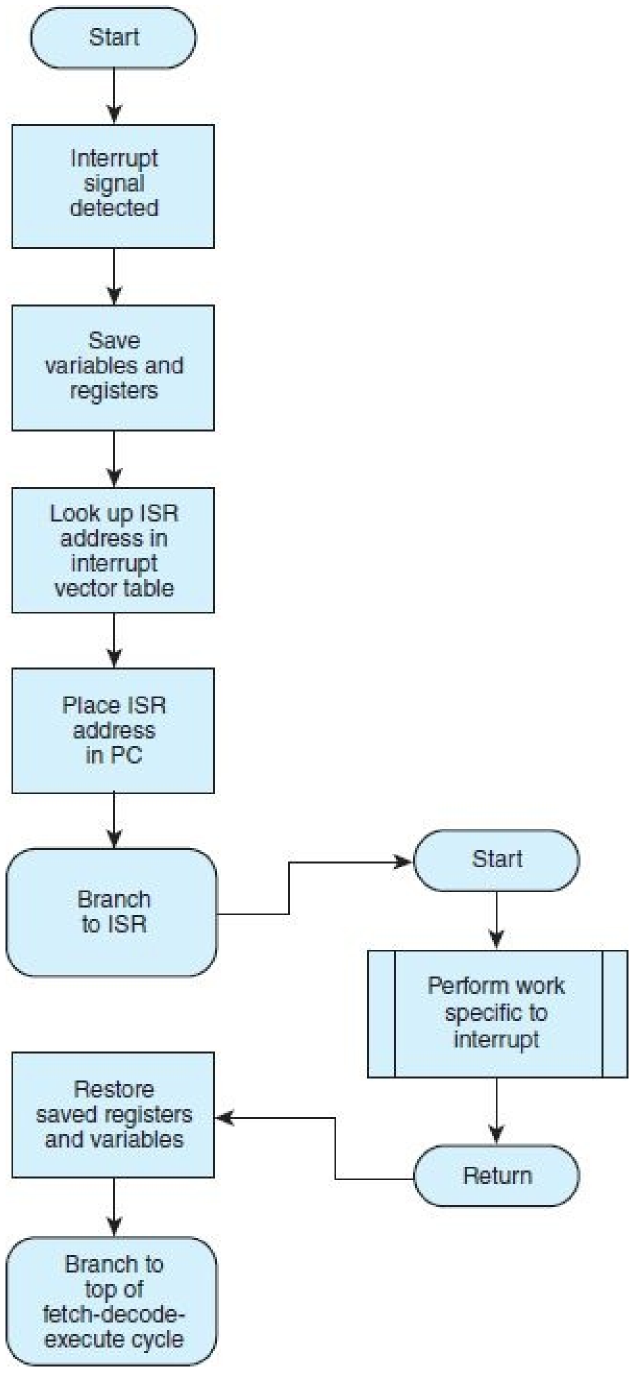

Start
Interrupt signal detected
Save variables and registers
Look up ISR address in interrupt vector table
Place ISR address in PC
Branch to ISR
Start
Perform work specific to interrupt
Return
Restore saved registers and variables
Branch to top of fetch-decode-execute cycle

FIGURE 7.4 Processing an Interrupt

We mentioned in Chapter 4 that I/O interrupts are usually maskable, and an I/O device can generate a nonmaskable interrupt if it encounters an error that it cannot handle, such as the removal or destruction of an I/O medium. We return to this topic later in the context of embedded systems in Chapter 10.

Memory-Mapped I/O

The design decisions made with regard to a system's I/O control method are enormously influential in determining the overall system architecture. If we decide to use programmed I/O, it is to our advantage to establish separate buses for memory traffic and I/O traffic so that the continued polling doesn't interfere with memory access. In this case, the system requires a set of distinct instructions for I/O control. Specifically, the system needs to know how to check the status of a device, transfer bytes to and from the device, and verify that the transfer has executed correctly. This approach has some serious limitations. For one, adding a new device type to the system may require changes to the processor's control store or hardwired control matrix.

A simpler and more elegant approach is **memory-mapped I/O** in which I/O devices and main memory share the same address space. Thus, each I/O device has its own reserved block of memory. Data transfers to and from the I/O device involve moving bytes to or from the memory address that is mapped to the device. Memory-mapped I/O therefore looks just like a memory access from the point of view of the CPU. This means we can use the same instructions to move data to and from both I/O and memory, greatly simplifying system design.

In small systems, the low-level details of the data transfers are offloaded to the I/O controllers built into the I/O devices themselves, as shown in Figure 7.1. The CPU does not need to concern itself with whether a device is ready, or counting the bytes in a transfer, or calculating error-correcting codes.

Direct Memory Access

With both programmed I/O and interrupt-driven I/O, the CPU moves data to and from the I/O device. During I/O, the CPU runs instructions similar to the following pseudocode:

```
WHILE More-input AND NOT (Error or Timeout)
       ADD 1 TO Byte-count
       IF Byte-count > Total-bytes-to-be-transferred THEN
         EXIT
       ENDIF
       Place byte in destination buffer
       Raise byte-ready signal
       Initialize timer

           REPEAT
             WAIT
           UNTIL Byte-acknowledged, Timeout, OR Error
    ENDWHILE
```

Clearly, these instructions are simple enough to be programmed in a dedicated chip. This is the idea behind **direct memory access** (**DMA**). When a system uses DMA, the CPU offloads execution of tedious I/O instructions. To effect the transfer, the CPU provides the DMA controller with the location of the bytes to be transferred, the number of bytes to be transferred, and the destination device or memory address. This communication usually takes place through special I/O registers on the CPU. A sample DMA configuration is shown in Figure 7.5. In this configuration, I/O and memory share the same address space, so it is one type of memory-mapped I/O.

Once the proper values are placed in memory, the CPU signals the DMA subsystem and proceeds with its next task, while the DMA takes care of the details of the I/O. After the I/O is complete (or ends in error), the DMA subsystem signals the CPU by sending it another interrupt.

As you can see in Figure 7.5, the DMA controller and the CPU share the bus. Only one of them at a time can have control of the bus, that is, be the **bus master**. Generally, I/O takes priority over CPU memory fetches for program instructions and data because many I/O devices operate within tight timing parameters. If they detect no activity within a specified period, they **timeout** and abort the I/O process. To avoid device timeouts, the DMA uses memory cycles that would otherwise be used by the CPU. This is called **cycle stealing**. Fortunately, I/O tends to create **bursty** traffic on the bus: data is sent in blocks, or clusters. The CPU should be granted access to the bus between bursts, though this access may not be of long enough duration to spare the system from accusations of "crawling during I/O."

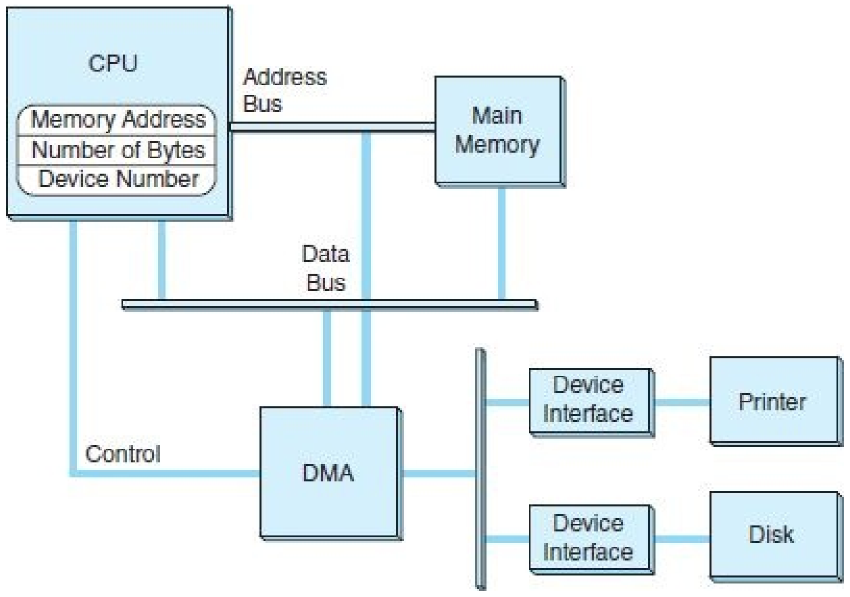


FIGURE 7.5 A Sample DMA Configuration

Figure 7.6 shows the activities of the CPU and the DMA. This swimlane diagram emphasizes how the DMA offloads I/O processing from the CPU.

Channel I/O

Programmed I/O transfers data one byte at a time. Interrupt-driven I/O can handle data one byte at a time or in small blocks, depending on the type of device participating in the I/O. Slower devices such as keyboards generate more interrupts per number of bytes transferred than disks or printers. DMA methods are all block-oriented, interrupting the CPU only after completion (or failure) of transferring a group of bytes. After the DMA signals the I/O completion, the CPU may give it the address of the next block of memory to be read from or written to. In the event of failure, the CPU is solely responsible for taking appropriate action. Thus, DMA I/O requires only a little less CPU participation than does interrupt-driven I/O. Such overhead is fine for small, single-user systems; however, it does not scale well to large, multiuser systems such as mainframe computers. Most large computer systems use an intelligent type of DMA interface known as an **I/O channel**. Although channel I/O is traditionally used on mainframe computers, it is becoming common on file servers and storage networks. Storage networks and other high-performance I/O implementations are presented in Chapter 13.

With **channel I/O**, one or more I/O processors control various I/O pathways called **channel paths**. Channel paths for “slow” devices such as terminals and printers can be combined (**multiplexed**), allowing management of several of these devices through only one controller. On IBM mainframes, a multiplexed channel path is called a **multiplexor channel**. Channels for disk drives and other “fast” devices are called **selector channels**.

I/O channels are driven by small CPUs called **I/O processors** (**IOPs**), which are optimized for I/O. Unlike DMA circuits, IOPs have the ability to execute programs that include arithmetic-logic and branching instructions. Figure 7.7 shows a simplified channel I/O configuration.

IOPs execute programs that are placed in main system memory by the host processor. These programs, consisting of a series of **channel command words** (**CCWs**), include not only the actual transfer instructions, but also commands that control the I/O devices. These commands include such things as device initializations, printer page ejects, and tape rewind commands, to name a few. Once the I/O program has been placed in memory, the host issues a **start subchannel** (**SSCH**) command, informing the

IOP of the location in memory where the program can be found. After the IOP has completed its work, it places completion information in memory and sends an interrupt to the CPU. The CPU then obtains the completion information and takes action appropriate to the return codes.

The principal distinction between standalone DMA and channel I/O lies in the intelligence of the IOP. The IOP negotiates protocols, issues device commands, and translates storage coding to memory coding, and can transfer entire files or groups of files independent of the host CPU. The host has only to create the program instructions for the I/O operation and tell the IOP where to find them.

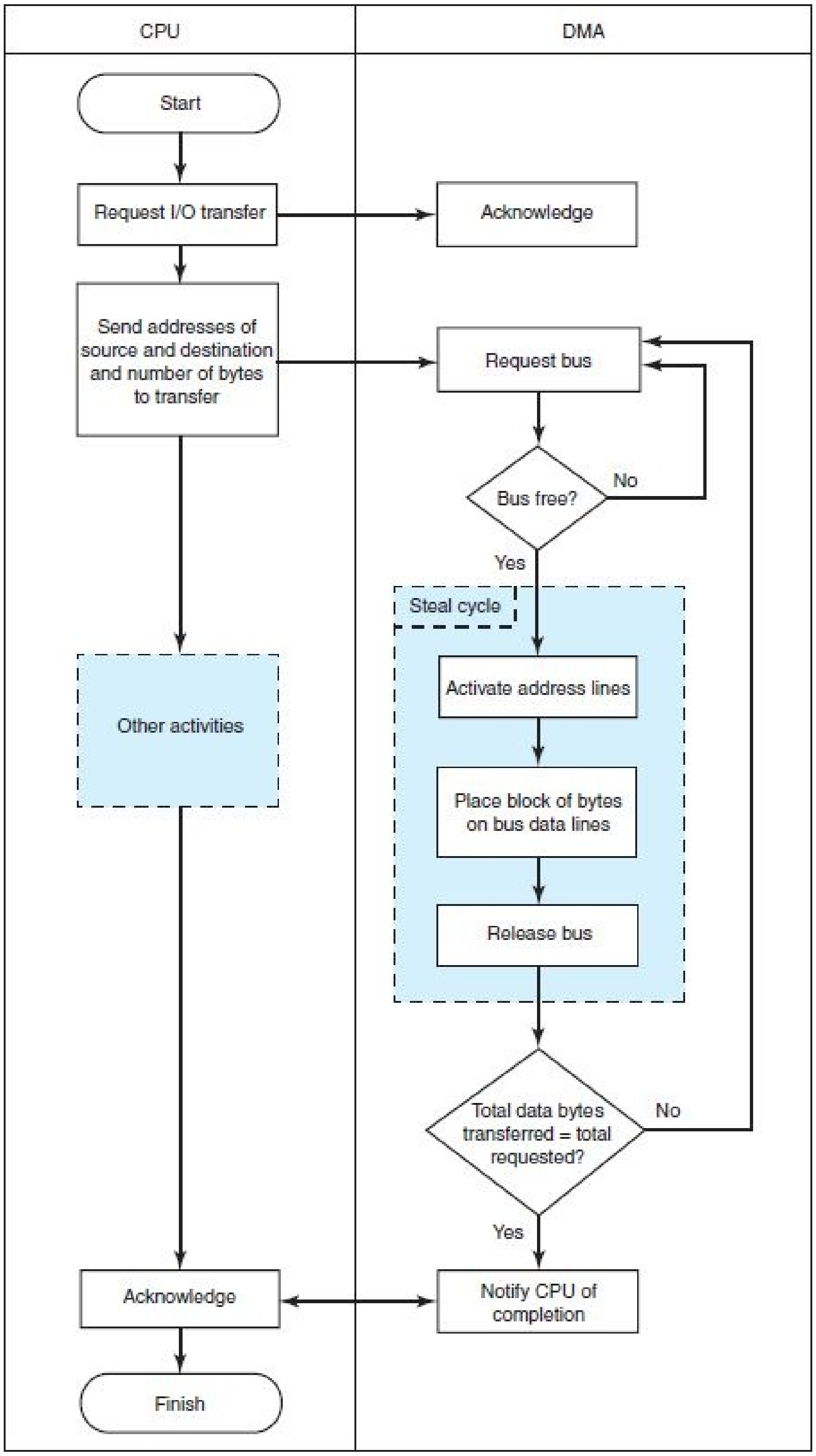


FIGURE 7.6 Swimlane Diagram Showing the Interaction of a CPU and a DMA

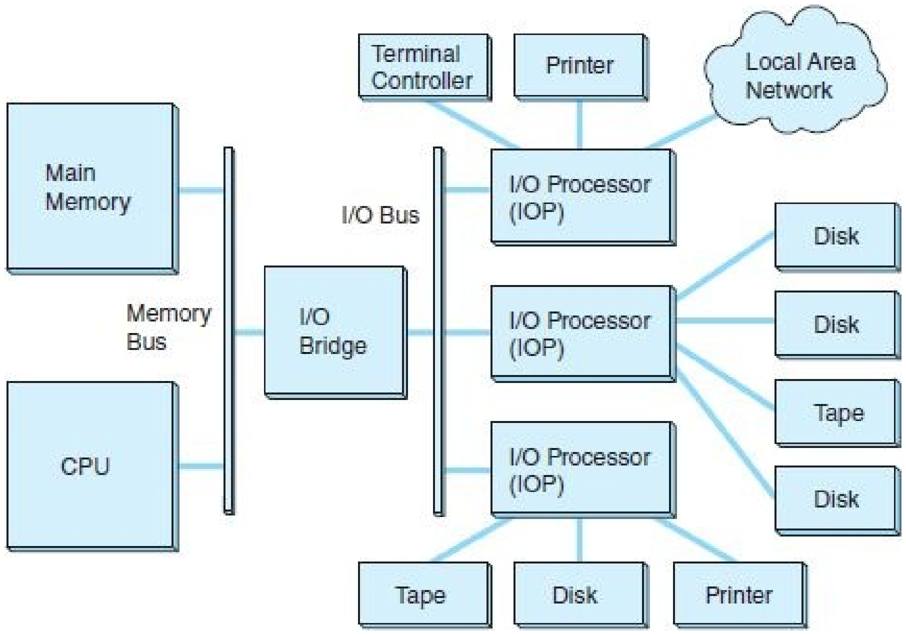


FIGURE 7.7 A Channel I/O Configuration

Like standalone DMA, an IOP must steal memory cycles from the CPU. Unlike standalone DMA, channel I/O systems are equipped with separate I/O buses, which help to isolate the host from the I/O operation. Thus, channel I/O is a type of **isolated I/O**. When copying a file from disk to tape, for example, the IOP uses the system memory bus only to fetch its instructions from main memory. The remainder of the transfer is effected using only the I/O bus. Because of its intelligence and bus isolation, channel I/O is used in high-throughput transaction processing environments, where its cost and complexity can be justified.

7.4.2 Character I/O Versus Block I/O

Pressing a key on a computer keyboard sets in motion a sequence of activities that process the keystroke as a single event (no matter how fast you type!). The reason for this is found within the mechanics of the keyboard. Each key controls a small switch that closes a connection in a matrix of conductors that runs horizontally and vertically beneath the keys. When a key switch closes, a distinct **scan code** is read by the keyboard circuitry. The scan code is then passed to a serial interface circuit, which translates the scan code into a character code. The interface places the character code in a keyboard buffer that is maintained in low memory. Immediately afterward, an I/O interrupt signal is raised. The characters wait patiently in the buffer until they are retrieved—one at a time—by a program (or until the buffer is reset). The keyboard circuits are able to process a new keystroke only after the old one is on its way to the buffer. Although it is certainly possible to press two keys at once, only one of the strokes can be processed at a time. Because of the random, sequential nature of character I/O as just described, it is best handled through interrupt-driven I/O processing.

Magnetic disks and tapes store data in blocks. Consequently, it makes sense to manage disk and tape I/O in block units. Block I/O lends itself to DMA or channel I/O processing. Blocks can be different sizes, depending on the particular hardware, software, and applications involved. Determining an ideal block size can be an important activity when a system is being tuned for optimum performance. High-performance systems handle large blocks more efficiently than they handle small blocks. Slower systems should manage bytes in smaller blocks; otherwise, the system may become unresponsive to user input during I/O.

7.4.3 I/O Bus Operation

In Chapter 1, we introduced you to computer bus architecture using the schematic shown in Figure 7.8. The important ideas conveyed by this diagram are:

- A system bus is a resource shared among many components of a computer system.
- Access to this shared resource must be controlled. This is why a control bus is required.

From our discussions in the preceding sections, it is evident that the memory bus and the I/O bus can be separate entities. In fact, it is often a good idea to separate them. One good reason for having memory on its own bus is that memory transfers can be **synchronous**, using some multiple of the CPU's clock cycles to retrieve data from main memory. In a properly functioning system, there is never an issue of the memory being offline or sustaining the same types of errors that afflict peripheral equipment, such as a printer running out of paper.

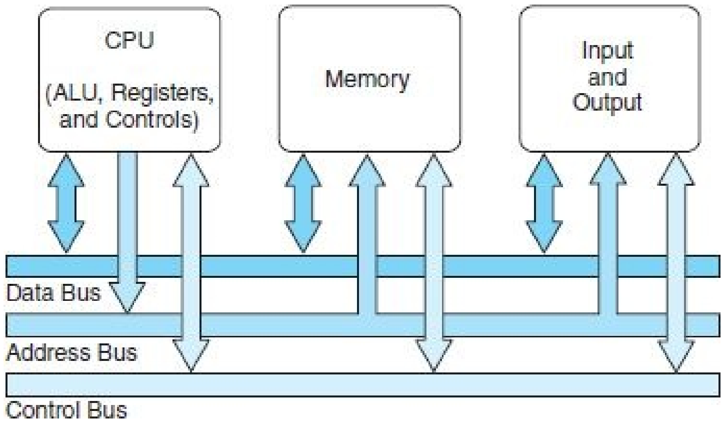


FIGURE 7.8 High-Level View of a System Bus

I/O buses, on the other hand, cannot operate synchronously. They must take into account the fact that I/O devices cannot always be ready to process an I/O transfer. I/O control circuits placed on the I/O bus and within the I/O devices negotiate with each other to determine the moment when each device may use the bus. Because these handshakes take place every time the bus is accessed, I/O buses are called **asynchronous**. We often distinguish synchronous from asynchronous transfers by saying that a synchronous transfer requires both the sender and the receiver to share a common clock for timing. But asynchronous bus protocols also require a clock for bit timing and to delineate signal transitions. This idea will become clear after we look at an example.

Consider, once again, the configuration shown in Figure 7.5. The connection between the DMA circuit and the device interface circuits is detailed in Figure 7.9, which shows the individual component buses.

Figure 7.10 gives the details of how the disk interface connects to all three buses. The address and data buses consist of a number of individual conductors, each of which carries one bit of information. The number of data lines determines the **width** of the bus. A data bus having eight data lines carries one byte at a time. The address bus has a sufficient number of conductors to uniquely identify each device on the bus.

The group of control lines shown in Figure 7.10 is the minimum that we need for our illustrative purpose. Real I/O buses typically have more than a dozen control lines. (The original IBM PC had more than 20!) Control lines coordinate the activities of the bus and its attached devices. To write data to the disk drive, our example bus executes the following sequence of operations:

1. The DMA circuit places the address of the disk controller on the address lines, and raises (asserts) the Request and Write signals.
2. With the Request signal asserted, decoder circuits in the controller interrogate the address lines.

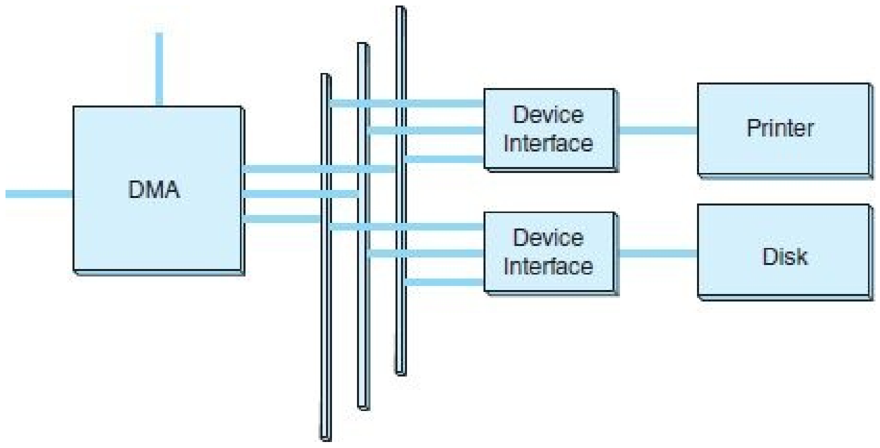


FIGURE 7.9 DMA Configuration Showing Separate Address, Data, and Control Lines

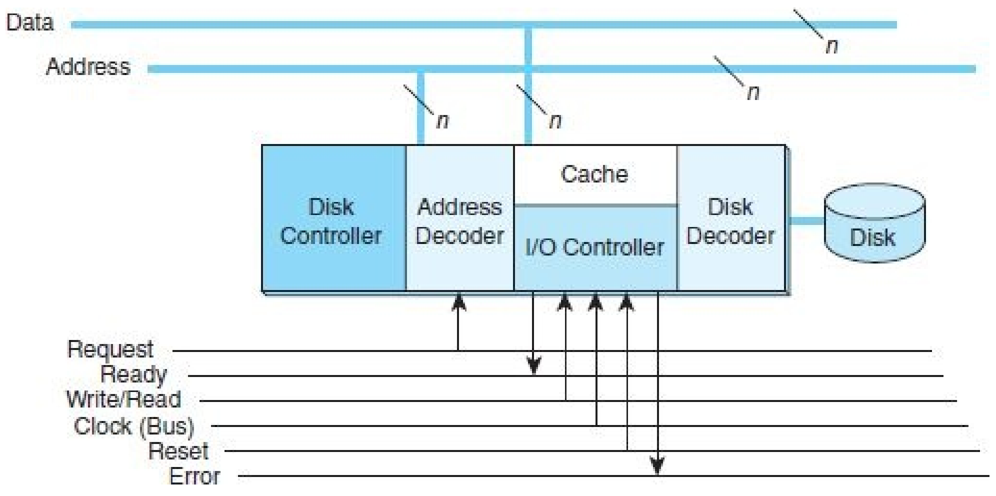


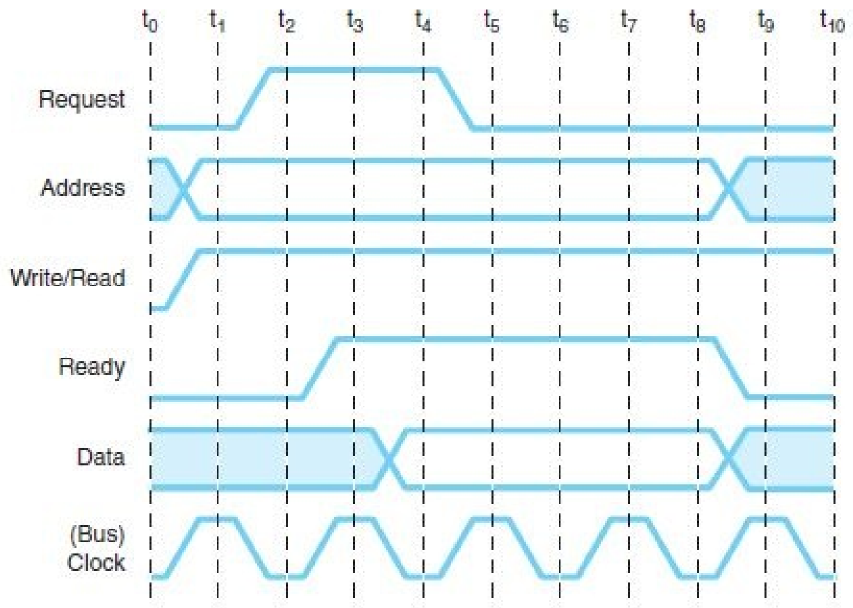
FIGURE 7.10 A Disk Controller Interface with Connections to the I/O Bus

3. Upon sensing its own address, the decoder enables the disk control circuits. If the disk is available for writing data, the controller asserts a signal on the Ready line. At this point, the handshake between the DMA and the controller is complete. With the Ready signal raised, no other devices may use the bus.
4. The DMA circuits then place the data on the lines and lower the Request signal.
5. When the disk controller sees the Request signal drop, it transfers the byte from the data lines to the disk buffer, and then lowers its Ready signal.

To make this picture clearer and more precise, engineers describe bus operation through **timing diagrams**. The timing diagram for our disk write operation is shown in Figure 7.11. The vertical lines, marked t_0 through t_{10}, specify the duration of the various signals. In a real timing diagram, an exact duration would be assigned to the timing intervals, usually in the neighborhood of 50ns. Signals on the bus can change only during a clock cycle transition. Notice that the signals shown in the diagram do not rise and fall instantaneously. This reflects the physical reality of the bus. A small amount of time must be allowed for the signal level to stabilize, or "settle down." This **settle time**, although small, contributes to a large delay over long I/O transfers.

The address and data lines in a timing diagram are rarely shown individually, but usually as a group. In our diagram, we imply the group by the use of a pair of lines. When the address and data lines transition from an active to an inactive state, we show the lines crossing. When the lines are inactive, we shade the space between them to make it clear that their state is undefined.

Many real I/O buses, unlike our example, do not have separate address and data lines. Because of the asynchronous nature of an I/O bus, the data lines can be used to hold the device address. All we need to do is add another control line that indicates whether the signals on the data lines represent an address or data. This approach contrasts to a memory bus, where the address and data must be simultaneously available.


t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
Request
Address
Write/Read
Ready
Data
(Bus) Clock

Time	Salient Bus Signal	Meaning
t_0	Assert Write	Bus is needed for writing (not reading)
t_0	Assert Address	Indicates where bytes will be written
t_1	Assert Request	Request write to address on address lines
t_2	Assert Ready	Acknowledges write request, bytes placed on data lines
t_3–t_7	Data Lines	Write data (requires several cycles)
t_8	Lower Ready	Release bus

FIGURE 7.11 A Bus Timing Diagram

7.5 DATA TRANSMISSION MODES

Bytes can be transmitted between a host and a peripheral device by sending one bit at a time or one byte at a time. These are called, respectively, **serial** and **parallel** transmission modes. Each transmission mode establishes a particular communication protocol between the host and the device interface.

BYTES, *DATA*, AND *INFORMATION* … FOR THE RECORD

Far too many people use the word *information* as a synonym for *data*, and *data* as a synonym for *bytes*. In fact, we have often used data as a synonym for bytes in this text for readability, hoping that the context makes the meaning clear. We are compelled, however, to point out that there is indeed a world of difference in the meanings of these words.

In its most literal sense, the word *data* is plural. It comes from the Latin singular *datum*. Hence, to refer to more than one datum, one properly uses the word *data*. It is in fact easy on our ears when someone says, “The recent mortality data *indicate* that people are now living longer than they did a century ago.” But we are at a loss to explain why we wince when someone says something like, “A page fault occurs when data *are* swapped from memory to disk.” When we are using *data* to refer to something stored in a computer system, we really are conceptualizing data as an “indistinguishable mass” in the same sense that we think of air and water. Air and water consist of various d iscrete elements called molecules. Similarly, a mass of data consists of discrete elements called data. No educated person who is fluent in English would say that she breathes *airs* or takes a bath in *waters*. So it seems reasonable to say, “… data *is* swapped from memory to disk.” Most scholarly sources (including the *American Heritage Dictionary*) now recognize *data* as a singular collective noun when used in this manner.

Strictly speaking, computer storage media don’t store data. They store bit patterns called bytes. For example, if you were to use a binary sector editor to examine the contents of a disk, you might see the pattern 01000100. So what knowledge have you gained on seeing it? For all you know, this bit pattern could be the binary code of a program, part of an operating system structure, a photograph, or even someone’s bank balance. If you know for a fact that the bits represent some numeric quantity (as opposed to program code or an image file, for example) and that it is stored in two’s complement binary, you can safely say that it is the decimal number 68. But you still don’t have a datum. Before you can have a datum, someone must ascribe some context to this number. Is it a person’s age or height? Is it the model number of a can opener? If you learn that 01000100 comes from a file that contains the temperature output from an automated weather station, then you have yourself a datum. The file on the disk can then be correctly called a *data file*.

By now, you’ve probably surmised that the weather data is expressed in degrees Fahrenheit, because no place on Earth has ever reached 68° Celsius. But you still don’t have information. The datum is meaningless: Is it the current temperature in Amsterdam? Is it the temperature that was recorded at 2:00 am three years ago in Miami? The *datum* 68 becomes *information* only when it has meaning to a human being.

Another plural Latin noun that has recently become recognized in singular usage is the word *media*. Formerly, educated people used this word only when they wished to refer to more than one *medium*. Newspapers are one kind of communication medium. Television is another. Collectively, they are media. But now some editors accept the singular usage, as in, “At this moment, the news media *is* gathering at the Capitol.”

Inasmuch as artists can paint using a watercolor medium or an oil paint medium, computer data recording equipment can write to an electronic medium such as tape or disk. Collectively, these are electronic media. But rarely will you find a practitioner who intentionally uses the term properly. It is much more common to encounter statements like, “Volume 2 ejected. Please place new *media* into the tape drive.” In this context, it’s debatable whether most people would even understand the directive “… place a new *medium* into the tape drive.”

Semantic arguments such as these are symptomatic of the kinds of problems computer professionals face when they try to express human ideas in digital form, and vice versa. There is bound to be something lost in the translation.

7.5.1 Parallel Data Transmission

Parallel communication systems operate in a manner analogous to the operation of a host memory bus. They require at least eight data lines (one for each bit) and one line for synchronization, sometimes called a **strobe**.

Parallel connections are effective over short distances—usually less than 30 feet—depending on the strength of the signal, the frequency of the signal, and the quality of the cable. At longer distances, signals in the cable begin to weaken, because of the internal resistance of the conductors. Electrical signal loss over time or distance is called **attenuation**. The problems associated with attenuation become clear by studying an example.

Figure 7.12 renders a simplified timing diagram for a parallel printer interface. The lines marked nStrobe and nAck are strobe and acknowledgment signals that are asserted when they carry low voltage. The Busy and Data signals are asserted when they carry high voltage. In other words, Busy and Data are positive logic signals, whereas nStrobe and nAck are negative logic signals. The data signal represents eight different lines. Each of these lines can be either high or low (signal 1 or 0). The signals on these lines are meaningless (shaded in diagram) before the nStrobe signal is asserted and after nAck is asserted. Arbitrary reference times are listed across the top of the diagram, t_0 through t_6. The difference between two consecutive times, Δt, determines the speed of transmission. Typically Δt will range between 1ms and 5ms.

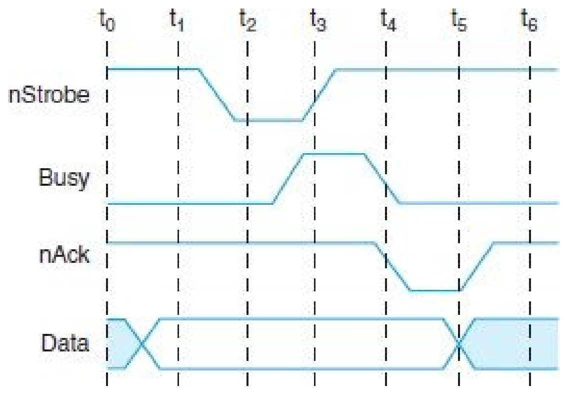


FIGURE 7.12 A Simplified Timing Diagram for a Parallel Printer

The signals illustrated in Figure 7.12 comprise the handshake that takes place between a printer interface circuit (on a host) and the host interface of a parallel printer. The process starts when a bit is placed on each of the eight data lines. Next, the busy line is checked to see that it is low. Once the busy line is low, the strobe signal is asserted so the printer will know that there is data on the data lines. As soon as the printer detects the strobe, it reads the data lines while raising the busy signal to prevent the host from placing more data on the data lines. After the printer has read the data lines, it lowers the busy signal and asserts the acknowledgment signal, nAck, to let the host know that the data has been received.

Notice that although the data signals are acknowledged, there is no guarantee of their correctness. Both the host and the printer assume that the signals received are the same as the signals that were sent. Over short distances, this is a fairly safe assumption. Over longer distances, this may not be the case.

Let's say that the bus operates on a voltage of ±5 volts. Anything between 0 and +5 volts is considered "high" and anything between 0 and –5 volts is considered "low." The host places voltages of +5 and –5 volts on the data lines, respectively, for each 1 and 0 of the data byte. Then it sets the strobe line to –5 volts.

With a case of "mild" attenuation, the printer could be slow to detect the nStrobe signal or the host could be slow to detect the nAck signal. This kind of sluggishness is hardly noticeable when printers are involved, but horrendously slow over a parallel disk interface, where we typically expect an instantaneous response.

Over a very long cable, we could end up with entirely different voltages at the printer end. By the time the signals arrive, “high” could be +1 volt and “low” could be –3 volts. If 1 volt is not sufficiently above the voltage threshold for a logical 1, we could end up with a 0 where a 1 should be, scrambling the output in the process. Also, over long distances, it is possible that the strobe signal gets to the printer before the data bits do. The printer then prints whatever is on the data lines at the time it detects the assertion of nStrobe. (The extreme case is when a text character is mistaken for a control character. This can cause remarkably bizarre printer behavior and the death of many trees.)

7.5.2 Serial Data Transmission

We have seen how a parallel data transmission moves one byte at a time along a data bus. A data line is required for each bit, and the data lines are activated by pulses in a separate strobe line. Serial data transmission differs from parallel data transmission in that only one conductor is used for sending data, one bit at a time, as pulses in a single data line. Other conductors can be provided for special signals, as defined in particular protocols. RS-232-C is one such serial protocol that requires separate signaling lines; the data, however, is sent over only one line (see Chapter 12). Serial storage interfaces incorporate these special signals into protocol frames exchanged along the data path. We will examine some serial storage protocols in Chapter 13. Generally speaking, serial data streams can be reliably sent faster over longer distances than parallel data. This makes serial transmission the method of choice for high-performance interfaces.

Serial transfer methods can also be used for time-sensitive **isochronous** data transfers. Isochronous protocols are used with real-time data such as voice and video signals. Because voice and video are intended for consumption by human senses, occasional transmission errors bear little notice. The approximate nature of the data permits less error control; hence, data can flow with minimal protocol-induced latency from its source to its destination.

7.6 MAGNETIC DISK TECHNOLOGY

Before the advent of disk drive technology, sequential media such as punched cards and magnetic or paper tape were the only kinds of durable storage available. If the data that someone needed were written at the trailing end of a tape reel, the entire volume had to be read—one record at a time. Sluggish readers and small system memories made this an excruciatingly slow process. Tape and cards were not only slow, but they also degraded rather quickly because of the physical and environmental stresses to which they were exposed. Paper tape often stretched and broke. Open reel magnetic tape not only stretched, but also was subject to mishandling by operators. Cards could tear, get lost, and warp.

In this technological context, it is easy to see how IBM fundamentally changed the computer world in 1956 when it deployed the first commercial disk-based computer called the **Random Access Method of Accounting and Control** computer, or **RAMAC** for short. By today’s standards, the disk in this early machine was incomprehensibly huge and slow. Each disk platter was 24 inches in diameter, containing only 50,000 7-bit characters of data on each surface. Fifty two-sided platters were mounted on a spindle that was housed in a flashy glass enclosure about the size of a small garden shed. The total storage capacity per spindle was a mere 5 million characters, and it took one full second, on average, to access data on the disk. The drive weighed more than a ton and cost millions of dollars to lease. (One could not *buy* equipment from IBM in those days.)

By contrast, in early 2000, IBM began marketing a high-capacity disk drive for use in palmtop computers and digital cameras. These disks were 1 inch (2.5cm) in diameter, held 1GB of data, and provided an average access time of 15ms. The drive weighed less than an ounce and retailed for less than $300! Since then, other manufacturers have produced 1-inch drives that are even less expensive and hold more data.

Disk drives are called **random** (sometimes **direct**) access devices because each unit of storage on a disk, the **sector**, has a unique address that can be accessed independently of the sectors around it. As shown in Figure 7.13, sectors are divisions of concentric circles called **tracks**. On most systems, every track contains exactly the same number of sectors. Each sector contains the same number of bytes. Hence, the data is written more "densely" at the center of the disk than at the outer edge. Some manufacturers pack more bytes onto their disks by making all sectors approximately the same size, placing more sectors on the outer tracks than on the inner tracks. This is called **zoned-bit** recording. Zoned-bit recording is rarely used because it requires more sophisticated drive control electronics than traditional systems.

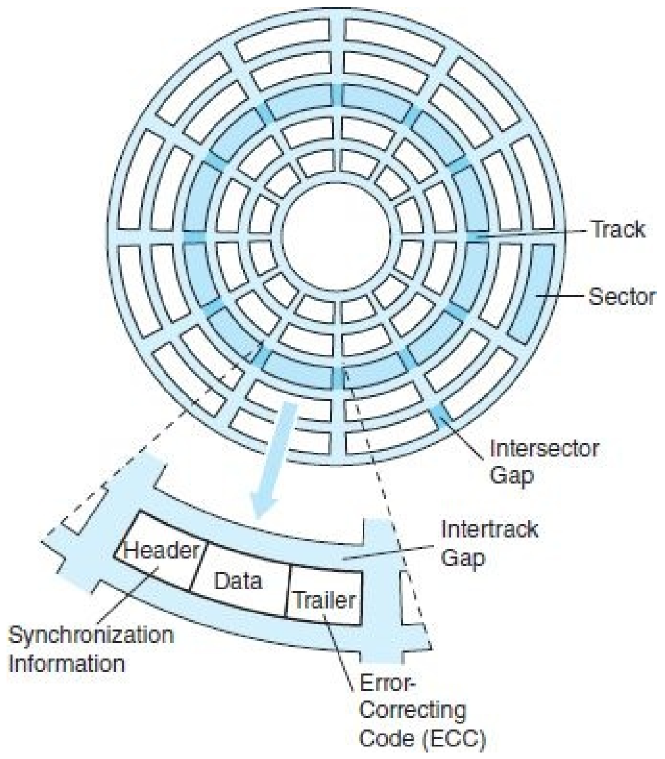


FIGURE 7.13 Disk Sectors Showing Intersector Gaps and Logical Sector Format

Disk tracks are consecutively numbered starting with track 0 at the outermost edge of the disk. Sectors, however, may not be in consecutive order around the perimeter of a track. They sometimes "skip around" to allow time for the drive circuitry to process the contents of a sector prior to reading the next sector. This is called **interleaving**. Interleaving varies according to the speed of rotation of the disk as well as the speed of the disk circuitry and its buffers. Most of today's fixed disk drives read disks a track at a time, not a sector at a time, so interleaving is becoming less common.

7.6.1 Rigid Disk Drives

Rigid ("hard" or fixed) disks contain control circuitry and one or more metal or glass disks called

platters to which a thin film of magnetizable material is bonded. Disk platters are stacked on a spindle, which is turned by a motor located within the drive housing. Disks can rotate as fast as 15,000 revolutions per minute (rpm), the most common speeds being 5,400rpm and 7,200rpm. Read/write heads are typically mounted on a rotating **actuator arm** that is positioned in its proper place by magnetic fields induced in coils surrounding the axis of the actuator arm (see Figure 7.14). When the actuator is energized, the entire comb of read/write heads moves toward or away from the center of the disk.

Despite continual improvements in magnetic disk technology, it is still impossible to mass-produce a completely error-free medium. Although the probability of error is small, errors must, nevertheless, be expected. Two mechanisms are used to reduce errors on the surface of the disk: special coding of the data itself and error-correcting algorithms. (This special coding and some error-correcting codes were discussed in Chapter 2.) These tasks are handled by circuits built into the disk controller hardware. Other circuits in the disk controller take care of head positioning and disk timing.

In a stack of disk platters, all of the tracks directly above and below each other form a **cylinder**. A comb of read/write heads accesses one cylinder at a time. Cylinders describe circular areas on each disk.

Typically, there is one read/write head per usable surface of the disk. (Older disks—particularly removable disks—did not use the top surface of the top platter or the bottom surface of the bottom platter.) Fixed disk heads never touch the surface of the disk. Instead, they float above the disk surface on a cushion of air only a few microns thick. When the disk is powered down, the heads retreat to a safe place. This is called **parking the heads**. If a read/write head were to touch the surface of the disk, the disk would become unusable. This condition is known as a **head crash**.

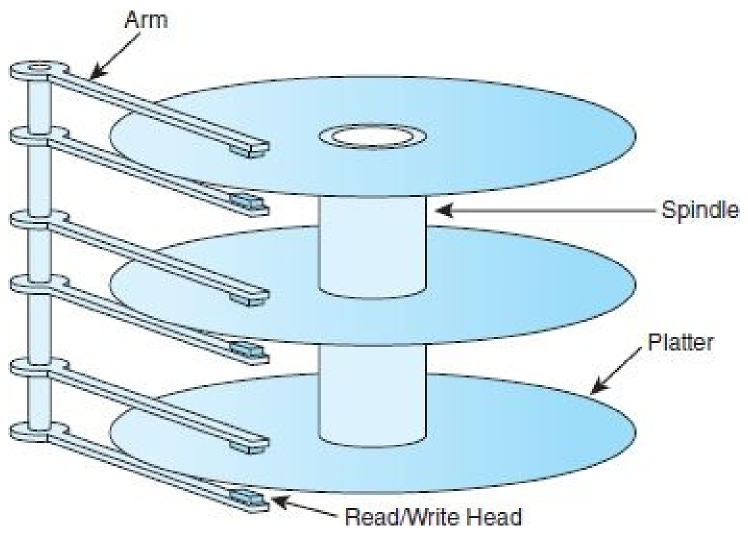


FIGURE 7.14 Rigid Disk Actuator (with Read/Write Heads) and Disk Platters

Head crashes were common during the early years of disk storage. First-generation disk drive mechanical and electronic components were costly with respect to the price of disk platters. To provide the most storage for the least money, computer manufacturers made disk drives with removable disks called **disk packs**. When the drive housing was opened, airborne impurities, such as dust and water vapor, would enter the drive housing. Consequently, large head-to-disk clearances were required to prevent these impurities from causing head crashes. (Despite these large head-to-disk clearances, frequent crashes persisted, with some companies experiencing as much downtime as uptime.) The price paid for the large head-to-disk clearance was substantially lower data density. The greater the distance between the head and the disk, the stronger the charge in the flux coating of the disk must be for the data to be readable. Stronger magnetic charges require more particles to participate in a flux transition, resulting in lower data density for the drive.

Eventually, cost reductions in controller circuitry and mechanical components permitted widespread use of sealed disk units. IBM invented this technology, which was developed under the code name "Winchester." **Winchester** soon became a generic term for any sealed disk unit. Today, with removable-pack drives no longer being manufactured, we have little need to make the distinction. Sealed drives permit closer head-to-disk clearances, increased data densities, and faster rotational speeds. These factors constitute the performance characteristics of a rigid disk drive.

Seek time is the time it takes for a disk arm to position itself over the required track. Seek time does not include the time that it takes for the head to read the disk directory. The **disk directory** maps logical file information, for example, my_story.doc, to a physical sector address, such as cylinder 7, surface 3, sector 72. Some high-performance disk drives practically eliminate seek time by providing a read/write head for each track of each usable surface of the disk. With no movable arms in the system, the only delays in accessing data are caused by rotational delay.

Rotational delay is the time it takes for the required sector to position itself under a read/write head. The sum of the rotational delay and seek time is known as the **access time**. If we add to the access time the time it takes to actually read the data from the disk, we get a quantity known as **transfer time**, which, of course, varies depending on how much data is read. **Latency** is a direct function of rotational speed. It is a measure of the amount of time it takes for the desired sector to move beneath the read/write head after the disk arm has positioned itself over the desired track. Usually cited as an average, it is calculated as:

$$\frac{\dfrac{60 \text{ seconds}}{\text{disk rotation speed}} \times \dfrac{1000 \text{ ms}}{\text{second}}}{2}$$

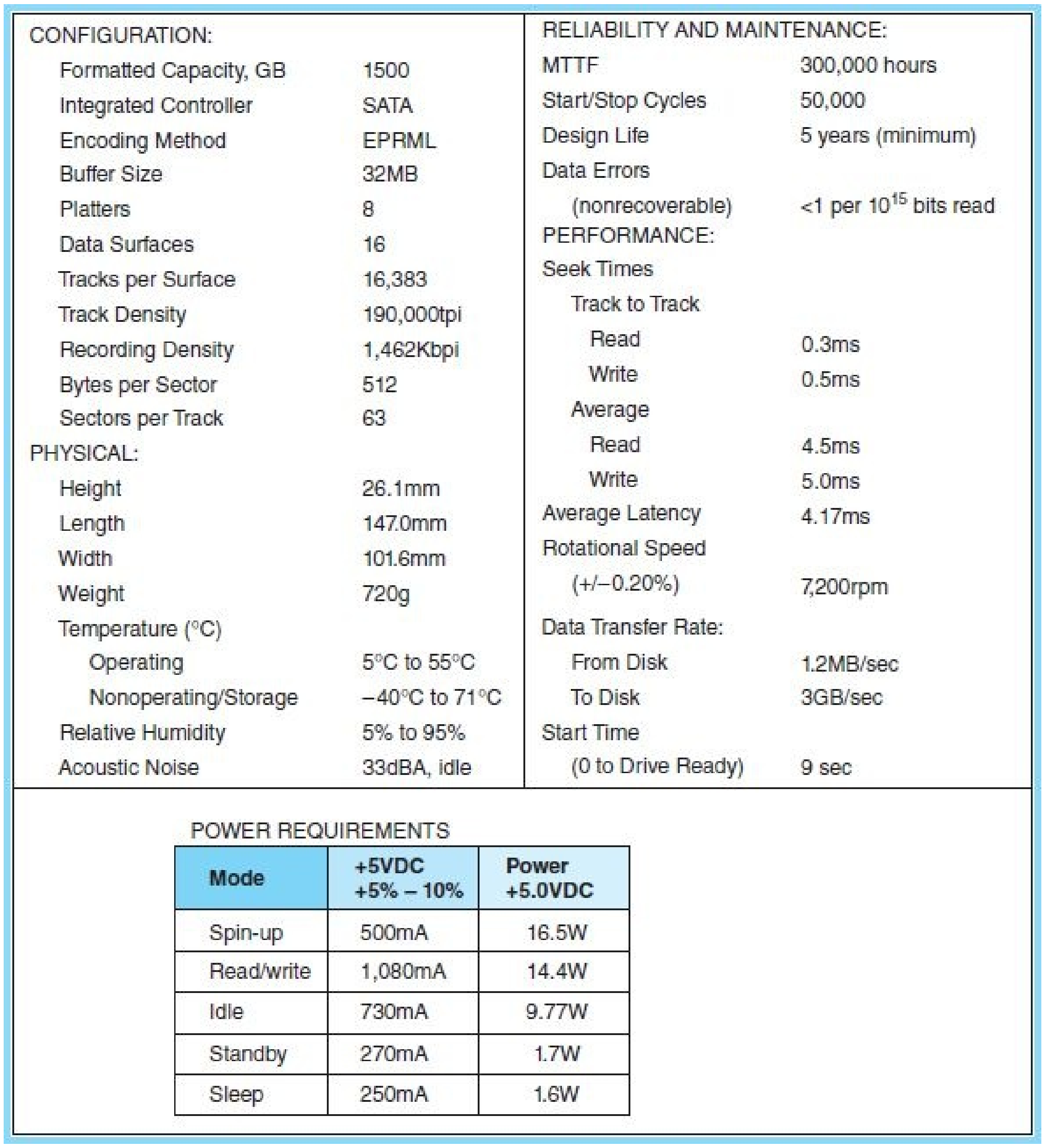
To help you appreciate how all of this terminology fits together, we have provided a typical disk specification as Figure 7.15.

Because the disk directory must be read prior to every data read or write operation, the location of the directory can have a significant effect on the overall performance of the disk drive. Outermost tracks have the lowest bit density per areal measure; hence, they are less prone to bit errors than the innermost tracks. To ensure the best reliability, disk directories can be placed at the outermost track, track 0. This means, for every access, the arm has to swing out to track 0 and then back to the required data track. Performance therefore suffers from the wide arc made by the access arms.

Improvements in recording technology and error-correction algorithms permit the directory to be placed in the location that gives the best performance: at the innermost track. This substantially reduces arm movement, giving the best possible throughput. Some, but not all, modern systems take advantage of center track directory placement.

Directory placement is one of the elements of the logical organization of a disk. A disk's logical organization is a function of the operating system that uses it. A major component of this logical organization is the way in which sectors are mapped. Fixed disks contain so many sectors that keeping tabs on each one is infeasible. Consider the disk described in our data sheet. Each track contains 746 sectors. There are 48,000 tracks per surface and 8 surfaces on the disk. This means there are more than 286 million sectors on the disk. An allocation table listing the status of each sector (the status being recorded in 1 byte) would therefore consume more than 200MB of disk space. Not only is this a lot of disk space spent for overhead, but reading this data structure would consume an inordinate amount of time whenever we needed to check the status of a sector. (This is a frequently executed task.) For this reason, operating systems address sectors in groups, called **blocks** or **clusters**, to make file management simpler.

The number of sectors per block determines the size of the allocation table. The smaller the size of the allocation block, the less wasted space there is when a file doesn't fill the entire block; however, smaller block sizes make the allocation tables larger and slower. We will look deeper into the relationship between directories and file allocation structures in our discussion of floppy disks in the next section.



CONFIGURATION:	
Formatted Capacity, GB	1500
Integrated Controller	SATA
Encoding Method	EPRML
Buffer Size	32MB
Platters	8
Data Surfaces	16
Tracks per Surface	16,383
Track Density	190,000tpi
Recording Density	1,462Kbpi
Bytes per Sector	512
Sectors per Track	63
PHYSICAL:	
Height	26.1mm
Length	147.0mm
Width	101.6mm
Weight	720g
Temperature (°C)	
Operating	5°C to 55°C
Nonoperating/Storage	−40°C to 71°C
Relative Humidity	5% to 95%
Acoustic Noise	33dBA, idle

RELIABILITY AND MAINTENANCE:	
MTTF	300,000 hours
Start/Stop Cycles	50,000
Design Life	5 years (minimum)
Data Errors	
(nonrecoverable)	<1 per 10^{15} bits read
PERFORMANCE:	
Seek Times	
Track to Track	
Read	0.3ms
Write	0.5ms
Average	
Read	4.5ms
Write	5.0ms
Average Latency	4.17ms
Rotational Speed	
(+/−0.20%)	7,200rpm
Data Transfer Rate:	
From Disk	1.2MB/sec
To Disk	3GB/sec
Start Time	
(0 to Drive Ready)	9 sec

POWER REQUIREMENTS

Mode	+5VDC +5% – 10%	Power +5.0VDC
Spin-up	500mA	16.5W
Read/write	1,080mA	14.4W
Idle	730mA	9.77W
Standby	270mA	1.7W
Sleep	250mA	1.6W

FIGURE 7.15 A Typical Rigid Disk Specification as Provided by a Disk Drive Manufacturer

One final comment about the disk specification shown in Figure 7.15: You can see that it also includes estimates of disk reliability under the heading of "Reliability and Maintenance." According to the manufacturer, this particular disk drive is designed to operate for five years and tolerate being stopped and started 50,000 times. Under the same heading, the **mean time to failure** (**MTTF**) is given as 300,000 hours. Surely this figure cannot be taken to mean that the expected value of the disk life is 300,000 hours—this is just over 34 years if the disk runs continuously. The specification states that the drive is designed to last only five years. This apparent anomaly owes its existence to statistical quality control methods commonly used in the manufacturing industry. Unless the disk is manufactured under a government contract, the exact method used for calculating the MTTF is at the discretion of the manufacturer. Usually the process involves taking random samples from production lines and running the disks under less-than-ideal conditions for a certain number of hours, typically more than 100. The number of failures are then plotted against probability curves to obtain the resulting MTTF figure. In short, the "design life" number is much more credible and understandable.

7.6.2 Solid State Drives

The limitations of magnetic disks are many. To begin with, it takes a lot longer to retrieve data from a magnetic disk than from main memory—around a million times longer. Magnetic disks are fragile. Even "ruggedized" models can break under extreme shock. Their many moving parts are susceptible to wear and failure. And—particularly problematic for mobile devices—magnetic disks are power hungry.

The clear solution to these problems is to replace the hard disk with nonvolatile RAM. Indeed, this switch happened decades ago in ultra-high-performance computers. Only recently has the price of memory become low enough to make this an attractive option for industrial, military, and consumer products.

Solid state drives (**SSDs**) consist of a microcontroller and a type of NAND- or NOR-based memory arrays called flash memory. Flash memory is distinguished from standard memory by the fact that it must be first erased ("in a flash") before it can be written to. NOR-based flash is byte addressable, making it more costly than NAND-based flash memory, which is instead organized in blocks (called pages), much like a magnetic disk.

Certainly we all enjoy our pocket-sized memory sticks, thumb drives, and jump drives. We are scarcely awed by the fact that these small devices can store an entire library on our key rings. Because of their low power consumption and durability, flash drives are now routinely replacing standard magnetic disks in portable devices. These applications also benefit from a performance boost: Access times and transfer rates of SSDs are typically 100 times faster than traditional disk drives. SSD accesses, however, are still slower than RAM by a factor of 100,000.

Although the data capacity of SSDs is nearing that of magnetic disks, SSDs tend to be roughly 2 to 3 times as costly. One would expect that the price gap will close considerably with the continued advances in SSD technology. For data centers, the increased cost of large arrays of SSDs may be offset by reduced electricity and air-conditioning costs.

Besides cost, another disadvantage of SSDs versus magnetic disks is that the bit cells of flash storage wear out after 30,000 to 1,000,000 updates to a page. This may seem like a long duty cycle, except that SSDs can store highly volatile data, such as virtual memory pagefiles. Standard magnetic disks tend to reuse the same disk sectors repeatedly, gradually filling up the disk. If this approach is used with an SSD, parts of the drive will eventually wear out and become unusable. So to extend the life of the disk, a technique called **wear leveling** is used to distribute data and erase/write cycles evenly over the entire disk. The drive's built-in microcontroller manages the free space on the disk to ensure that pages are reused in a round-robin-type rotation. This approach does offer a slight performance advantage; since a page must be erased before it can be written to, so the erasing and writing can occur simultaneously if the same page is not reused.

SSDs that are designed for use in servers are called **enterprise-grade SSDs.** These SSDs include cache memory for optimal performance and a small backup power source so that cache contents can be committed to flash in the event of a power failure. Figure 7.16 is a photograph of an Intel 910 800GB SSD. The microcontroller and flash memory chips dominate the card's real estate. The drive mounts in a server as simply as any other bus–attached card.

SSD specifications share many common elements with HDDs. Figure 7.17 is an example of a data sheet for an enterprise-grade SSD. In comparing the specifications in Figure 7.17 with those in Figure 7.15, you can see that there is no reference to platters, or rotational speed, or anything else having to do with the spinning disk form factor. However, the drive's physical characteristics, access time, transfer rate, and power consumption are still significant metrics.

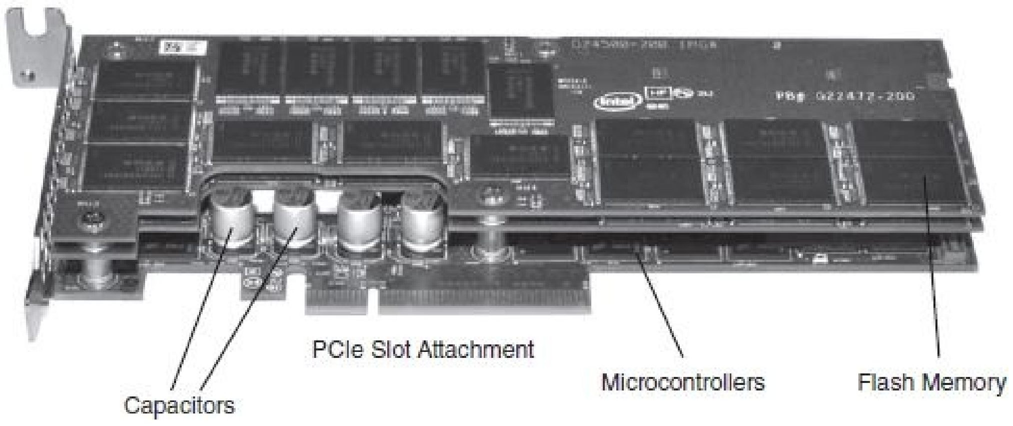
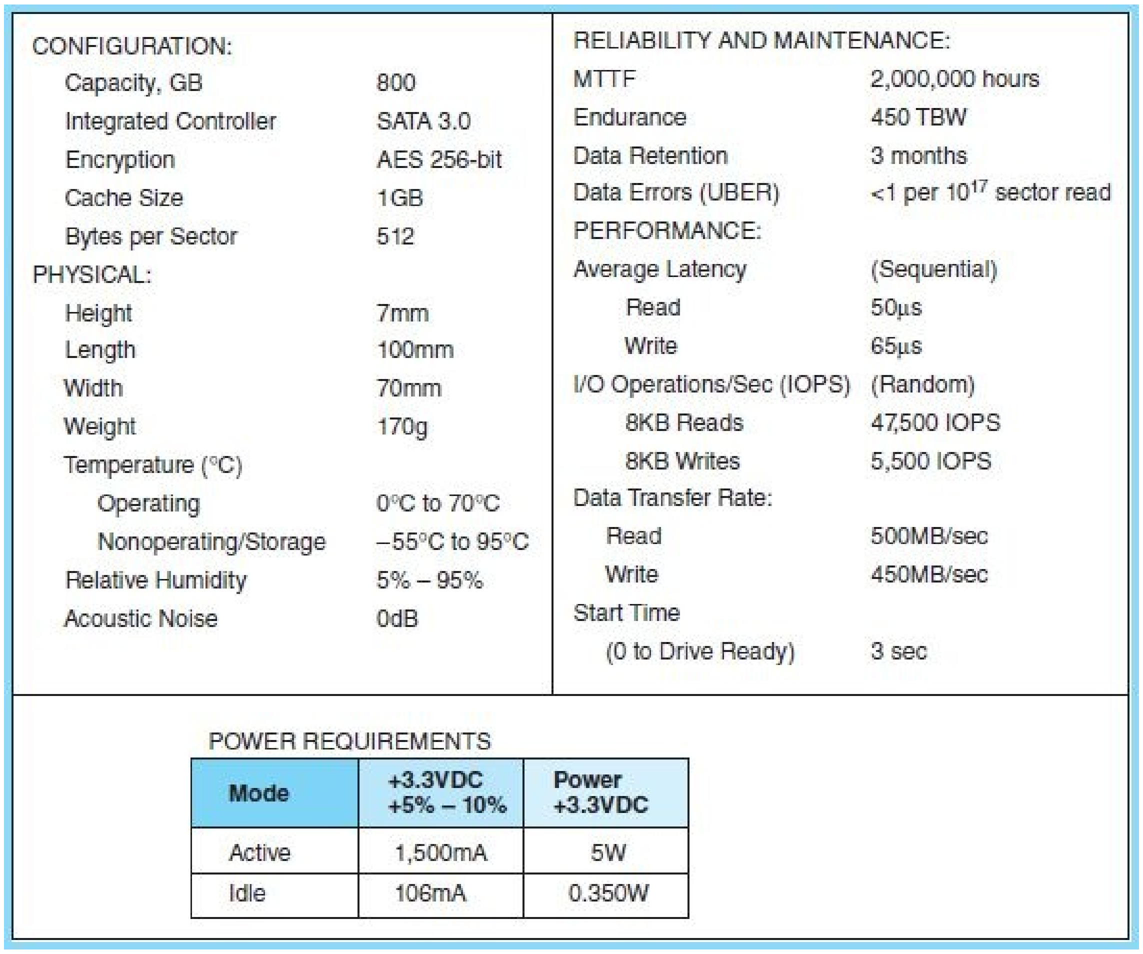


FIGURE 7.16 Intel 910 800GB SSD
Courtesy Intel Corporation.



CONFIGURATION:	
Capacity, GB	800
Integrated Controller	SATA 3.0
Encryption	AES 256-bit
Cache Size	1GB
Bytes per Sector	512
PHYSICAL:	
Height	7mm
Length	100mm
Width	70mm
Weight	170g
Temperature (°C)	
Operating	0°C to 70°C
Nonoperating/Storage	–55°C to 95°C
Relative Humidity	5% – 95%
Acoustic Noise	0dB

RELIABILITY AND MAINTENANCE:	
MTTF	2,000,000 hours
Endurance	450 TBW
Data Retention	3 months
Data Errors (UBER)	<1 per 10^{17} sector read
PERFORMANCE:	
Average Latency	(Sequential)
Read	50µs
Write	65µs
I/O Operations/Sec (IOPS)	(Random)
8KB Reads	47,500 IOPS
8KB Writes	5,500 IOPS
Data Transfer Rate:	
Read	500MB/sec
Write	450MB/sec
Start Time	
(0 to Drive Ready)	3 sec

POWER REQUIREMENTS

Mode	+3.3VDC +5% – 10%	Power +3.3VDC
Active	1,500mA	5W
Idle	106mA	0.350W

FIGURE 7.17 An SSD Data Sheet

The **Joint Electron Devices Engineering Council** (**JEDEC**) sets standards for SSD performance and reliability metrics. Two of the most important of these are **Unrecoverable Bit Error Ratio** (**UBER**) and **terabytes written** (**TBW**). UBER is calculated by dividing the number of data errors by the number of bits read using a simulated lifetime workload. TBW is the number of terabytes that can be written to the disk before the disk fails to meet specifications for speed and error rates. TBW is a measure of disk endurance (or service life) and UBER is a measure of disk reliability.

The cost of enterprise-grade SSDs makes sense where fast data retrieval is crucial. It is now common practice to push HDD performance to its limits through a practice known as **short stroking**. Short stroking involves the installation of many extra disk drives, each of which utilizes only a small percentage of its cylinders, thus keeping arm motion to a minimum. With less arm motion, access time decreases, saving a few milliseconds on each disk access. Thus, cost comparisons between HDDs and SSDs in the enterprise must take into account the number of gigabytes of useful storage, general reliability, and the low power consumption of enterprise SDDs.

As drive prices continue to plummet, SSDs are sure to start showing up in less demanding business environments. By some estimates, SSD costs will reach parity with HDDs well before the end of the 2010s.

7.7 OPTICAL DISKS

Optical storage systems offer (practically) unlimited data storage at a cost that is competitive with tape. Optical disks come in a number of formats, the most popular format being the ubiquitous **CD-ROM** (**compact disc-read only memory**), which can hold more than 0.5GB of data. CD-ROMs are a read-only medium, making them ideal for software and data distribution. **CD-R** (**CD-recordable**), **CD-RW** (**CD-rewritable**), and **WORM** (**write once read many**) disks are optical storage devices often used for long-term data archiving and high-volume data output. CD-R and WORM offer unlimited quantities of tamper-resistant storage for documents and data. For long-term archival storage of data, some computer systems send output directly to optical storage rather than paper or microfiche. This is called **computer output laser disc** (**COLD**). Robotic storage libraries called **optical jukeboxes** provide direct access to myriad optical disks. Jukeboxes can store dozens to hundreds of disks, for total capacities of 50GB to 1,200GB and beyond. Proponents of optical storage claim that optical disks, unlike magnetic media, can be stored for 100 years without noticeable degradation. (Who could possibly challenge this claim?)

7.7.1 CD-ROM

CD-ROMs are polycarbonate (plastic) disks 120mm (4.8 inches) in diameter to which a reflective aluminum film is applied. The aluminum film is sealed with a protective acrylic coating to prevent abrasion and corrosion. The aluminum layer reflects light that emits from a green laser diode situated beneath the disk. The reflected light passes through a prism, which diverts the light into a photodetector. The photodetector converts pulses of light into electrical signals, which it sends to decoder electronics in the drive (see Figure 7.18).

Compact discs are written from the center to the outside edge using a single spiraling track of bumps in the polycarbonate substrate. These bumps are called **pits** because they look like pits when viewed from the top surface of the CD. Lineal spaces between the pits are called **lands**. Pits measure 0.5μm wide and are between 0.83μm and 3.56μm long. (The edges of the pits correspond to binary 1s.) The bump formed by the underside of a pit is as high as one-quarter of the wave-length of the light produced by the laser diode. This means that the bump interferes with the reflection of the laser beam in such a way that the light bouncing off the bump exactly cancels out light incident from the laser. This results in pulses of light and dark, which are interpreted by drive circuitry as binary digits.

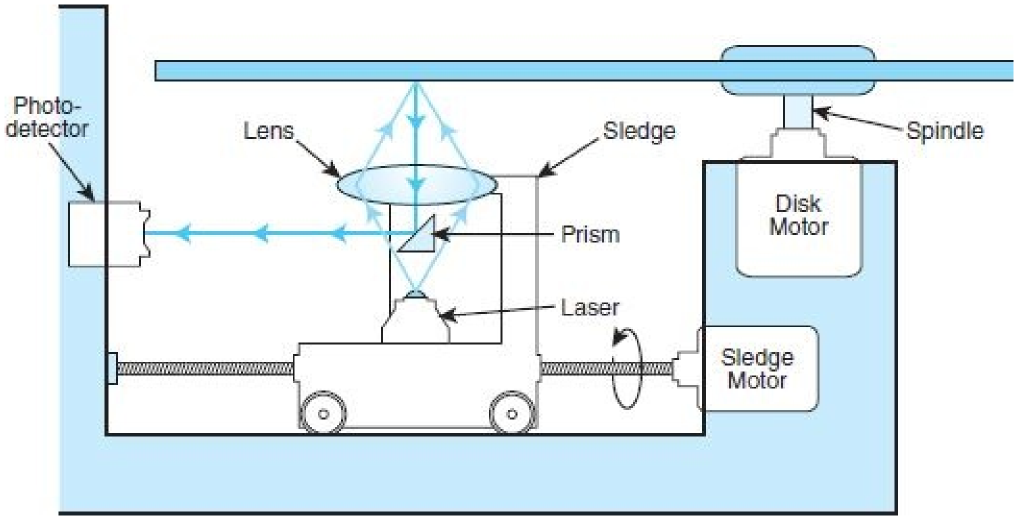


FIGURE 7.18 The Internals of a CD-ROM Drive

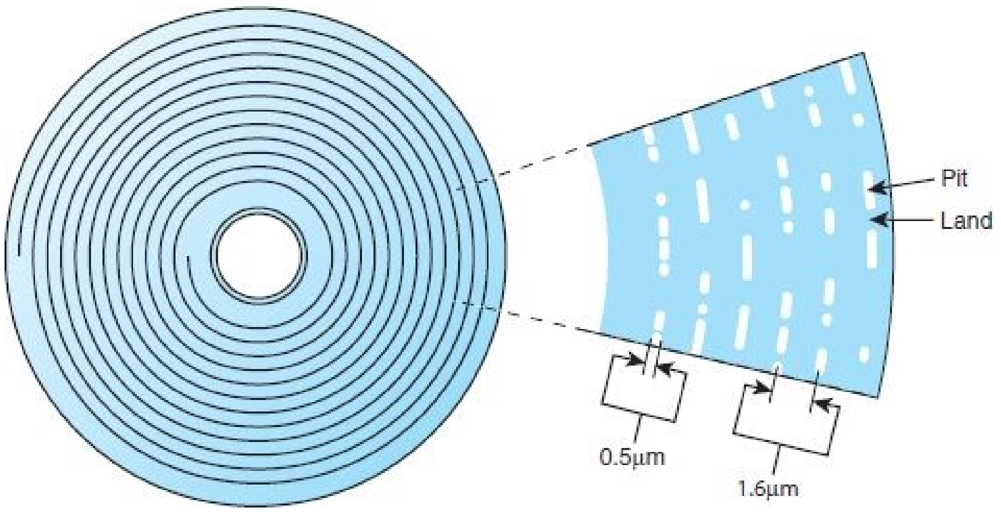


FIGURE 7.19 CD Track Spiral and Track Enlargement

The distance between adjacent turns of the spiral track, the **track pitch**, must be at least 1.6µm (see Figure 7.19). If you could "unravel" a CD-ROM or audio CD track and lay it on the ground, the string of pits and lands would extend nearly 5 miles (8km). (Being only 0.5µm wide—less than half the thickness of a human hair—it would be barely visible to the unaided eye.)

Although a CD has only one track, a string of pits and lands spanning 360° of the disk is referred to as a track in most optical disk literature. Unlike magnetic storage, tracks at the center of the disk have the same bit density as tracks at the outer edge of the disk.

CD-ROMs were designed for storing music and other sequential audio signals. Data storage applications were an afterthought, as you can see by the data sector format in Figure 7.20. Data is stored in 2,352-byte chunks called sectors that lie along the length of the track. Sectors are made up of 98 588-bit primitive units called **channel frames**. As shown in Figure 7.21, channel frames consist of synchronizing information, a header, and 33 17-bit symbols for a payload. The 17-bit symbols are encoded using an RLL(2, 10) code called **EFM** (**eight-to-fourteen modulation**). The disk drive electronics read and interpret (**demodulate**) channel frames to create yet another data structure called a **small frame**. Small frames are 33 bytes wide, 32 bytes of which are occupied by user data. The remaining byte is used for **subchannel** information. There are eight subchannels, named P, Q, R, S, T, U, V, and W. All except P (which denotes starting and stopping times) and Q (which contains control information) have meaning only for audio applications.

Most compact discs operate at **constant linear velocity** (**CLV**), which means that the rate at which sectors pass over the laser remains constant regardless of whether those sectors are at the beginning or the end of the disk. The constant velocity is achieved by spinning the disk slower when accessing the outermost tracks than the innermost. A sector number is addressable by the number of minutes and seconds of track that lie between it and the beginning (the center) of the disk. These "minutes and seconds" are calibrated under the assumption that the CD player processes 75 sectors per second. Computer CD-ROM drives are much faster than that with speeds up to 52 times (52×) the speed of audio CDs, 7.8MBps (with faster speeds sure to follow). To locate a particular sector, the sledge moves perpendicular to the disk track, taking its best guess as to where a particular sector may be. After an arbitrary sector is read, the head follows the track to the desired sector.

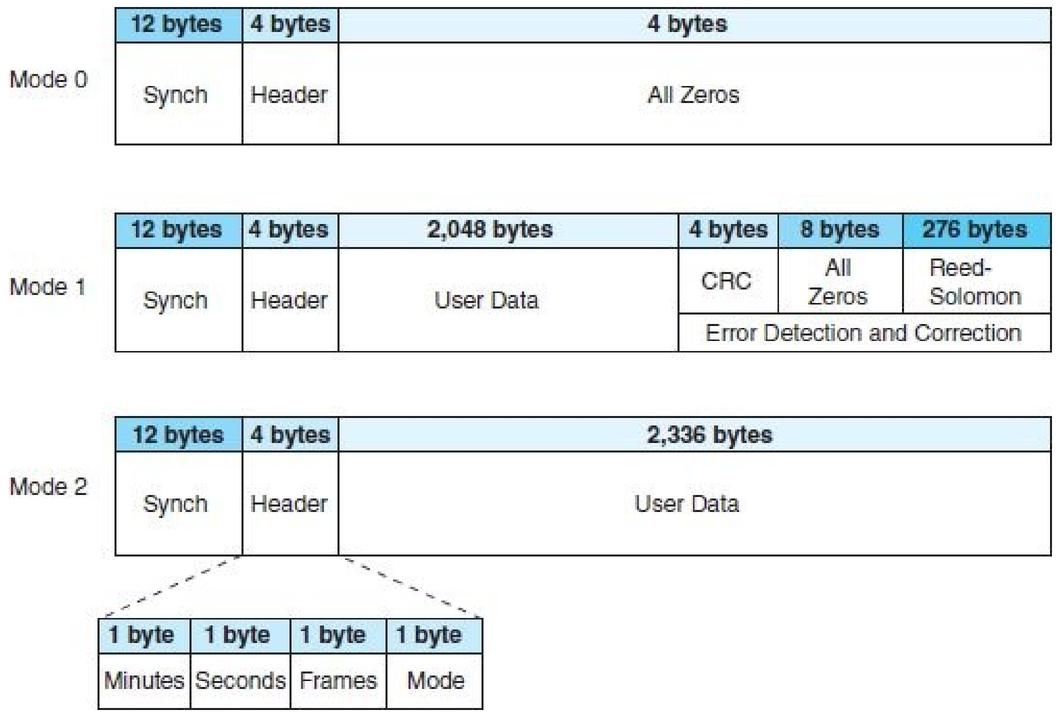


FIGURE 7.20 CD Data Sector Formats

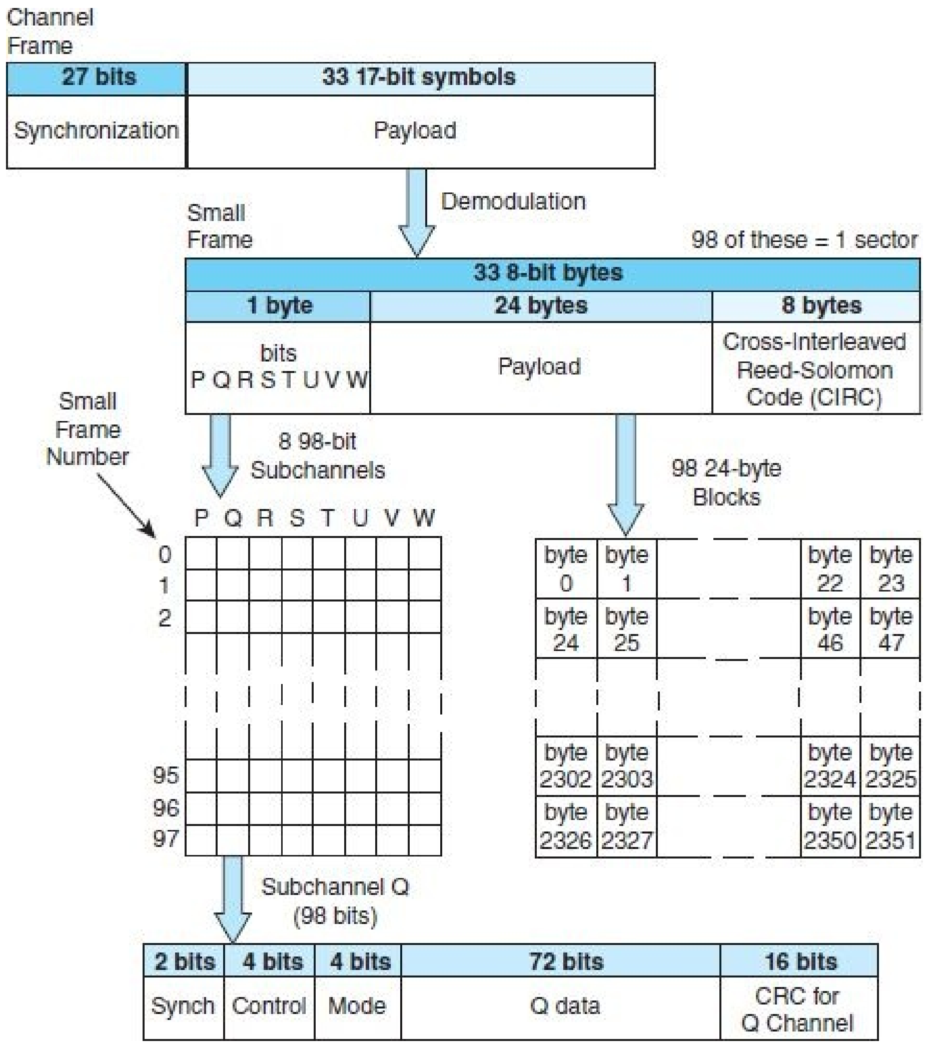


FIGURE 7.21 CD Physical and Logical Formats

Sectors can have one of three different formats, depending on which mode is used to record the data. There are three different modes. Modes 0 and 2, intended for music recording, have no error-correction capabilities. Mode 1, intended for data recording, sports two levels of error detection and correction. These formats are shown in Figure 7.20. The total capacity of a CD recorded in Mode 1 is 650MB. Modes 0 and 2 can hold 742MB, but cannot be used reliably for data recording.

The track pitch of a CD can be more than 1.6μm when multiple **sessions** are used. Audio CDs have songs recorded in sessions, which, when viewed from below, give the appearance of broad concentric rings. When CDs began to be used for data storage, the idea of a music “recording session” was extended (without modification) to include data recording sessions. There can be as many as 99 sessions on CDs. Sessions are delimited by a 4,500-sector (1-minute) **lead-in** that contains the table of contents for the data contained in the session and by a 6,750- or 2,250-sector **lead-out** (or **runout**) at the end. (The first session on the disk has 6,750 sectors of lead-out. Subsequent sessions have the shorter lead-out.) On CD-ROMs, lead-outs are used to store directory information pertaining to the data contained within the session.

7.7.2 DVD

Digital versatile discs, or **DVD**s (formerly called **digital video discs**), can be thought of as quad-density CDs. DVDs rotate at about three times the speed of CDs. DVD pits are approximately half the size of CD pits (0.4μm to 2.13μm), and the track pitch is 0.74μm. Like CDs, they come in recordable and nonrecordable varieties. Unlike CDs, DVDs can be single sided or double sided, single layer or double layer. Each layer is accessible by refocusing the laser, as shown in Figure 7.22. Single-layer, single-sided DVDs can store 4.78GB, and double-layer, double-sided DVDs can accommodate 17GB of data. The same 2,048-byte DVD sector format supports music, data, and video. With greater data density and improved access time, one can expect that DVDs will eventually replace CDs for long-term data storage and distribution.

DVD improves upon CD in many ways. One of the most important is that DVD uses a 650nm laser whereas CD employs a 780nm laser. This means the feature size can be much smaller on DVD, so the linear space occupied by a single bit is shorter. The shortest pit length on DVD is 0.4μm as opposed to the shortest pit length of 0.83μm on CD. And DVD tracks can be placed much closer together. The track pitch on DVD is 0.74μm, as opposed to 1.6μm for CD. This means the spiral track is longer on DVD. Remember, the track length of a CD—if it could be unwound from its spiral—is about 5 miles (8km). By comparison, if you were to unwind the track of a DVD, it would span about 7.35 miles (11.8km).

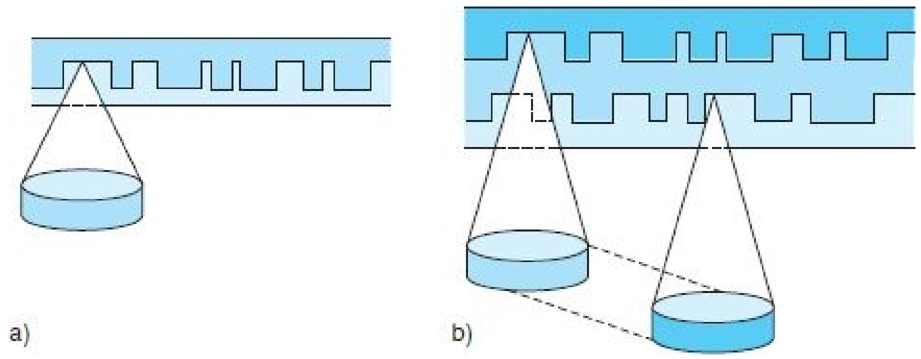


FIGURE 7.22 A Laser Focusing on a) a Single-Layer DVD and b) a Double-Layer DVD One Layer at a Time

A second great improvement is that DVD’s track format is much leaner than CD’s track format. Furthermore, DVD has a much more efficient error-correction algorithm than CD. DVD’s error correction provides better protection using a greatly reduced number of redundant bits over CD.

With its greater data density and improved access times, DVD might be an ideal medium for long-term data storage and retrieval. There are, however, many other media in the running.

7.7.3 Blue-Violet Laser Discs

If DVD’s 650nm laser provides more than twice the recording density of CD’s 750nm laser, then the 405nm wavelength of the blue-violet laser breaks all barriers. Recent advances in laser technology have given us inexpensive blue-violet laser disc drives that can be incorporated in a variety of consumer products. Two incompatible blue-violet disc formats, **Blu-Ray** and **HD-DVD**, fought for market dominance in the mid-2000s. Each brought its own distinct advantage: HD-DVD is backward compatible with traditional DVDs, but Blu-Ray’s storage capacity is greater.

The Blu-Ray Disc format was developed by the Blu-Ray Disc Association, a consortium of nine consumer electronic manufacturers. The group, led by MIT, includes such major companies as Sony, Samsung, and Pioneer. A Blu-Ray disk consists of a 120mm polycarbonate disk with data written in a single spiral track. The minimum pit length on a track is 0.13nm, and the track pitch is 0.32nm. The total recording capacity of a single layer disk is 25GB. Multiple layers can be “stacked” on a disk (up to six as of this writing), although only double-layer disks are available for in-home use. Blu-Ray ended up winning the blue-violet disk format battle because of the dominance of Sony in the movie industry and, above all, the release of Sony's enormously popular PlayStation 3, which used Blu-Ray disks for data storage.

For industrial-grade data storage, both Sony and the Plasmon Corporation released a blue laser medium designed especially for archival data storage. Both products are intended for use in large data centers, and thus are optimized for transfer speed (upwards of 6MB/s with verification). Sony's **Professional Disc for Data** (**PDD**) and Plasmon's second-generation **Ultra Density Optical (UDO-2)** disks can store up to 23GB and 60GB, respectively.

7.7.4 Optical Disk Recording Methods

Various technologies are used to enable recording on CDs and DVDs. The most inexpensive—and most pervasive—method uses heat-sensitive dye. The dye is sandwiched between the polycarbonate substrate and the reflective coating on the CD. When struck by light emitting from the laser, this dye creates a pit in the polycarbonate substrate. This pit affects the optical properties of the reflective layer.

Rewritable optical media, such as CD-RW, replace the dye and reflective coating layers of a CD-R disk with a metallic alloy that includes such exotic elements as indium, tellurium, antimony, and silver. In its unaltered state, this metallic coating is reflective to the laser light. When heated by a laser to about 500°C, it undergoes a molecular change, making it less reflective. (Chemists and physicists call this a **phase change**.) The coating reverts to its original reflective state when heated to only 200°C, thus allowing the data to be changed any number of times. (Industry experts have cautioned that phase-change CD recording may work for "only" 1,000 cycles.)

WORM drives, commonly found on large systems, employ higher-powered lasers than can be reasonably attached to systems intended for individual use. Lower-powered lasers are subsequently used to read the data. The higher-powered lasers permit different—and more durable—recording methods. Three of these methods are:

- **Ablative:** A high-powered laser melts a pit in a reflective metal coating sandwiched between the protective layers of the disk.
- **Bimetallic Alloy:** Two metallic layers are encased between protective coatings on the surfaces of the disk. Laser light fuses the two metallic layers together, causing a reflectance change in the lower metallic layer. Bimetallic Alloy WORM disk manufacturers claim that this medium will maintain its integrity for 100 years.
- **Bubble-Forming:** A single layer of thermally sensitive material is pressed between two plastic layers. When hit by high-powered laser light, bubbles form in the material, causing a reflectance change.

Despite their ability to use the same frame formats as CD-ROM, CD-R and CD-RW disks may not be readable in some CD-ROM drives. The incompatibility arises from the notion that CD-ROMs would be recorded (or pressed) in a single session. CD-Rs and CD-RWs, on the other hand, are most useful when they can be written incrementally like floppy disks. The first CD-ROM specification, ISO 9660, assumed single-session recording and has no provisions for allowing more than 99 sessions on the disk. Cognizant that the restrictions of ISO 9660 were inhibiting wider use of their products, a group of leading CD-R/CD-RW manufacturers formed a consortium to address the problem. The result of their efforts is the **Universal Disk Format Specification**, which allows an unlimited number of recording sessions for each disk. Key to this new format is the idea of replacing the table of contents associated with each session by a floating table of contents. This floating table of contents, called a **virtual allocation table** (**VAT**), is written to the lead-out following the last sector of user data written on the disk. As data is appended to what had been recorded in a previous session, the VAT is rewritten at the end of the new data. This process continues until the VAT reaches the last usable sector on the disk.

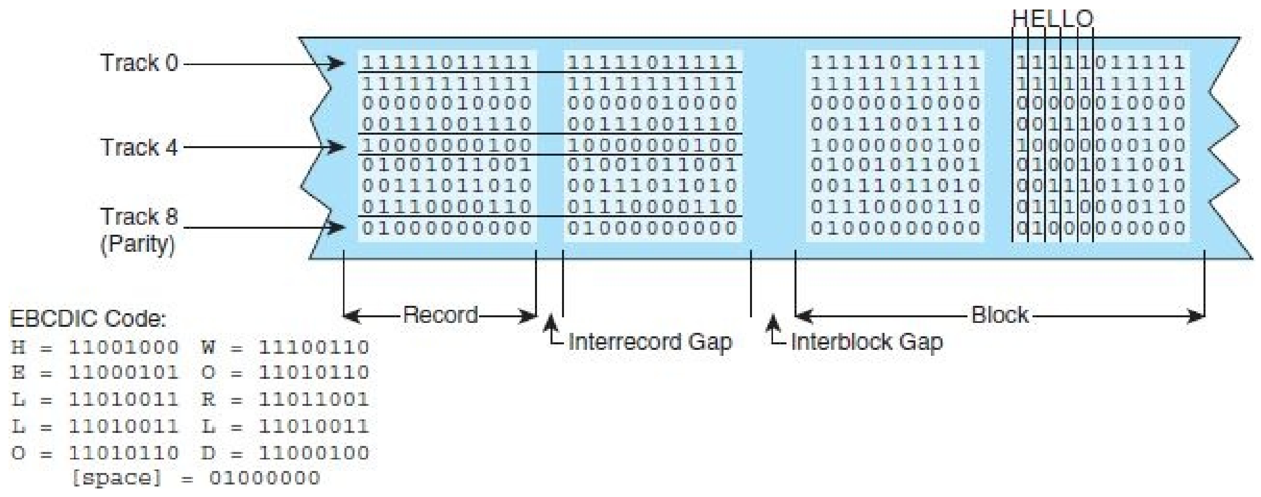
7.8 MAGNETIC TAPE

Magnetic tape is the oldest and most cost-effective of all mass-storage devices. First-generation magnetic tapes were made of the same material used by analog tape recorders. A cellulose-acetate film one-half inch wide (1.25cm) was coated on one side with a magnetic oxide. Twelve hundred feet of this material were wound onto a reel, which then could be hand-threaded on a tape drive. These tape drives were approximately the size of a small refrigerator. Early tapes had capacities under 11MB and required nearly half an hour to read or write the entire reel.

Data was written across the tape one byte at a time, creating one track for each bit. An additional track was added for parity, making the tape nine tracks wide, as shown in Figure 7.23. Nine-track tape used phase modulation coding with odd parity. The parity was odd to ensure that at least one "opposite" flux transition took place during long runs of zeros (nulls), characteristic of database records.

The evolution of tape technology over the years has been remarkable, with manufacturers constantly packing more bytes onto each linear inch of tape. Higher-density tapes are not only more economical to purchase and store, but they also allow backups to be made more quickly. This means that if a system must be taken offline while its files are being copied, downtime is reduced. Further economies can be realized when data is compressed before being written to the tape. (See "Focus on Data Compression" at the end of this chapter.)

The price paid for all of these innovative tape technologies is that a plethora of standards and proprietary techniques have emerged. Cartridges of various sizes and capacities have replaced nine-track open-reel tapes. Thin film coatings similar to those found on digital recording tape have replaced oxide coatings. Tapes support various track densities and employ serpentine or helical scan recording methods.



EBCDIC Code:

H =	11001000	W =	11100110
E =	11000101	O =	11010110
L =	11010011	R =	11011001
L =	11010011	L =	11010011
O =	11010110	D =	11000100
[space] =	01000000		

FIGURE 7.23 A Nine-Track Tape Format

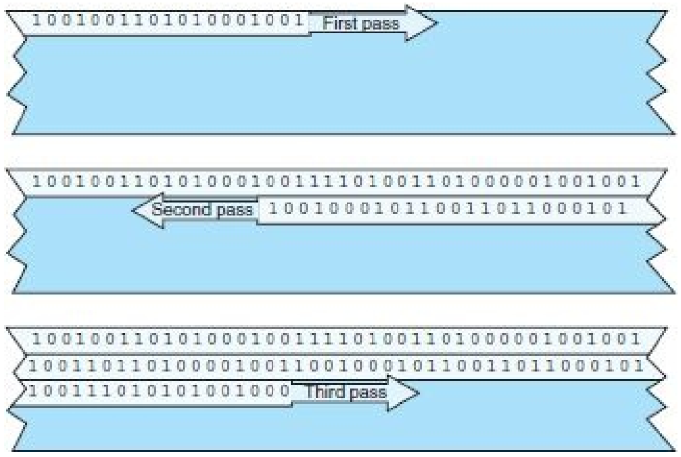


FIGURE 7.24 Three Recording Passes on a Serpentine Tape

Serpentine recording methods place bits on the tape in series. Instead of the bytes being perpendicular to the edges of the tape, as in the nine-track format, they are written "lengthwise," with each byte aligning in parallel with the edge of the tape. A stream of data is written along the length of the tape until the end is reached; then the tape reverses and the next track is written beneath the first one (see Figure 7.24). This process continues until the track capacity of the tape has been reached. **Digital linear tape** (**DLT**) and **Quarter Inch Cartridge** systems use serpentine recording with 50 or more tracks per tape.

Digital audio tape (**DAT**) and 8mm tape systems use **helical scan** recording. In other recording systems, the tape passes straight across a fixed magnetic head in a manner similar to a tape recorder. DAT systems pass tape over a tilted rotating drum (**capstan**), which has two read heads and two write heads, as shown in Figure 7.25. (During write operations, the read heads verify the integrity of the data just after it has been written.) The capstan spins at 2,000rpm in the direction opposite of the motion of the tape. (This configuration is similar to the mechanism used by VCRs.) The two read/write head assemblies write data at 40-degree angles to one another. Data written by the two heads overlaps, thus increasing the recording density. Helical scan systems tend to be slower, and the tapes are subject to more wear than serpentine systems with their simpler tape paths.

LTO: Linear Tape Open

For many years, manufacturers carefully guarded the technology that went into their tape drives. Tapes made for one brand of tape drive could not be read in another. Sometimes even different models of the same brand of tape drive were incompatible. Realizing that this situation was benefiting no one, Hewlett-Packard, IBM, and Seagate Technologies came together in 1997 to formulate an open specification for a best-of-breed tape format called **Linear Tape Open**, or simply **LTO**. In a rare display of collaboration and cooperation among competing vendors, LTO's track format, cartridge design, error-correction algorithm, and compression method incorporated the best ideas presented by each manufacturer. LTO was designed so that it could be refined through a series of "generations," with each generation doubling the capability of the one before it. Generation 5 was released in 2010. These tapes can hold up to 1.4TB with a transfer rate of 280MB per second without compression. Up to 2:1 compression is possible, thus doubling both the capacity and the transfer rate.

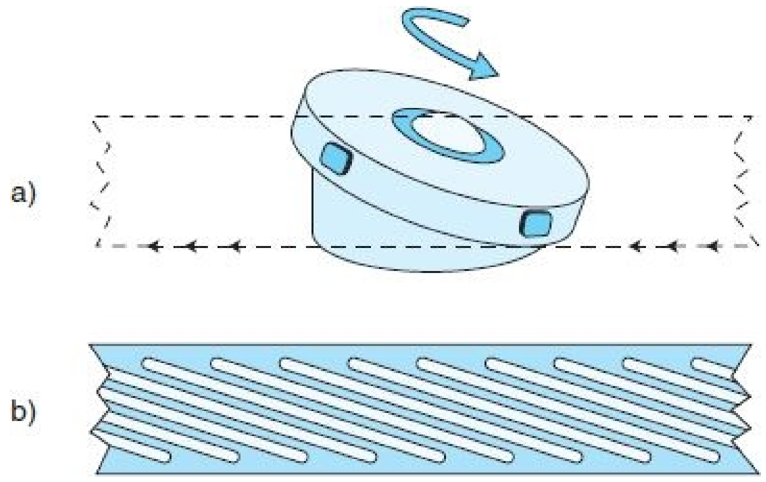


FIGURE 7.25 A Helical Scan Recording
a) The Read/Write Heads on Capstan
b) Pattern of Data Written on the Tape

The reliability and manageability of LTO far surpass all formats that came before it. Deep error-correction algorithms ensure that burst errors as well as single-bit errors are recoverable. The tape cartridge contains memory circuits that store historical information including the number of times the cartridge has been used, the locations and types of errors in the tape, and a table of contents for the data stored on the volume. Like DAT, LTO ensures data readability through simultaneous read/write operations. Errors discovered during this process are noted in cartridge memory and also on the tape itself. The data is then rewritten to a good segment of the tape. With its superb reliability, high data density, and transfer rates, LTO has found wide acceptance by and support from manufacturers and buyers alike.

Tape storage has been a staple of mainframe environments from the beginning. Tapes appear to offer “infinite” storage at bargain prices. They continue to be the primary medium for making file and system backups on large systems. Although the medium itself is inexpensive, cataloging and handling costs can be substantial, especially when the tape library consists of thousands of tape volumes. Recognizing this problem, several vendors have produced a variety of robotic devices that can catalog, fetch, and load tapes in seconds. **Robotic tape libraries**, also known as **tape silos**, can be found in many large data centers. The largest robotic tape library systems have capacities in the hundreds of terabytes and can load a cartridge at user request in less than half a minute.

The Long Future of Tape

Because tape is perceived as “old technology,” some people think that it has no place in the contemporary computing landscape. Moreover, with the cost of some tape cartridges exceeding US $100 each, it’s increasingly easy to argue that disk storage is cheaper than tape in terms of dollars per megabyte. The “obvious” conclusion is that a good deal of money can be saved using disk-to-disk backups instead of disk-to-tape configurations.

Indeed, disk-to-disk backup, in the form of “hot” mirroring, is the only solution for ultra-high-availability configurations. Such configurations consist of a set of backup disk drives that is updated in tandem with an identical set of primary disks. The mirrored disks can even be placed in a secure location miles from the main data center. If disaster strikes the data center, a copy of the important data will survive.

The biggest problem in relying exclusively on disk-to-disk backups is that there is no provision for archival copies of the data. Tape backups generally follow a rotation schedule. Two or three sets of monthly backups are taken and rotated offsite, along with several sets of weekly and daily backups. Each installation determines the rotation schedule based on a number of factors including the importance of the data, how often the data is updated (its **volatility**), and the amount of time required to copy the data to tape. Therefore, the oldest offsite backup may be an image taken months earlier.

Such "ancient" copies of data can rescue a database from human and programming errors. For example, it is possible that a damaging error will be discovered only after a program has been misbehaving for days or weeks. A mirror copy of the database would contain the same erroneous data as the primary set, and it would be no help in repairing the damage. If backups have been managed properly, chances are good that at least some of the data can be recovered from an old backup tape.

Some people complain that it takes too long to write data to tape and that there is no time in which transactional activity can be stopped long enough to copy the data to tape: the **backup window** is insufficient. One can't help but think that if the backup window is insufficient for a disk-to-disk backup, it is probably also insufficient for a tape backup as well. However, tape drives have transfer rates that are competitive with disk transfer rates; when the data is compressed as it is written to tape, tape transfer rates exceed those of disk. If the backup window is too small for either disk or tape backups, then a mirroring approach—with backups taken from the mirror set—should be used. This is known as a **disk-to-disk-to-tape** (**D2D2T**) backup method.

Another consideration is the idea of **information lifecycle management** (**ILM**), which seeks to match the cost of a storage medium with the value of the data that is stored on it. The most important data should be stored on the most accessible and reliable media. Government regulations such as the Sarbanes-Oxley Act of 2002 and Internal Revenue Service codes in the United States require retention of large amounts of data over long periods of time. If there is no compelling business need for instant access to the data, why should it be kept online? ILM practices tell us that at some point the data should be encrypted, removed from primary storage, and placed in a vault. Most corporate installations would be wise to resist shipping a $10,000 disk array offsite for indefinite storage.

For these reasons, tape will continue to be the archival medium of choice for many years to come. Its costs are well justified the moment you retrieve data that would have been long ago overwritten on disk storage.

7.9 RAID

In the 30 years following the introduction of IBM's RAMAC computer, only the largest computers were equipped with disk storage systems. Early disk drives were enormously costly and occupied a large amount of floor space in proportion to their storage capacity. They also required a strictly controlled environment: Too much heat would damage control circuitry, and low humidity caused static buildup that could scramble the magnetic flux polarizations on disk surfaces. Head crashes, or other irrecoverable failures, took an incalculable toll on business, scientific, and academic productivity. A head crash toward the end of the business day meant that all data input had to be redone to the point of the last backup, usually the night before.

Clearly, this situation was unacceptable and promised to grow even worse as everyone became increasingly reliant on electronic data storage. A permanent remedy was a long time coming. After all, weren’t disks as reliable as we could make them? It turns out that making disks more reliable was only part of the solution.

In their 1988 paper, “A Case for Redundant Arrays of Inexpensive Disks,” David Patterson, Garth Gibson, and Randy Katz of the University of California at Berkeley coined the acronym **RAID**. They showed how mainframe disk systems could realize both reliability and performance improvements if they would employ some number of “inexpensive” small disks (such as those used by microcomputers) instead of the **single large expensive disks (SLEDs)** typical of large systems. Because the term *inexpensive* is relative and can be misleading, the proper meaning of the acronym is now generally accepted as Redundant Array of *Independent* Disks.

In their paper, Patterson, Gibson, and Katz defined five types (called **levels**) of RAID, each having different performance and reliability characteristics. These original levels were numbered 1 through 5. Definitions for RAID levels 0 and 6 were later recognized. Various vendors have invented other levels, which may in the future become standards also. These are usually combinations of the generally accepted RAID levels. In this section, we briefly examine each of the seven RAID levels as well as a few hybrid systems that combine different RAID levels to meet particular performance or reliability objectives.

Every vendor of enterprise-class storage systems offers at least one type of RAID implementation. But not all storage systems are automatically protected by RAID. Those systems are often referred to as **just a bunch of disks (JBOD)**.

7.9.1 RAID Level 0

RAID Level 0, or RAID-0, places data blocks in stripes across several disk surfaces so that one record occupies sectors on several disk surfaces, as shown in Figure 7.26. This method is also called **drive spanning**, block interleave data striping, or **disk striping**. (Striping is simply the segmentation of logically sequential data so that segments are written across multiple physical devices. These segments can be as small as a single bit, as in RAID-0, or blocks of a specific size.)

Because it offers no redundancy, of all RAID configurations, RAID-0 offers the best performance, particularly if separate controllers and caches are used for each disk. RAID-0 is also very inexpensive. The problem with RAID-0 lies in the fact that the overall reliability of the system is only a fraction of what would be expected with a single disk. Specifically, if the array consists of five disks, each with a design life of 50,000 hours (about six years), the entire system has an expected design life of 50,000 / 5 = 10,000 hours (about 14 months). As the number of disks increases, the probability of failure increases to the point where it approaches certainty. RAID-0 offers no-fault tolerance because there is no redundancy. Therefore, the only advantage offered by RAID-0 is in performance. Its lack of reliability is downright scary. RAID-0 is recommended for noncritical data (or data that changes infrequently and is backed up regularly) that requires high-speed reads and writes, is low cost, is used in applications such as video or image editing.

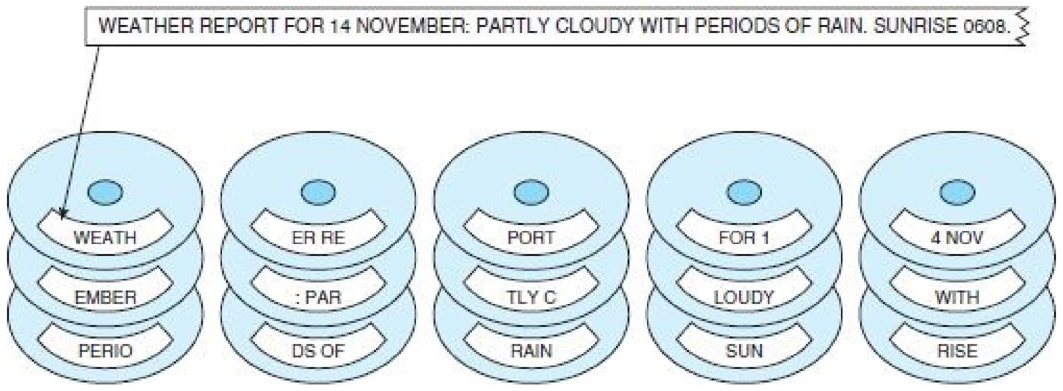

WEATHER REPORT FOR 14 NOVEMBER: PARTLY CLOUDY WITH PERIODS OF RAIN. SUNRISE 0608.
WEATH
EMBER
PERIO
ER RE
: PAR
DS OF
PORT
TLY C
RAIN
FOR 1
LOUDY
SUN
4 NOV
WITH
RISE

FIGURE 7.26 A Record Written Using RAID-0, Block Interleave Data Striping with No Redundancy

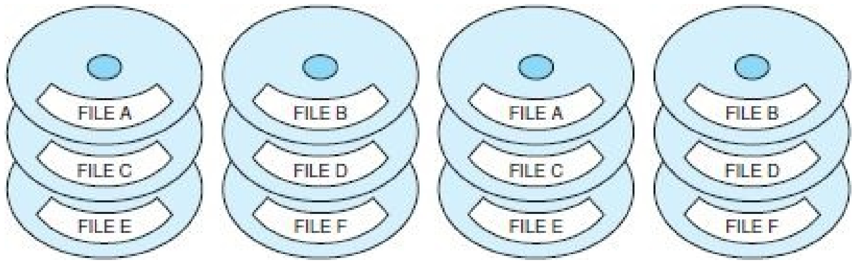


FIGURE 7.27 RAID-1, Disk Mirroring

7.9.2 RAID Level 1

RAID Level 1, or RAID-1 (also known as **disk mirroring**), gives the best failure protection of all RAID schemes. Each time data is written, it is duplicated onto a second set of drives called a **mirror set**, or **shadow set** (as shown in Figure 7.27). This arrangement offers acceptable performance, particularly when the mirror drives are synchronized 180° out of rotation with the primary drives. Although performance on writes is slower than that of RAID-0 (because the data has to be written twice), reads are much faster, because the system can read from the disk arm that happens to be closer to the target sector. This cuts rotational latency in half on reads. RAID-1 is best suited for transaction-oriented, high-availability environments and other applications requiring high-fault tolerance, such as accounting or payroll.

7.9.3 RAID Level 2

The main problem with RAID-1 is that it is costly: You need twice as many disks to store a given amount of data. A better way might be to devote one or more disks to storing information about the data on the other disks. RAID-2 defines one of these methods.

RAID-2 takes the idea of data striping to the extreme. Instead of writing data in blocks of arbitrary size, RAID-2 writes one bit per strip (as shown in Figure 7.28). This requires a minimum of eight surfaces just to accommodate the data. Additional drives are used for error-correction information generated using a Hamming code. The number of Hamming code drives needed to correct singlebit errors is proportionate to the log of the number of data drives to be protected. If any one of the drives in the array fails, the Hamming code words can be used to reconstruct the failed drive. (Obviously, the Hamming drive can be reconstructed using the data drives.)

Because one bit is written per drive, the entire RAID-2 disk set acts as though it were one large data disk. The total amount of available storage is the sum of the storage capacities of the data drives. All of the drives—including the Hamming drives—must be synchronized exactly, otherwise the data becomes scrambled and the Hamming drives do no good. Hamming code generation is time-consuming; thus RAID-2 is too slow for most commercial implementations. In fact, most hard drives today have built-in CRC error correction. RAID-2, however, forms the theoretical bridge between RAID-1 and RAID-3, both of which are used in the real world.

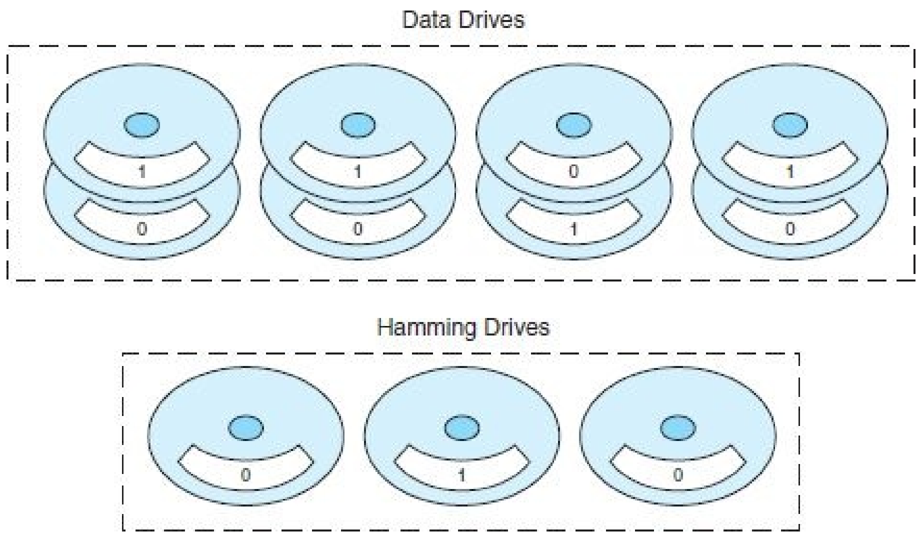


FIGURE 7.28 RAID-2, Bit Interleave Data Striping with a Hamming Code

7.9.4 RAID Level 3

Like RAID-2, RAID-3 stripes (interleaves) data one bit at a time across all of the data drives. Unlike RAID-2, however, RAID-3 uses only one drive to hold a simple parity bit, as shown in Figure 7.29. The parity calculation can be done quickly in hardware using an exclusive OR (XOR) operation on each data bit (shown as b_n) as follows (for even parity):

Parity = b_0 XOR b_1 XOR b_2 XOR b_3 XOR b_4 XOR b_5 XOR b_6 XOR b_7

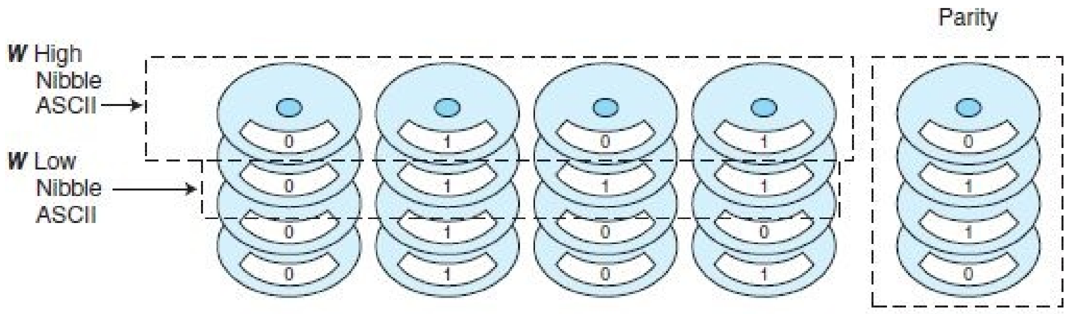
Equivalently,

$$\text{Parity} = (b_0 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7) \bmod 2$$

A failed drive can be reconstructed using the same calculation. For example, assume that drive number 6 fails and is replaced. The data on the other seven data drives and the parity drive are used as follows:

$$b_6 = b_0 \text{ XOR } b_1 \text{ XOR } b_2 \text{ XOR } b_3 \text{ XOR } b_4 \text{ XOR } b_5 \text{ XOR Parity XOR } b_7$$

RAID-3 requires the same duplication and synchronization as RAID-2, but is more economical than either RAID-1 or RAID-2 because it uses only one drive for data protection. RAID-3 has been used in some commercial systems over the years, but it is not well suited for transaction-oriented applications. RAID-3 is most useful for environments where large blocks of data would be read or written, such as with image or video processing.



Letter	ASCII	Parity(even)	
		High Nibble	Low Nibble
W	0101 0111	0	1
E	0100 0101	1	0
A	0100 0001	1	1
T	0101 0100	0	1
H	0100 1000	1	1
E	0100 0101	1	0
R	0101 0010	0	1

FIGURE 7.29 RAID-3: Bit Interleave Data Striping with Parity Disk

7.9.5 RAID Level 4

RAID-4 is another "theoretical" RAID level (like RAID-2). RAID-4 would offer poor performance if it were implemented as Patterson et al. describe. A RAID-4 array, like RAID-3, consists of a group of data disks and a parity disk. Instead of writing data one bit at a time across all of the drives, RAID-4 writes data in strips of uniform size, creating a stripe across all of the drives, as described in RAID-0. Bits in the data strip are XORed with each other to create the parity strip.

You could think of RAID-4 as being RAID-0 with parity. However, adding parity results in a substantial performance penalty caused by contention with the parity disk. For example, suppose we want to write to Strip 3 of a stripe spanning five drives (four data, one parity), as shown in Figure 7.30. First we must read the data currently occupying Strip 3 as well as the parity strip. The old data is XORed with the new data to give the new parity. The data strip is then written along with the updated parity.

Imagine what happens if there are write requests waiting while we are twiddling the bits in the parity block, say one write request for Strip 1 and one for Strip 4. If we were using RAID-0 or RAID-1, both of these pending requests could have been serviced concurrently with the write to Strip 3. Thus, the parity drive becomes a bottleneck, robbing the system of all potential performance gains offered by multiple disk systems.

Some writers have suggested that the performance of RAID-4 can be improved if the size of the stripe is optimized with the record size of the data being written. Again, this might be fine for applications (such as voice or video processing) where the data occupy records of uniform size. However, most database applications involve records of widely varying size, making it impossible to find an "optimum" size for any substantial number of records in the database. Because of its expected poor performance, RAID-4 is not considered suitable for commercial implementations.

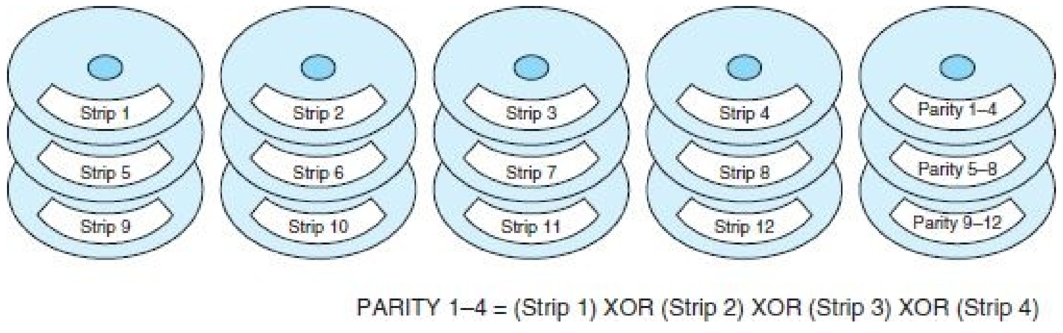


FIGURE 7.30 RAID-4, Block Interleave Data Striping with One Parity Disk

7.9.6 RAID Level 5

Most people agree that RAID-4 would offer adequate protection against single-disk failure. The bottleneck caused by the parity drives, however, makes RAID-4 unsuitable for use in environments that require high-transaction throughput. Certainly, throughput would be better if we could effect some sort of load balancing, writing parity to several disks instead of just one. This is what RAID-5 is all about. RAID-5 is RAID-4 with the parity disks spread throughout the entire array, as shown in Figure 7.31.

Because some requests can be serviced concurrently, RAID-5 provides the best read throughput of all the parity models and gives acceptable throughput on write operations. For example, in Figure 7.31, the array could service a write to drive 4 Strip 6 concurrently with a write to drive 1 Strip 7 because these requests involve different sets of disk arms for both parity and data. However, RAID-5 requires the most complex disk controller of all levels.

Compared with other RAID systems, RAID-5 offers the best protection for the least cost. As such, it has been a commercial success, having the largest installed base of any of the RAID systems. Recommended applications include file and application servers, email and news servers, database servers, and Web servers.

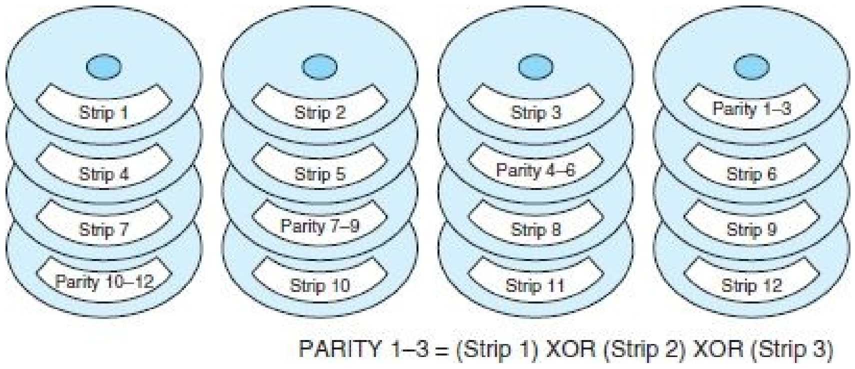


FIGURE 7.31 RAID-5, Block Interleave Data Striping with Distributed Parity

7.9.7 RAID Level 6

Most of the RAID systems just discussed can tolerate at most one disk failure at a time. The trouble is that disk drive failures in large systems tend to come in clusters. There are two reasons for this. First, disk drives manufactured at approximately the same time reach the end of their expected useful lives at approximately the same time. So if you are told that your new disk drives have a useful life of about six years, you can expect problems in year six, possibly concurrent failures.

Second, disk drive failures are often caused by a catastrophic event such as a power surge. A power surge hits all the drives at the same instant, the weakest one failing first, followed closely by the next weakest, and so on. Sequential disk failures like these can extend over days or weeks. If they happen to occur within the Mean Time To Repair (MTTR), including call time and travel time, a second disk could fail before the first one is replaced, thereby rendering the whole array unserviceable and useless.

Systems that require high availability must be able to tolerate more than one concurrent drive failure, particularly if the MTTR is a large number. If an array can be designed to survive the concurrent failure of two drives, we effectively double the MTTR. RAID-1 offers this kind of survivability; in fact, as long as a disk and its mirror aren’t both wiped out, a RAID-1 array could survive the loss of half its disks.

RAID-6 provides an economical answer to the problem of multiple disk failures. It does this by using two sets of error-correction strips for every **rank** (or horizontal row) of drives. A second level of protection is added with the use of Reed-Solomon error-correcting codes in addition to parity. Having two errordetecting strips per stripe does increase storage costs. If unprotected data could be stored on *N* drives, adding the protection of RAID-6 requires *N* + 2 drives. Because of the two-dimensional parity, RAID-6 offers very poor write performance. A RAID-6 configuration is shown in Figure 7.32.

Until recently, there were no commercial deployments of RAID-6. There are two reasons for this. First, there is a sizable overhead penalty involved in generating the Reed-Solomon code. Second, twice as many read/write operations are required to update the error-correcting codes resident on the disk. IBM was first to bring RAID-6 to the marketplace with its RAMAC RVA 2 Turbo disk array. The RVA 2 Turbo array eliminates the write penalty of RAID-6 by keeping running “logs” of disk strips within cache memory on the disk controller. The log data permits the array to handle data one stripe at a time, calculating all parity and error codes for the entire stripe before it is written to the disk. Data is never rewritten to the same stripe it occupied prior to the update. Instead, the formerly occupied stripe is marked as free space, once the updated stripe has been written elsewhere.

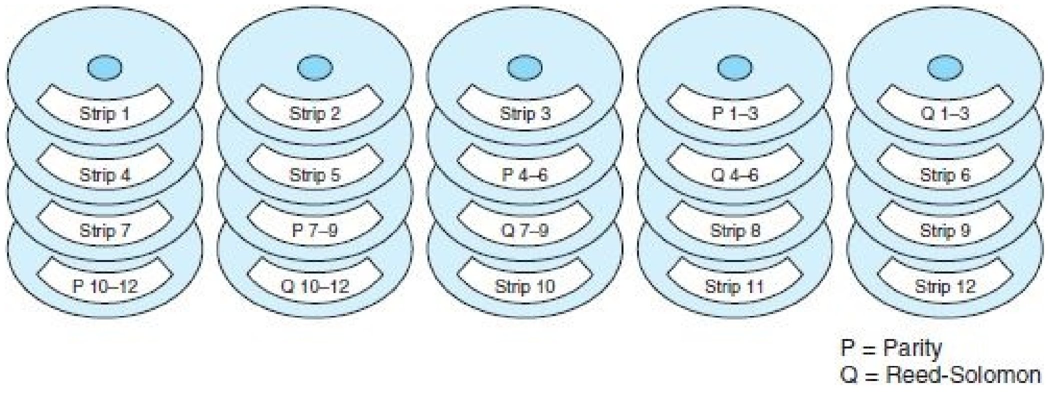
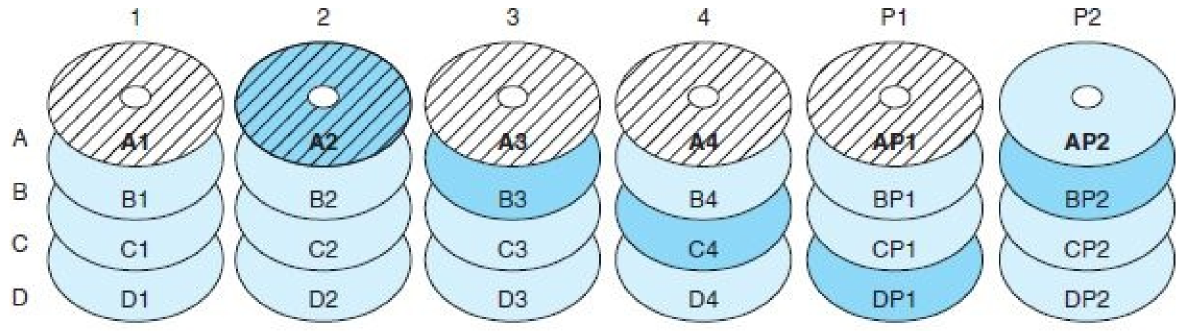


FIGURE 7.32 RAID-6, Block Interleave Data Striping with Dual Error Protection

7.9.8 RAID DP

A relatively new RAID technique employs a pair of parity blocks that protect overlapping sets of data blocks. This method goes by different names depending on the drive manufacturer (there are slight differences among the implementations). The most popular name at this writing seems to be **double parity RAID** (**RAID DP**). Others that crop up in the literature include **EVENODD**, **diagonal parity RAID** (also **RAID DP**), **RAID 5DP**, **advanced data guarding RAID** (**RAID ADG**), and—erroneously!—RAID 6.

The general idea is that any single-disk data block is protected by two linearly independent parity functions. Like RAID-6, RAID DP can tolerate the simultaneous loss of two disk drives without loss of data. In the schematic in Figure 7.33, observe that the contents of each of the RAID surfaces on disk P1 is a function of all the horizontal surfaces to its immediate left. For example, AP1 is a function of A1, A2, A3, and A4. The contents of P2 are functions of diagonal patterns of the surfaces. For example, BP2 is a function of A2, B3, C4, and DP1. Note that AP1 and BP2 overlap on A2. This overlap allows any two drives to be reconstructed by iteratively restoring the overlapped surfaces. This process is illustrated in Figure 7.34.



$$AP1 = A1 \oplus A2 \oplus A3 \oplus A4 \qquad AP2 = A1 \oplus B2 \oplus C3 \oplus D4$$
$$BP1 = B1 \oplus B2 \oplus B3 \oplus B4 \qquad BP2 = A2 \oplus B3 \oplus C4 \oplus DP1$$
$$CP1 = C1 \oplus C2 \oplus C3 \oplus C4 \qquad CP2 = A3 \oplus B4 \oplus CP1 \oplus D1$$
$$DP1 = D1 \oplus D2 \oplus D3 \oplus D4 \qquad DP2 = A4 \oplus BP1 \oplus C1 \oplus D2$$

FIGURE 7.33 Error Recovery Pattern for RAID DP
The recovery of A2 is provided by the overlap of equations AP1 and BP2.

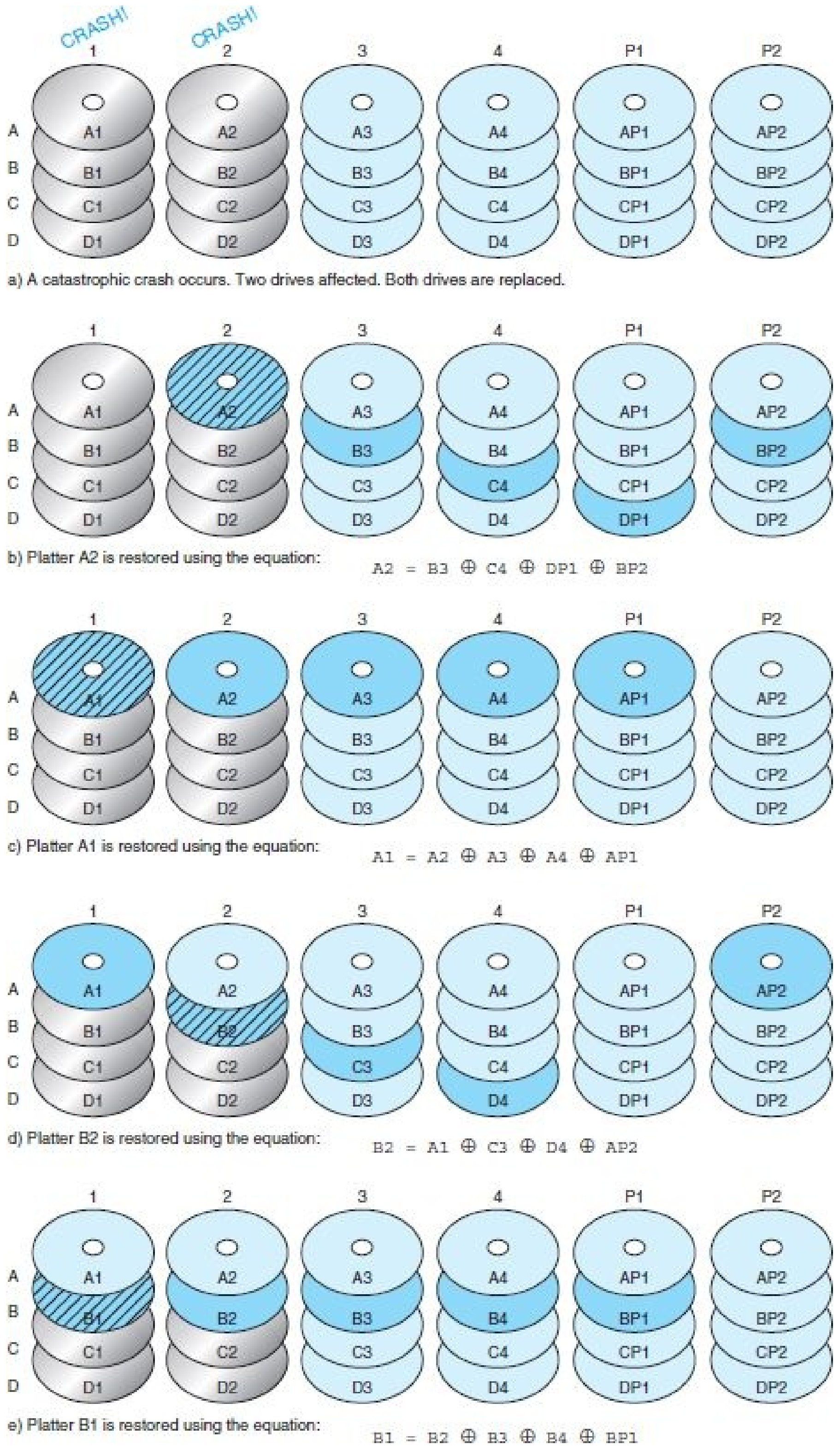

CRASH!
CRASH!
1
2
3
4
P1
P2
A
B
C
D
a) A catastrophic crash occurs. Two drives affected. Both drives are replaced.
b) Platter A2 is restored using the equation:
A2 = B3 ⊕ C4 ⊕ DP1 ⊕ BP2
c) Platter A1 is restored using the equation:
A1 = A2 ⊕ A3 ⊕ A4 ⊕ AP1
d) Platter B2 is restored using the equation:
B2 = A1 ⊕ C3 ⊕ D4 ⊕ AP2
e) Platter B1 is restored using the equation:
B1 = B2 ⊕ B3 ⊕ B4 ⊕ BP1

FIGURE 7.34 Restoring Two Crashed Spindles Using RAID DP

Owing to the dual parity functions, RAID DP can be used over arrays that contain many more physical disks than can be reliably protected using only the simple parity protection of RAID 5. At the choice of the manufacturer, data can be in stripes or blocks. The simple parity function gives much better performance than the Reed-Solomon correction of RAID-6. However, the write performance of RAID DP is still somewhat degraded from RAID 5 because of the need for dual reads and writes, but the trade-off is in having much-improved reliability.

7.9.9 Hybrid RAID Systems

Many large systems are not limited to using only one type of RAID. In some cases, it makes sense to balance high availability with economy. For example, we might want to use RAID-1 to protect the drives that contain our operating system files, whereas RAID-5 is sufficient for data files. RAID-0 would be good enough for “scratch” files used only temporarily during long processing runs and could potentially reduce the execution time of those runs because of the faster disk access.

Sometimes RAID schemes can be combined (or nested) to form a “new” kind of RAID. RAID-10, shown in Figure 7.35a is one such system. It combines the striping of RAID-0 with the mirroring of RAID-1. Although enormously expensive, RAID-10 gives the best possible read performance while providing the best possible availability. Another hybrid level is RAID 0+1, or RAID 01 (not to be confused with RAID 1), which is used for both sharing and replicating data. Like RAID 10, it also combines mirroring and striping, but in a reversed configuration as shown in Figure 7.35b. RAID 01 allows the disk array to continue operating if more than one drive fails in the same mirrored set, and offers substantially improved read and write performance. RAID 50, shown in Figure 7.36, is a combination of striping and distributed parity. This RAID configuration is good in situations that need good fault tolerance with high capacity. RAID levels can be combined in just about any configuration; although nesting is typically limited to two levels, triple-nested RAID configurations are being explored as viable candidates.

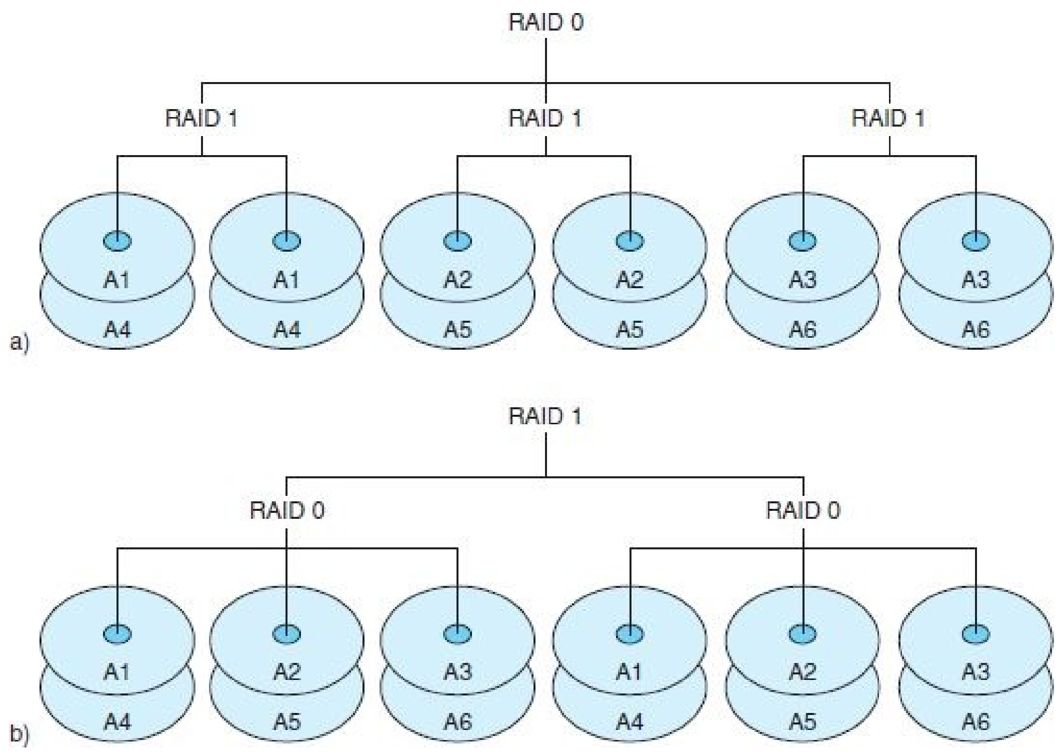


FIGURE 7.35 Hybrid RAID Levels

a) RAID 10, Stripe of Mirrors

b) RAID 01, Mirror of Stripes

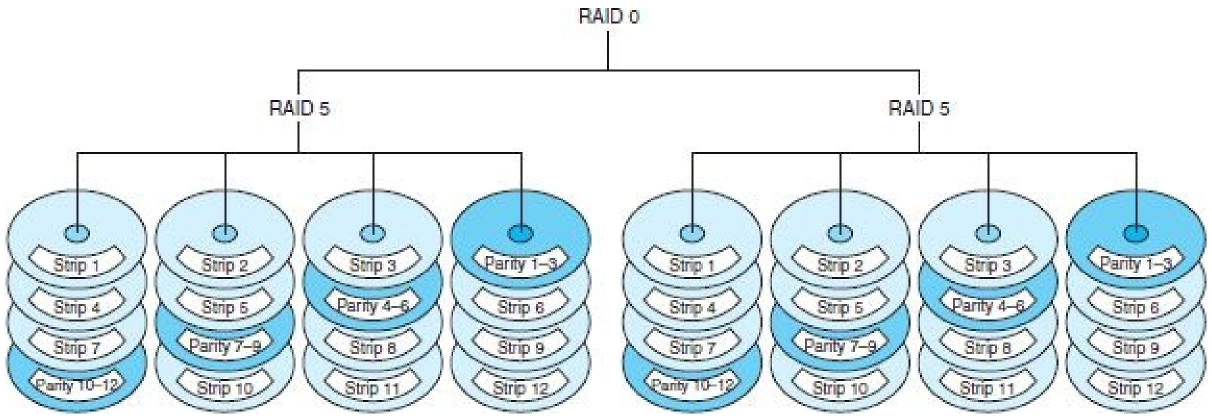


FIGURE 7.36 RAID 50, Striping and Parity

After reading the foregoing sections, it should be clear to you that higher-numbered RAID levels are not necessarily “better” RAID systems. Nevertheless, many people have a natural tendency to think that a higher number of something always indicates something better than a lower number of something. For this reason, various attempts have been made to reorganize and rename the RAID systems that we have just presented. We have chosen to retain the “Berkeley” nomenclature in this book because it is still the most widely recognized. Table 7.1 summarizes the RAID levels just described.

7.10 THE FUTURE OF DATA STORAGE

No one has yet been so bold as to assert a Moore’s Law–like prediction for disk storage. In fact, just the opposite is true: Over the years, experts have periodically pronounced that the limits of disk storage have been reached, only to have their predictions shattered when a manufacturer subsequently announces a product exceeding the latest “theoretical storage limit.” In the 1970s, the density limit for disk storage was thought to be around 2MB/in^2. Today’s disks typically support more than 20GB/in^2. Thus, the “impossible” has been achieved ten thousand times over. These gains have been made possible through advances in several different technologies, including magnetic materials sciences, magneto-optical recording heads, and the invention of more efficient error-correcting codes. But as data densities increase, there’s no getting around the fact that fewer magnetic grains are available within the boundaries of each bit cell. The smallest possible bit cell area is reached when the thermal properties of the disk cause encoded magnetic grains to spontaneously change their polarity, causing a 1 to change to a 0 or a 0 to a 1. This behavior is called **superparamagnetism**, and the bit density at which it occurs is called the **superparamagnetic limit**. At this writing, the superparamagnetic limit is thought to be between 150GB/in^2 and 200GB/in^2. Even if this figure is in error by a few orders of magnitude, it is likely that the greatest increases in magnetic data density have already been achieved. Future exponential gains in data densities will almost certainly be realized by using entirely new storage technologies. With this in mind, research is under way to invent biological, holographic, or mechanical replacements for magnetic disks.

RAID Level	Description	Reliability	Throughput	Pro and Con
0	Block interleave data striping	Worse than single disk	Very good	Least cost, no protection
1	Data mirrored on second identical set	Excellent	Better than single disk on reads, worse on writes	Excellent protection, high cost
2	Bit interleave data striping with Hamming code	Good	Very good	Good performance, high cost, not used in practice
3	Bit interleave data striping with parity disk	Good	Very good	Good performance, reasonable cost
4	Block interleave data striping with one parity disk	Very good	Much worse on writes as single disk, very good on reads	Reasonable cost, poor performance, not used in practice
5	Block interleave data striping with distributed parity	Very good	On writes not as good as single disk, very good on reads	Good performance, reasonable cost
6	Block interleave data striping with dual error protection	Excellent	On writes much worse than single disk, very good on reads	Good performance, reasonable cost, complex to implement
10	Mirrored disk striping	Excellent	Better than single disk on reads, not as good as single disk on writes	Good performance, high cost, excellent protection
50	Parity with striping	Excellent	Excellent read performance. Better than RAID 5; not as good as RAID 10	Good performance, high cost, good protection
DP	Block interleave data striping with dual parity disks	Excellent	Better than single disk on reads, not as good as single disk on writes	Good performance, reasonable cost, excellent protection

TABLE 7.1 Summary of RAID Capabilities

Biological materials can store data in many different ways. Of course, the ultimate data storage is found in DNA, where trillions of different messages can be encoded in a small strand of genetic material. But creating a practical DNA storage device is decades away. Less ambitious approaches combine inorganic magnetic materials (such as iron) with biologically produced materials (such as oils or proteins). Successful prototypes have encouraged the expectation that biological materials will be capable of supporting data densities of 1Tb/in^2. Mass-produced biological storage devices might be brought to market in the second or third decade of the twenty-first century.

A **hologram** is a three-dimensional image rendered by the manipulation of laser beams. Credit cards and some copyrighted CDs and DVDs are emblazoned with iridescent holograms to deter counterfeiting. For at least 50 years, the notion of holographic data storage has ignited the imaginations of fiction writers and computer researchers alike. Thanks to advances in polymer science, holographic storage is finally poised to leap from the pages of pulp magazines and into the data center.

In **holographic data storage**, as shown in Figure 7.37, a laser beam is divided into two separate beams, an object beam and a reference beam. The **object beam** passes through a modulator to produce a coded data pattern. The modulated object beam then intersects with the **reference beam** to produce an interference pattern in the polymer recording medium. Data is recovered from the medium when it is illuminated by the reference beam, thereby reproducing the original coded object beam.

Holographic data storage is exciting for a number of reasons. Foremost is the enormous data density made possible by the use of a three-dimensional medium. Initial experimental systems provide more than 30GB/in^2 with transfer rates of around 1GBps. Holographic data storage is also unique in its ability to provide mass storage that is content addressable. This implies that holographic storage would not necessarily require a directory system as we find on magnetic disks today. All accesses would go directly to where a file is placed, without any need to first consult any file allocation tables.

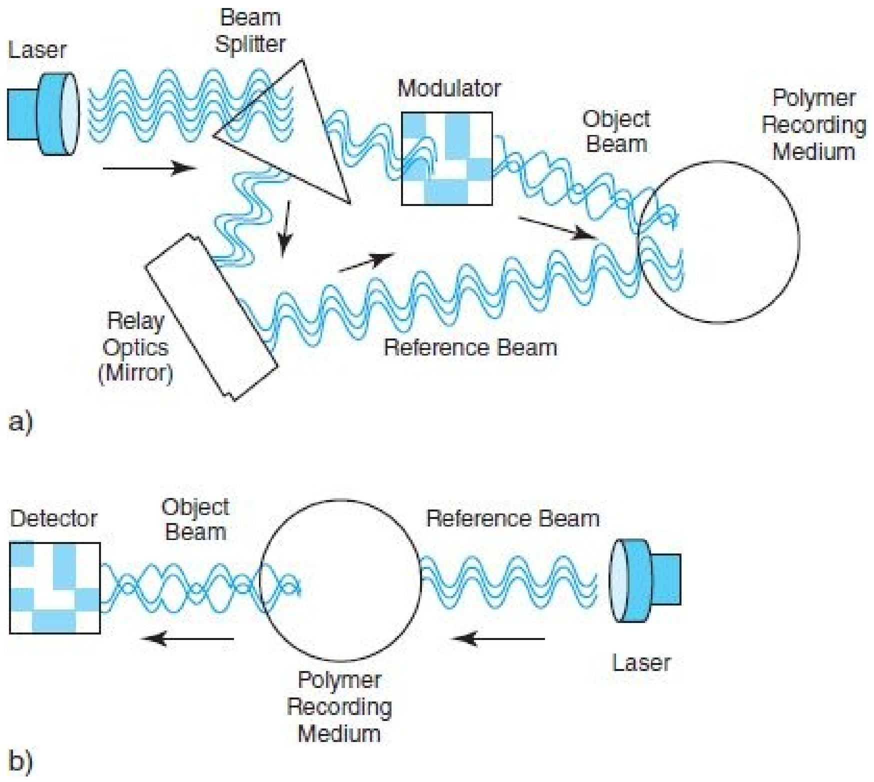


FIGURE 7.37 Holographic Storage
a) Writing Data
b) Reading Data

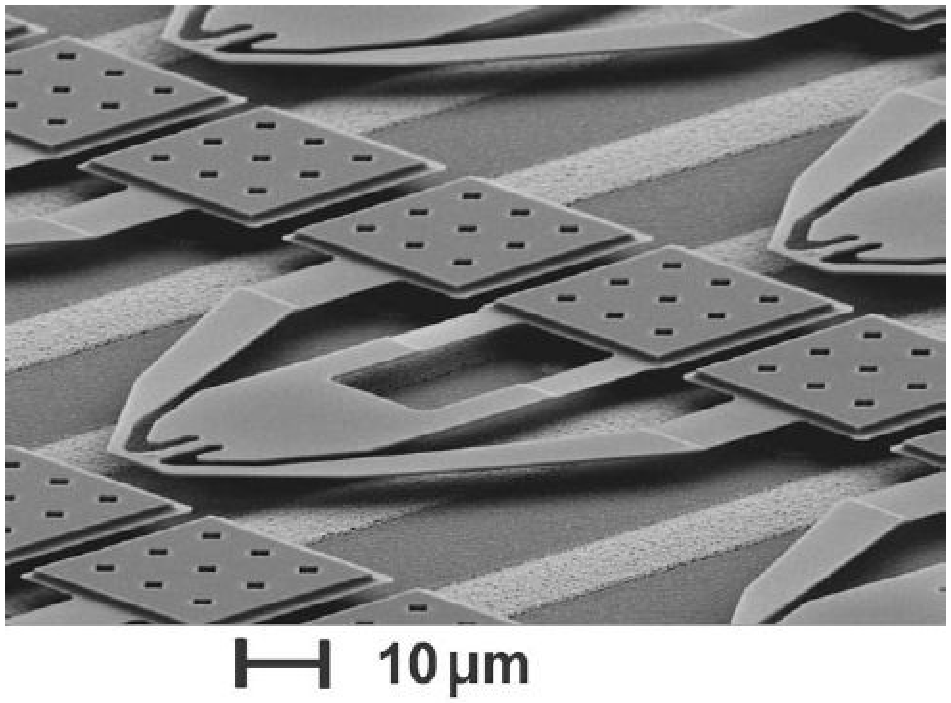


FIGURE 7.38 A Scanning Electron Microscope Image of the Three-Terminal Integrated Cantilevers of IBM's "Millipede" Storage Device The cantilevers are 70 µm long and 75 µm wide. The outer arms of the cantilevers are just 10 µm wide. Reprint Courtesy of International Business Machines Corporation © 2006 International Business Machines Corporation.

The greatest challenge in bringing holographic data storage to commercial production has been in the creation of a suitable polymer medium. Although great progress has been made, inexpensive, rewriteable, stable media still seem to be several years away.

Micro-electro-mechanical (**MEMS**) devices offer another approach to transcending the limits of magnetic storage. One such device is IBM's Millipede. Millipede consists of thousands of microscopic cantilevers that record a binary 1 by pressing a heated microscopic tip into a polymer substrate. The tip reads a binary 1 when it dips into the imprint in the polymer. Laboratory prototypes have achieved densities of more than 100GB/in^2, with 1Tb/in^2 expected as the technology is refined. An electron micrograph of a Millipede cantilever is shown in Figure 7.38.

Even using traditional magnetic disks, enterprise-class storage continues to grow in size and complexity. Terabyte-sized storage systems are now commonplace. It is increasingly likely that the storage problem of the future is not in having sufficient capacity, but in finding the useful information after the data has been stored. This problem may prove to be the most intractable of all.

Carbon nanotubes (CNTs) are among many recent discoveries in the field of nanotechnology. As the name suggests, CNTs are a cylindrical form of elemental carbon where the walls of the cylinders are one atom thick. Carbon nanotubes can be made to act like switches, opening and closing to store bits. Scientists have devised a number of different nanotube memory configurations. Figure 7.39 is a schematic of the configuration that Nantero, Inc., uses in its NRAM product. The nanotube is suspended over a conductor called a gate (Figure 7.39a). This represents the zero state. To set the gate to 1, voltage that is sufficient to attract the nanotube is applied to the gate (Figure 7.39b). The tube stays in place until a release voltage is applied. Thus, the bit cell consumes no power at all until it is read from or written to.

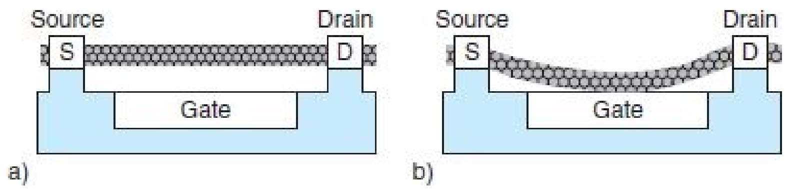


FIGURE 7.39 Carbon Nanotube BIt Storage
a) Set to 0
b) Set to 1

With access times measured in the neighborhood of 3ns, CNTs have been billed as a nonvolatile replacement for volatile RAM as well as a replacement for flash memory. Although not yet in mass production as of this writing, the manufacturability of CNT memories has been demonstrated. It is easy to see that if large-capacity CNT memories can be manufactured economically, they will effectively flatten the storage hierarchy and might all but eliminate the need for at least one level of cache memory in large computer systems.

Like CNTs, **memristor memories** are a type of nonvolatile RAM. Memristors are a rather recently discovered electronic component that combines the properties of a resistor with memory. This is to say, the component's resistance to current flow can be controlled so that states of "high" and "low" can effectively store data bits. These states of high resistance and low resistance are controlled by the application of certain threshold currents that change the physical properties of the underlying semiconductive materials. Like CNTs, memristor memories promise to replace flash memory and flatten the storage hierarchy. This goal can be achieved only if large-capacity devices can be manufactured economically.

Corporations and governments are investing tremendous amounts of research money to bring new storage technologies to market. There seems no end to our thirst for data—accessible data—with which we can infer all kinds of trends and predictions of human behavior, known as **big data**. Big data, however, is becoming increasingly expensive as terabytes of disk storage spin 24 hours a day, consuming gigawatts of electrical power in the process. Even with billions of investment dollars flowing into these new technologies, the payoff is expected to be even greater—by orders of magnitude.

CHAPTER SUMMARY

This chapter has given you a broad overview of many aspects of computer input/output and storage systems. You have learned that different classes of machines require different I/O architectures. Large systems store and access data in ways that are fundamentally different from the methods used by smaller computers. For the very smallest computers—such as embedded processors—programmed I/O is most suitable. It is flexible, but it doesn't offer good performance in general-purpose systems. For single-user systems, interrupt-driven I/O is the best choice, particularly when multitasking is involved. Single-user and medium-sized systems usually employ DMA I/O, in which the CPU offloads I/O to the DMA circuits. Channel I/O is best for high-performance systems. It allocates separate high-capacity paths to facilitate large data transfers.

I/O can be processed character by character or in blocks. Character I/O is best for serial data transfers. Block I/O can be used in serial or parallel data transmission. The original articles that describe IBM's RAMAC system can be found in Lesser & Haanstra (2000) and Noyes & Dickinson (2000).

You have seen how data is stored on a variety of media, including magnetic tape and disk and optical media. Your understanding of magnetic disk operations will be particularly useful to you if you are ever in a position to analyze disk performance in the context of programming, system design, or problem diagnosis.

Our discussion of RAID systems should help you to understand how RAID can provide both improved performance and increased availability for systems upon which we all depend. The most important RAID implementations are given in Table 7.1.

We hope that throughout our discussions, you have gained an appreciation for the trade-offs that are involved with virtually every system decision. You have seen how we must often make choices between "better" and "faster," and "faster" and "cheaper," in so many of the areas that we have just studied. As you assume leadership in systems projects, you must be certain that your customers understand these trade-offs as well. Often you need the tact of a diplomat to thoroughly convince your clients that there is no such thing as a free lunch.

FURTHER READING

You can learn more about Amdahl's Law by reading his original paper (Amdahl, 1967). Hennessy and Patterson (2011) provide additional coverage of Amdahl's Law. Gustavson's tutorial (1984) on computer buses is well worth reading.

Rosch (1997) contains a wealth of detail relevant to many of the topics described in this chapter, although it focuses primarily on small computer systems. It is well organized, and its style is clear and readable. Anderson's article (2003) takes a slightly different point of view of topics discussed in this chapter.

Rosch (1997) also presents a good overview of CD storage technology. More complete coverage, including CD-ROM's physical, mathematical, and electrical engineering underpinnings, can be found in Stan (1998) and Williams (1994).

Patterson, Gibson, and Katz (1988) provide the foundation paper for the RAID architecture. RAID DP is nicely explained in papers by Blaum et al. (1994) and Corbett et al. (2004).

The IBM Corporation hosts what is by far the best website for detailed technical information. IBM stands alone in making prodigious quantities of excellent documentation available for all seekers. Its home page can be found at www.ibm.com. IBM also has a number of sites devoted to specific areas of interest, including storage systems (www.storage.ibm.com), in addition to its server product lines (www.ibm.com/eservers). IBM's research and development pages contain the latest information relevant to emerging technologies (www.research.ibm.com). High-quality scholarly research journals can be found through this site at www.research.ibm.com/journal. Jaquette's LTO article (2003) explains this topic well.

Holographic storage has been discussed with varying intensity over the years. Two recent articles are Ashley et al. (2000) and Orlov (2000). Visitors to IBM's Zurich Research Laboratory website (www.zurich.ibm.com/st) will be rewarded with fascinating photographs and detailed descriptions of the MEMS storage system. Two good articles on the same subject are Carley, Ganger, & Nagle (2000) and Vettiger (2000).

Michael Cornwell's (2012) article on solid state drives gives the reader some good general information. The websites of various manufacturers are loaded with technical documentation regarding these devices. SanDisk (http://www.sandisk.com), Intel (http://www.intel.com), and Hewlett-Packard (http://www.hp.com/) are three such sites.

The article by Kryder and Kim (2009) is a look forward into some of the storage technologies that were discussed in this chapter as well as some interesting ones that were not. More information about memristor storage can be found in papers by Anthes (2011) and Ohta (2011). You can explore the amazing world of CNTs in the papers by Bichoutskaia et al. (2008), Zhou et al. (2007), and Paulson (2004). We encourage you to keep alert for breaking news in the areas of memristor and CNT storage. Both of these technologies are well positioned to make the spinning disk obsolete.

REFERENCES

Amdahl, G. M. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities." *Proceedings of AFIPS 1967 Spring Joint Computer Conference 30*, April 1967, Atlantic City, NJ, pp. 483–485.

Anderson, D. "You Don't Know Jack about Disks." *ACM Queue*, June 2003, pp. 20–30.

Anthes, G. "Memristors: Pass or Fail?" *Communications of the ACM 54*:3, March 2011, pp. 22–23.

Ashley, J., et al. "Holographic Data Storage." *IBM Journal of Research and Development 44*:3, May 2000, pp. 341–368.

Bichoutskaia, E., Popov, A. M., & Lozovik, Y. E. "Nanotube-Based Data Storage Devices." *Materials Today 11*:6, June 2008, pp. 38–43.

Blaum, M., Brady, J., Bruck, J., & Menon, J. "EVENODD: An Optimal Scheme for Tolerating Double Disk Failures in RAID Architectures." *ACM SIGARCH Computer Architecture News 22*:2, April 1994, pp. 245–254.

Carley, R. L., Ganger, G. R., & Nagle, D. F. "MEMS-Based Integrated-Circuit Mass-Storage Systems." *Communications of the ACM 43*:11, November 2000, pp. 72–80.

Corbett, P., et al. "Row-Diagonal Parity for Double Disk Failure Correction." *Proceedings of the Third USENIX Conference on File and Storage Technologies*. San Francisco, CA, 2004.

Cornwell, M. "Anatomy of a Solid-State Drive." *CACM 55*:12, December 2012, pp. 59–63.

Gustavson, D. B. "Computer Buses—A Tutorial." *IEEE Micro*, August 1984, pp. 7–22.

Hennessy, J. L., & Patterson, D. A. *Computer Architecture: A Quantitative Approach*, 5th ed. San Francisco, CA: Morgan Kaufmann Publishers, 2011.

Jaquette, G. A. "LTO: A Better Format for Midrange Tape." *IBM Journal of Research and Development 47*:4, July 2003, pp. 429–443.

Kryder, M. H. & Kim, C. S. "After Hard Drives—What Comes Next?" *IEEE Transactions on Magnetics 45*:10, October 2009, pp. 3406–3413.

Lesser, M. L., & Haanstra, J. W. "The Random Access Memory Accounting Machine: I. System Organization of the IBM 305." *IBM Journal of Research and Development 1*:1, January 1957. Reprinted in Vol. 44, No. 1/2, January/March 2000, pp. 6–15.

Noyes, T., & Dickinson, W. E. "The Random Access Memory Accounting Machine: II. System Organization of the IBM 305." *IBM Journal of Research and Development 1*:1, January 1957. Reprinted in Vol. 44, No. 1/2, January/March 2000, pp. 16–19.

Ohta, T. "Phase Change Memory and Breakthrough Technologies." *IEEE Transactions on Magnetics 47*:3, March 2011, pp. 613–619.

Orlov, S. S. "Volume Holographic Data Storage." *Communications of the ACM 43*:11, November 2000, pp. 47–54.

Patterson, D. A., Gibson, G., & Katz, R. "A Case for Redundant Arrays of Inexpensive Disks (RAID)." *Proceedings of the ACM SIGMOD Conference on Management of Data*, June 1988, pp. 109–116.

Paulson, L. D. "Companies Develop Nanotech RAM Chips." *IEEE Computer*. August 2004, p. 28.

Rosch, W. L. *The Winn L. Rosch Hardware Bible*. Indianapolis: Sams Publishing, 1997.

Stan, S. G. *The CD-ROM Drive: A Brief System Description*. Boston: Kluwer Academic Publishers, 1998.

Vettiger, P., et al. "The 'Millipede'—More than One Thousand Tips for Future AFM Data Storage." *IBM Journal of Research and Development 44*:3, May 2000, pp. 323–340.

Williams, E. W. *The CD-ROM and Optical Recording Systems*. New York: Oxford University Press, 1994.

Zhou, C., Kumar, A., & Ryu, K. "Small Wonder." *IEEE Nanotechnology Magazine 1*:1, September 2007, pp. 13, 17.

REVIEW OF ESSENTIAL TERMS AND CONCEPTS

1. State Amdahl's Law in words.
2. What is speedup?
3. What is a protocol, and why is it important in I/O bus technology?
4. Name three types of durable storage.
5. Explain how programmed I/O is different from interrupt-driven I/O.
6. What is polling?
7. How are address vectors used in interrupt-driven I/O?
8. How does direct memory access (DMA) work?

9. What is a bus master?

10. Why does DMA require cycle stealing?

11. What does it mean when someone refers to I/O as bursty?

12. How is channel I/O different from interrupt-driven I/O?

13. How is channel I/O similar to DMA?

14. What is multiplexing?

15. What distinguishes an asynchronous bus from a synchronous bus?

16. What is settle time, and what can be done about it?

17. Why are magnetic disks called direct access devices?

18. Explain the relationship among disk platters, tracks, sectors, and clusters.

19. What are the major physical components of a rigid disk drive?

20. What is zoned-bit recording?

21. What is seek time?

22. What is the sum of rotational delay and seek time called?

23. Explain the differences between an SSD and a magnetic disk.

24. By how much is an SSD faster than a magnetic disk?

25. What is short stroking, and how does it affect the relative cost per gigabyte of SSDs?

26. How do enterprise SSDs differ from SSDs intended for laptop computers?

27. What is wear leveling, and why is it needed for SSDs?

28. What is the name for robotic optical disk library devices?

29. What is the acronym for computer output that is written directly to optical media rather than paper or microfiche?

30. Magnetic disks store bytes by changing the polarity of a magnetic medium. How do optical disks store bytes?

31. How is the format of a CD that stores music different from the format of a CD that stores data? How are the formats alike?

32. Why are CDs especially useful for long-term data storage?

33. Do CDs that store data use recording sessions?

34. How do DVDs store so much more data than regular CDs?

35. Explain why Blu-Ray discs hold so much more data than regular DVDs.

36. Name the three methods for recording WORM disks.

37. Why is magnetic tape a popular storage medium?

38. Explain how serpentine recording differs from helical scan recording.

39. What are two popular tape formats that use serpentine recording?

40. Which RAID levels offer the best performance?

41. Which RAID levels offer the best economy while providing adequate redundancy?

42. Which RAID level uses a mirror (shadow) set?

43. What are hybrid RAID systems?

44. What is the significance of the superparamagnetic limit?

45. What does the superparamagnetic limit mean for disk drives?

46. Explain how holographic storage works.

47. What is the general idea behind MEMS storage?

48. How does CNT storage work?

49. What is a memristor, and how does it store data?

EXERCISES

◆ **1.** Calculate the overall speedup of a system that spends 65% of its time on I/O with a disk upgrade that provides for 50% greater throughput.

2. Calculate the overall speedup of a system that spends 40% of its time in calculations with a processor upgrade that provides for 100% greater throughput.

3. Suppose your company has decided that it needs to make certain busy servers 50% faster. Processes in the workload spend 60% of their time using the CPU and 40% on I/O. In order to achieve an overall system speedup of 25%:

a) How much faster does the CPU need to be?

b) How much faster does the disk need to be?

4. Suppose your company has decided that it needs to make certain busy servers 30% faster. Processes in the workload spend 70% of their time using the CPU and 30% on I/O. In order to achieve an overall system speedup of 30%:

a) How much faster does the CPU need to be?

b) How much faster does the disk need to be?

5. Suppose that you are designing a game system that responds to players' pressing buttons and toggling joysticks. The prototype system is failing to react in time to these input events, causing noticeable annoyance to the gamers. You have calculated that you need to improve overall system performance by 50%. This is to say that the entire system needs to be 50% faster than it is now. You know that these I/O events account for 75% of the system workload. You figure that a new I/O interface card should do the trick. If the system's existing I/O card runs at 10kHz (pulses per second), what is the speed of the I/O card that you need to order from the supplier?

6. Suppose that you are designing an electronic musical instrument. The prototype system occasionally produces off-key notes, causing listeners to wince and grimace. You have determined that the cause of the problem is that the system becomes overwhelmed in processing the complicated input. You are thinking that if you could boost overall system performance by 12% (making it 12% faster than it is now), you could eliminate the problem. One option is to use a faster processor. If the processor accounts for 25% of the workload of this system, and you need to boost performance by 12%, how much faster does the new processor need to be?

7. Your friend has just bought a new personal computer. She tells you that her new system runs at 1GHz, which makes it more than three times faster than her old 300MHz system. What would you tell her? (Hint: Consider how Amdahl's Law applies.)

8. Suppose the daytime processing load consists of 60% CPU activity and 40% disk activity. Your customers are complaining that the system is slow. After doing some research, you learn that you can upgrade your disks for $8,000 to make them 2.5 times as fast as they are currently. You have also learned that you can upgrade your CPU to make it 1.4 times as fast for $5,000.

   a) Which would you choose to yield the best performance improvement for the least amount of money?

   b) Which option would you choose if you don't care about the money, but want a faster system?

   c) What is the break-even point for the upgrades? That is, what price would we need to charge for the CPU (or the disk—change only one) so the result was the same cost per 1% increase for both?

◆ **9.** How would you answer exercise 8 if the system activity consists of 55% processor time and 45% disk activity?

10. Amdahl's Law is as applicable to software as it is to hardware. An oft-cited programming truism states that a program spends 90% of its time executing 10% of its code. Thus, tuning a small amount of program code can often have an enormous effect on the overall performance of a software product. Determine the overall system speedup if:

    a) 90% of a program is made to run 10 times as fast (900% faster).

    b) 80% of a program is made to run 20% faster.

11. Name the four types of I/O architectures. Where are each of these typically used, and why are they used there?

◆ **12.** A CPU with interrupt-driven I/O is busy servicing a disk request. While the CPU is midway through the disk-service routine, another I/O interrupt occurs.

    a) What happens next?

    b) Is it a problem?

    c) If not, why not? If so, what can be done about it?

13. A generic DMA controller consists of the following components:

    - Address generator
    - Address bus interface
    - Data bus interface
    - Bus requestor
    - Interrupt signal circuits
    - Local peripheral controller

    The local peripheral controller is the circuitry that the DMA uses to select among the peripherals connected to it. This circuit is activated right after the bus is requested. What is the purpose of each of the other components listed above, and when are they active? (Use Figure 7.6 as a guide.)

14. Of programmed I/O, interrupt-driven I/O, DMA, or channel I/O, which is most suitable for processing the I/O of a:

a) Mouse
b) Game controller
c) CD
d) Thumb drive or memory stick

Explain your answers.

15. Why are I/O buses provided with clock signals?

16. If an address bus needs to be able to address eight devices, how many conductors will be required? What if each of those devices also needs to be able to talk back to the I/O control device?

17. The protocol for a certain data bus is shown in the table below. Draw the corresponding timing diagram. You may refer to Figure 7.11.

Time	**Salient Bus Signal**	**Meaning**
t_0	Assert Read	Bus is needed for reading (not writing)
t_1	Assert Address	Indicates where bytes will be written
t_2	Assert Request	Request read to address on address lines
t_3–t_7	Data Lines	Read data (requires several cycles)
t_4	Assert Ready	Acknowledges read request, bytes placed on data lines
t_4	Lower Request	Request signal no longer needed
t_8	Lower Ready	Release bus

18. With regard to Figure 7.11 and exercise 17, we have not provided for any type of error handling, such as if the address on the address lines were invalid, or the memory couldn't be read because of a hardware error. What could we do with our bus model to provide for such events?

19. We pointed out that I/O buses do not need separate address lines. Construct a timing diagram similar to Figure 7.11 that describes the handshake between an I/O controller and a disk controller for a write operation. (Hint: You will need to add a control signal.)

***20.** If each interval shown in Figure 7.11 is 50ns, how long would it take to transfer 10 bytes of data? Devise a bus protocol, using as many control lines as you need, that would reduce the time required for this transfer to take place. What happens if the address lines are eliminated and the data bus is used for addressing instead? (Hint: An additional control line may be needed.)

21. Define the terms *seek time*, *rotational delay*, and *transfer time*. Explain their relationship.

◆ **22.** Why do you think the term *random access device* is something of a misnomer for disk drives?

23. Why do differing systems place disk directories in different track locations on the disk? What are the advantages of using each location that you cited?

◆ **24.** Verify the average latency rate cited in the disk specification of Figure 7.15. Why is the calculation divided by 2?

25. By inspection of the disk specification in Figure 7.15, what can you say about whether the disk drive uses zoned-bit recording?

26. The disk specification in Figure 7.15 gives a data transfer rate of 60MB per second when reading from the disk, and 320MB per second when writing to the disk. Why are these numbers different? (Hint: Think about buffering.)

27. Do you trust disk drive MTTF figures? Explain.

◆ **28.** Suppose a disk drive has the following characteristics:

- 4 surfaces
- 1,024 tracks per surface
- 128 sectors per track
- 512 bytes/sector
- Track-to-track seek time of 5ms
- Rotational speed of 5,000 rpm

a) What is the capacity of the drive?

b) What is the access time?

29. Suppose a disk drive has the following characteristics:

- 5 surfaces
- 1,024 tracks per surface
- 256 sectors per track
- 512 bytes/sector
- Track-to-track seek time of 8ms
- Rotational speed of 7,500 rpm

a) What is the capacity of the drive?

b) What is the access time?

c) Is this disk faster than the one described in exercise 28? Explain.

30. Suppose a disk drive has the following characteristics:

- 6 surfaces
- 16,383 tracks per surface
- 63 sectors per track
- 512 bytes/sector
- Track-to-track seek time of 8.5ms
- Rotational speed of 7,200 rpm

a) What is the capacity of the drive?

b) What is the access time?

31. Suppose a disk drive has the following characteristics:

- 6 surfaces
- 953 tracks per surface
- 256 sectors per track
- 512 bytes/sector
- Track-to-track seek time of 6.5ms
- Rotational speed of 5,400 rpm

a) What is the capacity of the drive?

b) What is the access time?

c) Is this disk faster than the one described in exercise 30? Explain.

32. Transfer rate of a disk drive can be no faster than the bit density (bits/track) times the rotational speed of the disk. Figure 7.15 gives a data transfer rate of 112GB/sec. Assume that the average track length of the disk is 5.5 inches. What is the average bit density of the disk?

33. What are the advantages and disadvantages of having a small number of sectors per disk cluster? (Hint: You may want to think about retrieval time and the required lifetime of the archives.)

34. How does the organization of an optical disk differ from the organization of a magnetic disk?

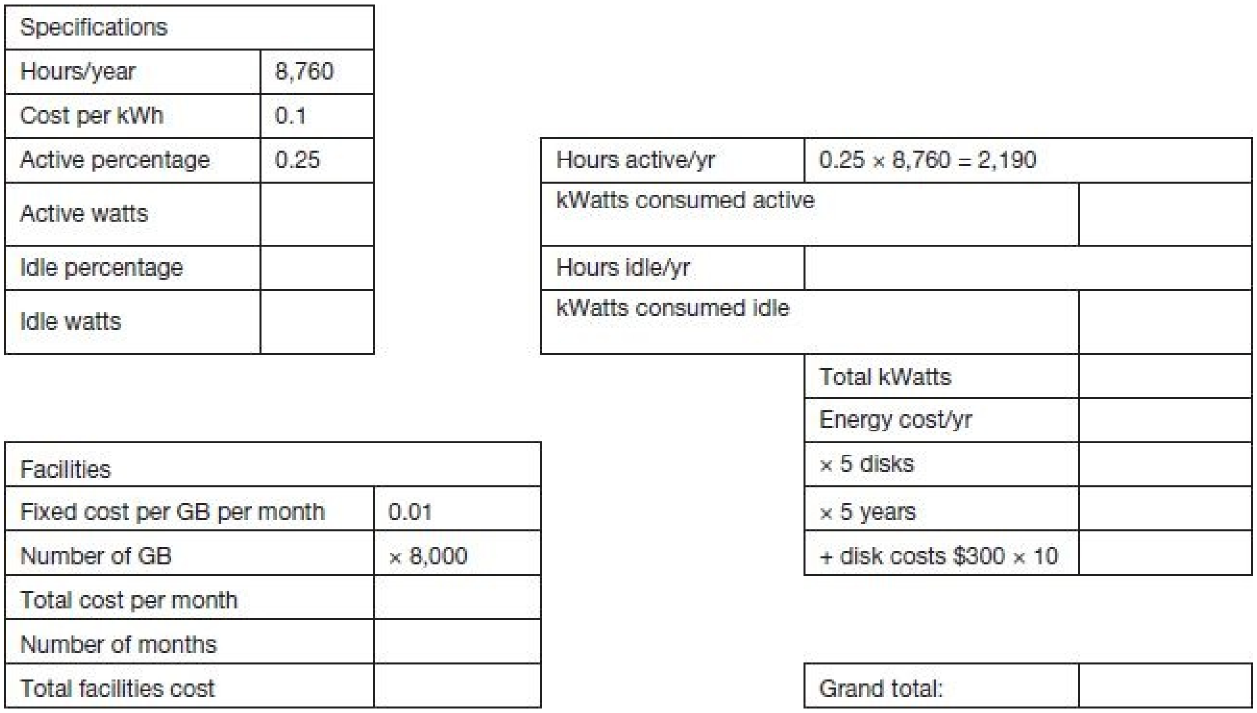
35. How does the organization of an SSD differ from a magnetic disk? How are they similar to a disk?

36. In Section 7.6.2, we said that magnetic disks are power hungry as compared to main memory. Why do you think this is the case?

37. Explain wear leveling and why it is needed for SSDs. We said that wear-leveling is important for the continual updating of virtual memory pagefiles. What problem does wear-leveling aggravate for pagefiles?

38. Compare the disk specifications for the HDD and SSD in Figures 7.15 and 7.17, respectively. Which items are the same? Why? Which items are different? Why?

39. If 800GB server-grade HDDs cost $300, electricity costs $0.10 per kilowatt hour, and facilities cost $0.01 per GB per month, use the disk specification in Figure 7.15 to determine how much it costs to store 8TB of data online for 5 years. Assume that the HDD is active 25% of the time. What can be done to reduce this cost? Hint: Use the “Read/Write” and “Idle” power requirements in Figure 7.15. Use the tables below as a guide.



Specifications	
Hours/year	8,760
Cost per kWh	0.1
Active percentage	0.25
Active watts	
Idle percentage	
Idle watts	

Hours active/yr	0.25 × 8,760 = 2,190
kWatts consumed active	
Hours idle/yr	
kWatts consumed idle	
Total kWatts	
Energy cost/yr	
× 5 disks	
× 5 years	
+ disk costs $300 × 10	

Facilities	
Fixed cost per GB per month	0.01
Number of GB	× 8,000
Total cost per month	
Number of months	
Total facilities cost	

Grand total:	

40. The disk drives connected to the servers in your company’s server farm are nearing the end of their useful lives. Management is considering replacing 8TB of disk capacity with SSDs. Someone is making the argument that the difference in the cost between the SSDs and traditional magnetic disks will be offset by the cost of electricity saved by the SSDs. The 800GB SSDs cost $900. The 800GB server-grade HDDs cost $300. Use the disk specifications in Figures 7.15 and 7.17 to confirm or refute this claim. Assume that both the HDD and the SSD are active 25% of the time and that the cost of electricity is $0.10 per kilowatt hour. Hint: Use the "Read/Write" and "Idle" power requirements in Figure 7.15.

41. A company that has engaged in a business that requires fast response times has just received a bid for a new system that includes much more storage than was specified in the requirements document. When the company questioned the vendor about the increased storage, the vendor said he was bidding a set of the smallest-capacity disk drives that the company makes. Why didn't the vendor just bid fewer disks?

42. Discuss the difference between how DLT and DAT record data. Why would you say that one is better than the other?

43. How would the error-correction requirements of an optical document storage system differ from the error-correction requirements of the same information stored in textual form? What are the advantages offered by having different levels of error correction for optical storage devices?

44. You have a need to archive a large amount of data. You are trying to decide whether to use tape or optical storage methods. What are the characteristics of this data and how it is used that will influence your decision?

45. Discuss the pros and cons of using disk versus tape for backups.

46. Suppose you have a 100GB database housed on a disk array that supports a transfer rate of 60MBps and a tape drive that supports 200GB cartridges with a transfer rate of 80MBps. How long will it take to back up the database? What is the transfer time if 2:1 compression is possible?

***47.** A particular high-performance computer system has been functioning as an e-business server on the Web. This system supports $10,000 per hour in gross business volume. It has been estimated that the net profit per hour is $1,200. In other words, if the system goes down, the company will lose $1,200 every hour until repairs are made. Furthermore, any data on the damaged disk would be lost. Some of this data could be retrieved from the previous night's backups, but the rest would be gone forever. Conceivably, a poorly timed disk crash could cost your company hundreds of thousands of dollars in immediate revenue loss, and untold thousands in permanent business loss. The fact that this system is not using any type of RAID is disturbing to you.

Although your chief concern is data integrity and system availability, others in your group are obsessed with system performance. They feel that more revenue would be lost in the long run if the system slowed down after RAID is installed. They have stated specifically that a system with RAID performing at half the speed of the current system would result in gross revenue dollars per hour declining to $5,000 per hour.

In total, 80% of the system e-business activity involves a database transaction. The database transactions consist of 60% reads and 40% writes. On average, disk access time is 20ms.

The disks on this system are nearly full and are nearing the end of their expected lives, so new ones must be ordered soon. You feel that this is a good time to try to install RAID, even though you'll need to buy extra disks. The disks that are suitable for your system cost $2,000 for each 10GB spindle. The average access time of these new disks is 15ms with an MTTF of 20,000 hours and an

MTTR of 4 hours. You have projected that you will need 60GB of storage to accommodate the existing data as well as the expected data growth over the next 5 years. (All of the disks will be replaced.)

a) Are the people who are against adding RAID to the system correct in their assertion that 50% slower disks will result in revenues declining to $5,000 per hour? Justify your answer.

b) What would be the average disk access time on your system if you decide to use RAID-1?

c) What would be the average disk access time on your system using a RAID-5 array with two sets of four disks if 25% of the database transactions must wait behind one transaction for the disk to become free?

d) Which configuration has a better cost justification, RAID-1 or RAID-5? Explain your answer.

48. **a)** Which of the RAID systems described in this chapter cannot tolerate a single disk failure?

b) Which can tolerate more than one simultaneous disk failure?

49. Our discussion of RAID is biased toward consideration of standard rotating magnetic disks. Is RAID necessary for SSD storage? If not, does this make SSD storage slightly more affordable for the enterprise? If it is necessary, do the redundant disks necessarily also need to be SSD?

FOCUS ON DATA COMPRESSION

7A.1 INTRODUCTION

No matter how cheap storage gets, no matter how much of it we buy, we never seem to be able to get enough of it. New huge disks fill rapidly with all the things we wish we could have put on the old disks. Before long, we are in the market for another set of new disks. Few people or corporations have access to unlimited resources, so we must make optimal use of what we have. One way to do this is to make our data more compact, compressing it before writing it to disk. (In fact, we could even use some kind of compression to make room for a parity or mirror set, adding RAID to our system for "free"!)

Data compression can do more than just save space. It can also save time and help to optimize resources. For example, if compression and decompression are done in the I/O processor, less time is required to move the data to and from the storage subsystem, freeing the I/O bus for other work.

The advantages offered by data compression in sending information over communication lines are obvious: less time to transmit and less storage on the host. Although a detailed study is beyond the scope of this text (see the references section for some resources), you should understand a few basic data compression concepts to complete your understanding of I/O and data storage.

When we evaluate various compression algorithms and compression hardware, we are often most concerned with how fast a compression algorithm executes and how much smaller a file becomes after the compression algorithm is applied. The **compression factor** (sometimes called **compression ratio**) is a statistic that can be calculated quickly and is understandable by virtually anyone who would care. There are a number of different methods used to compute a compression factor. We will use the following:

$$\text{Compression factor} = 1 - \left[\frac{\text{compressed size}}{\text{uncompressed size}}\right] \times 100\%$$

For example, suppose we start with a 100KB file and apply some kind of compression to it. After the algorithm terminates, the file is 40KB in size. We can say that the algorithm achieved a compression factor of $(1 - (\frac{40}{100})) \times 100\% = 60\%$ for this particular file. An exhaustive statistical study should be undertaken before inferring that the algorithm would *always* produce 60% file compression. We can, however, determine an expected compression ratio for particular messages or message types once we have a little theoretical background.

The study of data compression techniques is a branch of a larger field of study called **information theory**. Information theory concerns itself with the way in which information is stored and coded. It was born in the late 1940s through the work of Claude Shannon, a scientist at Bell Laboratories. Shannon established a number of information metrics, the most fundamental of which is **entropy**. Entropy is the measure of information content in a message. Messages with higher entropy carry more information than messages with lower entropy. This definition implies that a message with lower information content would compress to a smaller size than a message with a higher information content.

Determining the entropy of a message requires that we first determine the frequency of each symbol within the message. It is easiest to think of the symbol frequencies in terms of probability. For example, in the famous program output statement:

HELLO WORLD!

the probability of the letter L appearing is $\frac{3}{12}$, or $\frac{1}{4}$. In symbols, we have $P(L) = 0.25$. To map this probability to bits in a code word, we use the base-2 log of this probability. Specifically, the minimum number of bits required to encode the letter L is $-\log_2 P(L)$, or 2. The entropy of the message is the weighted average of the number of bits required to encode each of the symbols in the message. If the probability of a symbol x appearing in a message is $P(x)$, then the entropy, H, of the symbol x is:

$$H = -P(x) \times \log_2 P(x)$$

The average entropy over the entire message is the sum of the weighted probabilities of all n symbols of the message:

$$\sum_{i=1}^{n} -P(x_i) \times \log_2 P(x_i)$$

Entropy establishes a lower bound on the number of bits required to encode a message. Specifically, if we multiply the number of characters in the entire message by the weighted entropy, we get the theoretical minimum of the number of bits that are required to encode the message without loss of information. Bits in addition to this lower bound do not add information. They are therefore **redundant**. The objective of data compression is to remove redundancy while preserving information content. We can quantify the average redundancy for each character contained in a coded message of length n containing code words of length l by the formula:

$$\sum_{i=1}^{n} -P(x_i) \times l_i - \sum_{i=1}^{n} - P(x_i) \times \log_2 P(x_i)$$

This formula is most useful when we are comparing the effectiveness of one coding scheme over another for a given message. The code producing the message with the least amount of redundancy is the better code in terms of data compression. (Of course, we must also consider such things as speed and computational complexity as well as the specifics of the application before we can say that one method is *better* than another.)

Finding the entropy and redundancy for a text message is a straightforward application of the formula above. With a fixed-length code, such as ASCII or EBCDIC, the left-hand sum above is exactly the length of the code, usually 8 bits. In our *HELLO WORLD!* example (using the right-hand sum), we find the average symbol entropy is about 3.022. This means that if we reached the theoretical maximum entropy, we would need only 3.022 bits per character × 12 characters = 36.26, or 37 bits, to encode the entire message. The 8-bit ASCII version of the message, therefore, carries 96 – 37, or 59, redundant bits.

7A.2 STATISTICAL CODING

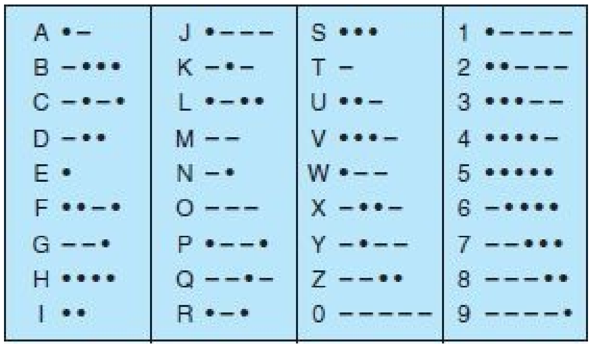
The entropy metric just described can be used as a basis for devising codes that minimize redundancy in the compressed message. Generally, any application of statistical compression is a relatively slow and I/O-intensive process, requiring two passes to read the file before it is compressed and written.

Two passes over the file are needed because the first pass is used for tallying the number of occurrences of each symbol. These tallies are used to calculate probabilities for each different symbol in the source message. Values are assigned to each symbol in the source message according to the calculated probabilities. The newly assigned values are subsequently written to a file along with the information required to decode the file. If the encoded file—along with the table of values needed to decode the file—is smaller than the original file, we say that data compression has occurred.

Huffman and arithmetic coding are two fundamental statistical data compression methods. Variants of these methods can be found in a large number of popular data compression programs. We examine each of these in the next sections, starting with Huffman coding.

7A.2.1 Huffman Coding

Suppose that after we determine probabilities for each of the symbols in the source message, we create a variable-length code that assigns the most frequently used symbols to the shortest code words. If the code words are shorter than the information words, it stands to reason that the resulting compressed message will be shorter as well. David A. Huffman formalized this idea in a paper published in 1952. Interestingly, one form of **Huffman coding**, Morse code, has been around since the early 1800s.



A •–	J •–––	S •••	1 •––––
B –•••	K –•–	T –	2 ••–––
C –•–•	L •–••	U ••–	3 •••––
D –••	M ––	V •••–	4 ••••–
E •	N –•	W •––	5 •••••
F ••–•	O –––	X –••–	6 –••••
G ––•	P •––•	Y –•––	7 ––•••
H ••••	Q ––•–	Z ––••	8 –––••
I ••	R •–•	0 –––––	9 ––––•

FIGURE 7A.1 The International Morse Code

Morse code was designed using typical letter frequencies found in English writing. As you can see in Figure 7A.1, the shorter codes represent the more frequently used letters in the English language. These frequencies clearly cannot apply to every single message. A notable exception would be a telegram from

Uncle Zachary vacationing in Zanzibar, requesting a few quid so he can quaff a quart of quinine! Thus, the most accurate statistical model would be individualized for each message. To accurately assign code words, the Huffman algorithm builds a binary tree using symbol probabilities found in the source message. A traversal of the tree gives the bit pattern assignments for each symbol in the message. We illustrate this process using a simple nursery rhyme. For clarity, we render the rhyme in all uppercase with no punctuation, as follows:

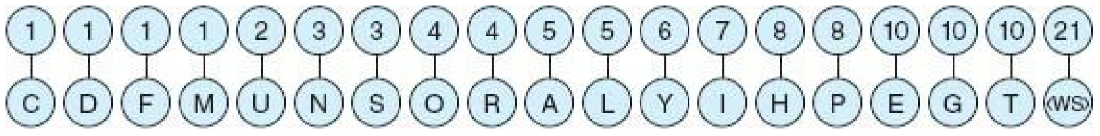
HIGGLETY PIGGLETY POP
THE DOG HAS EATEN THE MOP
THE PIGS IN A HURRY THE CATS IN A FLURRY
HIGGLETY PIGGLETY POP

We start by tabulating all occurrences of each letter in the rhyme. We will use the abbreviation <ws> (white space) for the space characters between each word as well as the newline characters (see Table 7A.1).

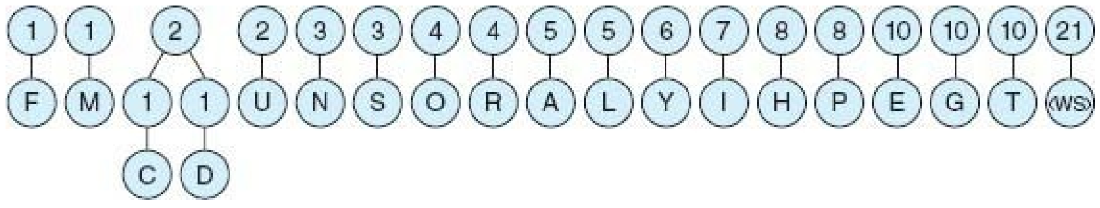
Letter	Count	Letter	Count
A	5	N	3
C	1	O	4
D	1	P	8
E	10	R	4
F	1	S	3
G	10	T	10
H	8	U	2
I	7	Y	6
L	5	<ws>	21
M	1		

TABLE 7A.1 Letter Frequencies

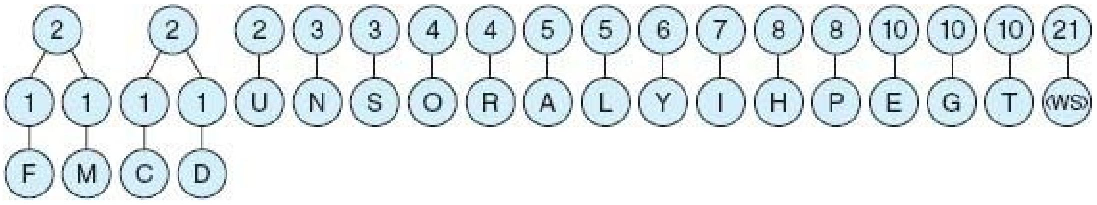
These letter frequencies are associated with each letter using two nodes of a tree. The collection of these trees (a **forest**) is placed in a line ordered by the letter frequencies like this:



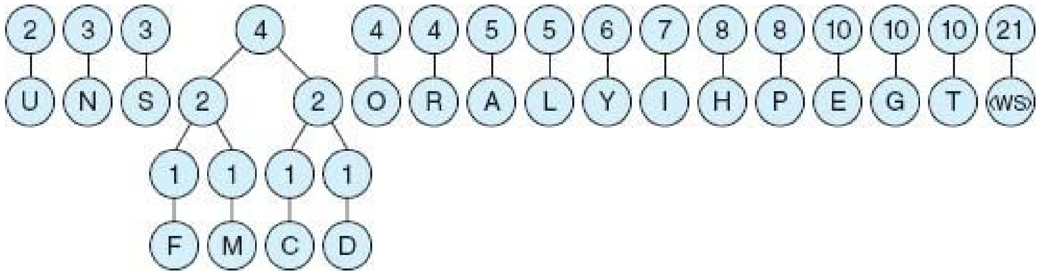
We begin building the binary tree by joining the nodes with the two smallest frequencies. Because we have a four-way tie for the smallest, we arbitrarily select the leftmost two nodes. The sum of the combined frequencies of these two nodes is two. We create a parent node labeled with this sum and place it back into the forest in the location determined by the label on the parent node, as shown:



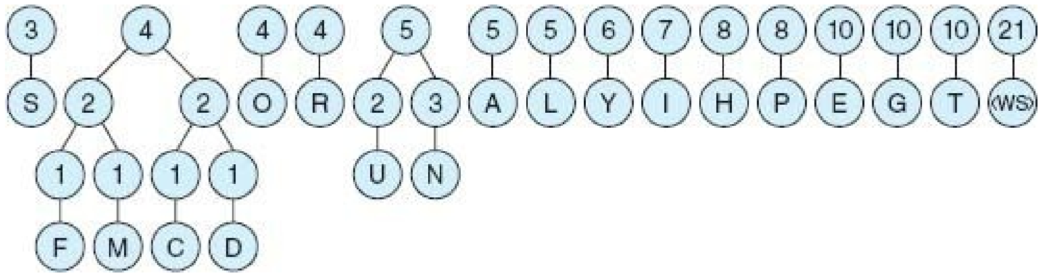
We repeat the process for the nodes now having the lowest frequencies:



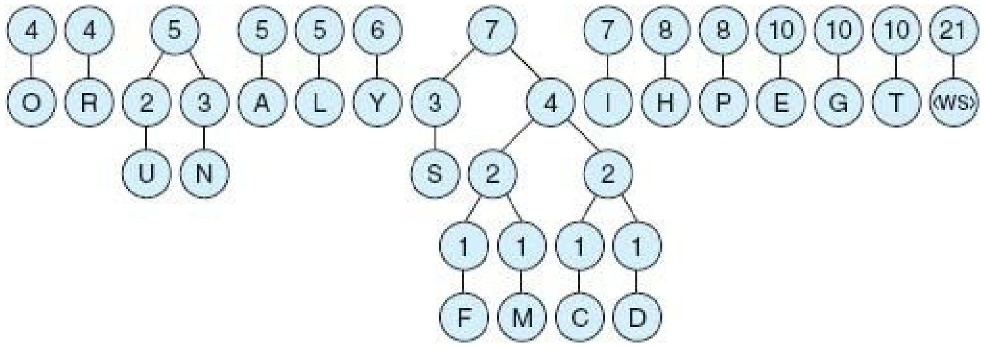
The two smallest nodes are the parents of F, M, C, and D. Taken together, they sum to a frequency of 4, which belongs in the fourth position from the left:



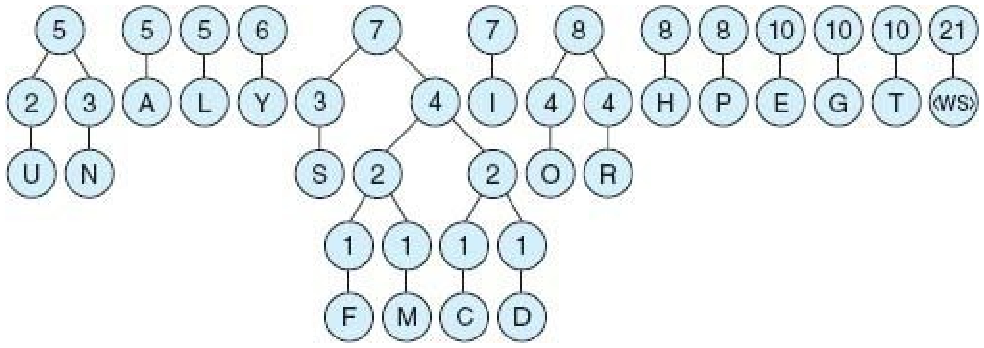
The leftmost nodes add up to 5. They are moved to their new position in the tree as shown:



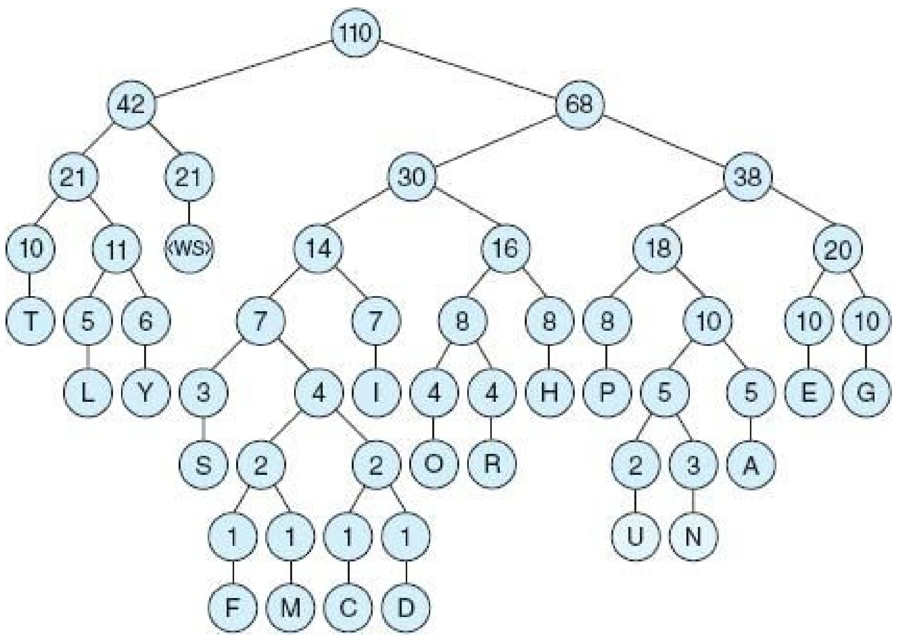
The two smallest nodes now add up to 7. Create a parent node and move the subtree to the middle of the forest with the other node with frequency 7:



The leftmost pair combine to create a parent node with a frequency of 8. It is placed back in the forest as shown:



After several more iterations, the completed tree looks like this:



This tree establishes the framework for assigning a Huffman value to each symbol in the message. We start by labeling every right branch with a binary 1, then each left branch with a binary 0. The result of this step is shown below. (The frequency nodes have been removed for clarity.)

Letter	Code	Letter	Code
<ws>	01	O	10100
T	000	R	10101
L	0010	A	11011
Y	0011	U	110100
I	1001	N	110101
H	1011	F	1000100
P	1100	M	1000101
E	1110	C	1000110
G	1111	D	1000111
S	10000		

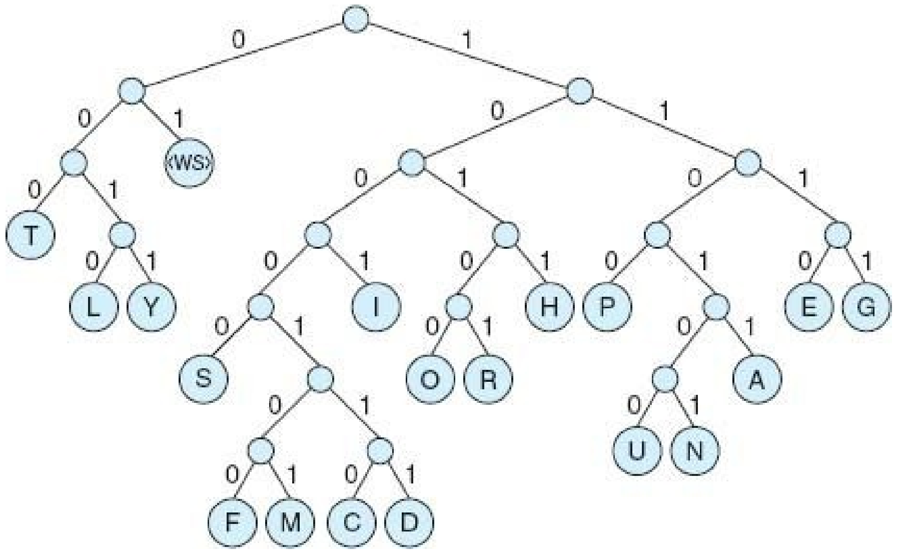


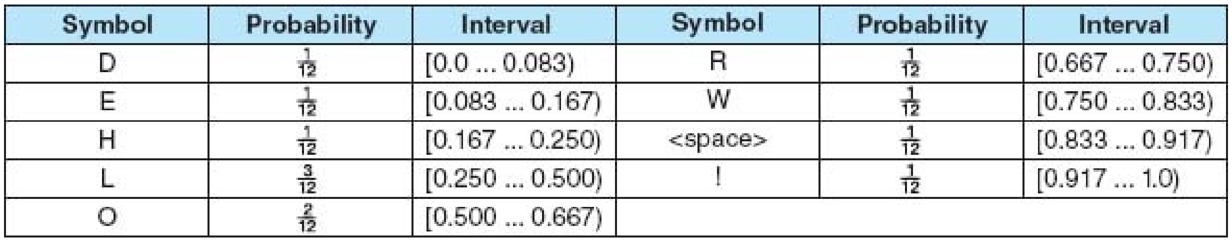
TABLE 7A.2 The Coding Scheme

All we need to do now is traverse the tree from its root to each leaf node, keeping track of the binary digits encountered along the way. The completed coding scheme is shown in Table 7A.2.

As you can see, the symbols with the highest frequencies end up having the fewest bits in their code. The entropy of this message is approximately 3.82 bits per symbol. The theoretical lower bound compression for this message is therefore 110 symbols × 3.82 bits = 421 bits. This Huffman code renders the message in 426 bits, or about 1% more than is theoretically necessary.

7A.2.2 Arithmetic Coding

Huffman coding cannot always achieve theoretically optimal compression because it is restricted to using an integral number of bits in the resulting code. In the nursery rhyme in the previous section, the entropy of the symbol *S* is approximately 1.58. An optimal code would use 1.58 bits to encode each occurrence of *S*. With Huffman coding, we are restricted to using at least 2 bits for this purpose. This lack of precision cost us a total of 5 redundant bits in the end. Not too bad, but it seems we could do better.



Symbol	Probability	Interval	Symbol	Probability	Interval
D	$\frac{1}{12}$	[0.0 ... 0.083)	R	$\frac{1}{12}$	[0.667 ... 0.750)
E	$\frac{1}{12}$	[0.083 ... 0.167)	W	$\frac{1}{12}$	[0.750 ... 0.833)
H	$\frac{1}{12}$	[0.167 ... 0.250)	<space>	$\frac{1}{12}$	[0.833 ... 0.917)
L	$\frac{3}{12}$	[0.250 ... 0.500)	!	$\frac{1}{12}$	[0.917 ... 1.0)
O	$\frac{2}{12}$	[0.500 ... 0.667)			

TABLE 7A.3 Probability Interval Mapping for HELLO WORLD!

Huffman coding falls short of optimality because it is trying to map probabilities—which are elements of the set of real numbers—to elements of a small subset of the set of integers. We are bound to have problems! So why not devise some sort of real-to-real mapping to achieve data compression? In 1963, Norman Abramson conceived of such a mapping, which was subsequently published by Peter Elias. This real-to-real data compression method is called **arithmetic coding**.

Conceptually, arithmetic coding partitions the real number line in the interval between 0 and 1 using the probabilities in the symbol set of the message. More frequently used symbols get a larger chunk of the interval.

Returning to our favorite program output, HELLO WORLD!, we see that there are 12 characters in this imperative statement. The lowest probability among the symbols is $\frac{1}{12}$. All other probabilities are a multiple of $\frac{1}{12}$. Thus, we divide the 0 – 1 interval into 12 parts. Each of the symbols except L and O are assigned $\frac{1}{12}$ of the interval. L and O get $\frac{3}{12}$ and $\frac{2}{12}$, respectively. Our probability-to-interval mapping is shown in Table 7A.3.

We encode a message by successively dividing a range of values (starting with 0.0 through 1.0) proportionate to the interval assigned to the symbol. For example, if the "current interval" is $\frac{1}{8}$ and the letter *L* gets $\frac{1}{4}$ of the current interval, as shown above, then to encode the *L*, we multiply $\frac{1}{8}$ by $\frac{1}{4}$, giving $\frac{1}{32}$ as the new current interval. If the next character is another *L*, $\frac{1}{32}$ is multiplied by $\frac{1}{4}$, yielding $\frac{1}{128}$ for the current interval. We proceed in this vein until the entire message is encoded. This process becomes clear after studying the pseudocode below. A trace of the pseudocode for HELLO WORLD! is given in Figure 7A.2.

```
ALGORITHM Arith_Code (Message)
  HiVal ← 1.0                   /* Upper limit of interval. */
  LoVal ← 0.0                   /* Lower limit of interval. */
  WHILE (more characters to process)
       Char ← Next message character
       Interval ← HiVal – LoVal
       CharHiVal ← Upper interval limit for Char
       CharLoVal ← Lower interval limit for Char
       HiVal ← LoVal +  Interval * CharHiVal
       LoVal ← LoVal +  Interval * CharLoVal
```

Symbol	Interval	CharLoVal	CharHiVal	LoVal	HiVal
				0.0	1.0
H	1.0	0.167	0.25	0.167	0.25
E	0.083	0.083	0.167	0.173889	0.180861
L	0.006972	0.25	0.5	0.1756320	0.1773750
L	0.001743	0.25	0.5	0.17606775	0.17650350
O	0.00043575	0.5	0.667	0.176285625	0.176358395
<sp>	0.00007277025	0.833	0.917	0.1763462426	0.1763523553
W	0.00000611270	0.75	0.833	0.1763508271	0.1763513345
O	0.00000050735	0.5	0.667	0.1763510808	0.1763511655
R	0.00000008473	0.667	0.75	0.1763511373	0.1763511444
L	0.00000000703	0.25	0.5	0.1763511391	0.1763511409
D	0.00000000176	0	0.083	0.1763511391	0.1763511392
!	0.00000000015	0.917	1	0.176351139227	0.176351139239
				0.176351139227	

FIGURE 7A.2 Encoding HELLO WORLD! with Arithmetic Coding

```
  ENDWHILE
  OUTPUT (LoVal)
END Arith_Code
```

The message is decoded using the same process in reverse, as shown by the pseudocode that follows. A trace of the pseudocode is given in Figure 7A.3.

```
ALGORITHM Arith_Decode (CodedMsg)
    Finished ← FALSE
    WHILE NOT Finished
       FoundChar ← FALSE          /* We could do this search much more    */
       WHILE NOT FoundChar        /* efficiently in a real implementation. */
          PossibleChar ← next symbol from the code table
          CharHiVal ← Upper interval limit for PossibleChar
          CharLoVal ← Lower interval limit for PossibleChar
          IF CodedMsg < CharHiVal AND CodedMsg > CharLoVal THEN
             FoundChar ← TRUE
          ENDIF              / * We now have a character whose interval  */
       ENDWHILE              /* surrounds the current message value.     */
       OUTPUT(Matching Char)
       Interval ← CharHiVal - CharLoVal
    CodedMsgInterval ← CodedMsg - CharLoVal
    CodedMsg ← CodedMsgInterval / Interval
    IF CodedMsg = 0.0 THEN
       Finished ← TRUE
    ENDIF
END WHILE
END Arith_Decode
```

Symbol	LowVal	HiVal	Interval	CodedMsg-Interval	CodedMsg
					0.176351139227
H	0.167	0.25	0.083	0.009351139227	0.112664328032
E	0.083	0.167	0.084	0.029664328032	0.353146762290
L	0.25	0.5	0.250	0.103146762290	0.412587049161
L	0.25	0.5	0.250	0.162587049161	0.650348196643
O	0.5	0.667	0.167	0.15034819664	0.90028860265
<sp>	0.833	0.917	0.084	0.0672886027	0.8010547935
W	0.75	0.833	0.083	0.051054793	0.615117994
O	0.5	0.667	0.167	0.11511799	0.6893293
R	0.667	0.75	0.083	0.0223293	0.2690278
L	0.25	0.5	0.250	0.019028	0.076111
D	0	0.083	0.083	0.0761	0.917
!	0.917	1	0.083	0.000	0.000

FIGURE 7A.3 A Trace of Decoding HELLO WORLD!

You may have noticed that neither of the arithmetic coding/decoding algorithms contains any error checking. We have done this for the sake of clarity. Real implementations must guard against floating-point underflow in addition to making sure that the number of bits in the result are sufficient for the entropy of the information.

Differences in floating-point representations can also cause the algorithm to miss the zero condition when the message is being decoded. In fact, an end-of-message character is usually inserted at the end of the message during the coding process to prevent such problems during decoding.

7A.3 ZIV-LEMPEL (LZ) DICTIONARY SYSTEMS

Although arithmetic coding can produce nearly optimal compression, it is even slower than Huffman coding because of the floating-point operations that must be performed during both the encoding and decoding processes. If speed is our first concern, we might wish to consider other compression methods, even if it means that we can't achieve a perfect code. Surely, we would gain considerable speed if we could avoid scanning the input message twice. This is what dictionary methods are all about.

Jacob Ziv and Abraham Lempel pioneered the idea of building a dictionary during the process of reading information and writing encoded bytes. The output of dictionary-based algorithms contains either literals or pointers to information that has previously been placed in the dictionary. Where there is substantial "local" redundancy in the data, such as long strings of spaces or zeros, dictionary-based techniques work exceptionally well. Although referred to as LZ dictionary systems, the name "Ziv-Lempel" is preferred to "Lempel-Ziv" when using the authors' full names.

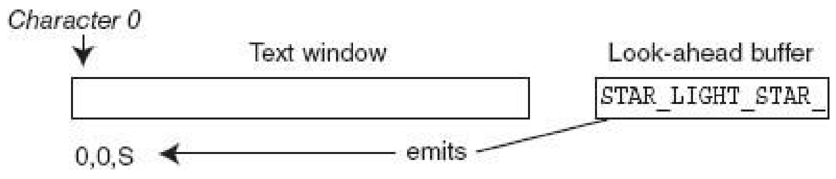
Ziv and Lempel published their first algorithm in 1977. This algorithm, known as the **LZ77 compression algorithm**, uses a text window in conjunction with a look-ahead buffer. The look-ahead buffer contains the information to be encoded. The text window serves as the dictionary. If any characters inside the look-ahead buffer can be found in the dictionary, the location and length of the text in the window is written to the output. If the text cannot be found, the unencoded symbol is written with a flag indicating that the symbol should be used as a literal.

There are many variants of LZ77, all of which build on one basic idea. We will explain this basic version through an example, using another nursery rhyme. We have replaced all spaces by underscores for clarity:

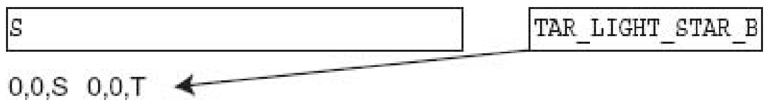
STAR_LIGHT_STAR_BRIGHT_
FIRST_STAR_I_SEE_TONIGHT_
I_WISH_I_MAY_I_WISH_I_MIGHT_
GET_THE_WISH_I_WISH_TONIGHT

For illustrative purposes, we will use a 32-byte text window and a 16-byte look-ahead buffer. (In practice, these two areas usually span several kilobytes.) The text is first read into the look-ahead buffer. Having nothing in the text window yet, the *S* is placed in the text window and a triplet composed of:

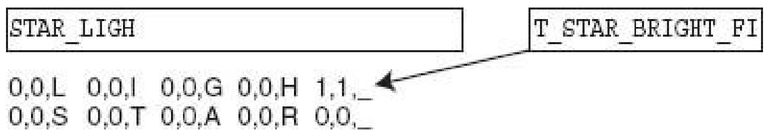
1. The offset to the text in the text window
2. The length of the string that has been matched
3. The first symbol in the look-ahead buffer that follows the phrase



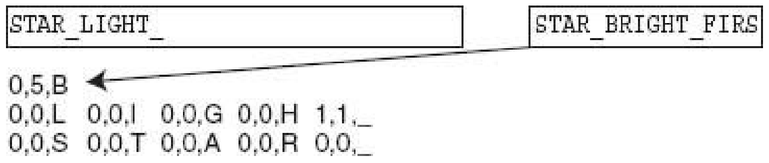
In the example above, there is no match in the text, so the offset and string length are both zeros. The next character in the look-ahead buffer also has no match, so it is also written as a literal with index and length both zero.



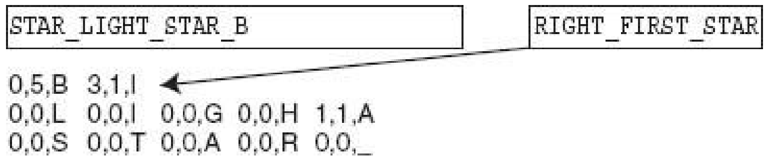
We continue writing literals until a *T* appears as the first character of the look-ahead buffer. This matches the *T* that is in position 1 of the text window. The character following the *T* in the look-ahead buffer is an underscore, which is the third item in the triplet that is written to the output.



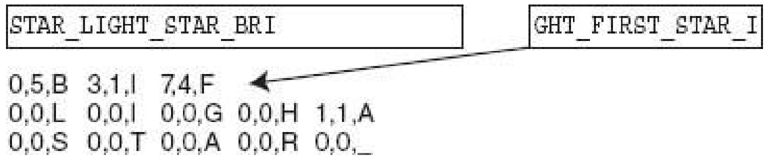
The look-ahead buffer now shifts by two characters. *STAR_* is now at the beginning of the look-ahead buffer. It has a match at the first character position (position 0) of the text window. We write 0, 5, B because *B* is the character following *STAR_* in the buffer.



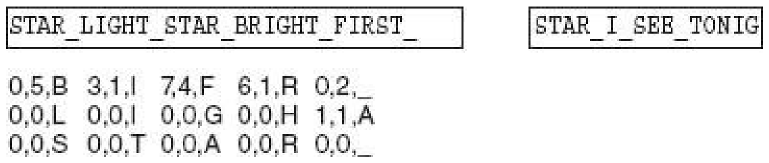
We shift the look-ahead buffer by six characters and look for a match on the *R*. We find one at position 3 of the text, writing 3, 1, I.



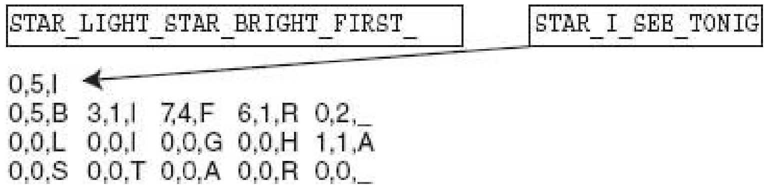
GHT is now at the beginning of the buffer. It matches four characters in the text starting at position 7. We write 7, 4, F.



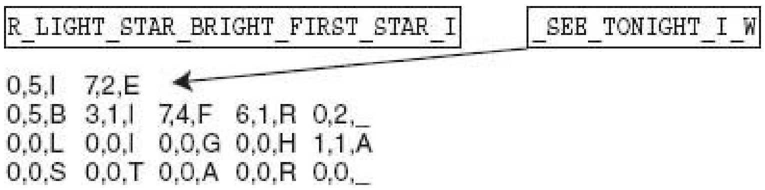
After a few more iterations, the text window is nearly full:


STAR_LIGHT_STAR_BRIGHT_FIRST_
STAR_I_SEE_TONIG
0,5,B 3,1,I 7,4,F 6,1,R 0,2,_
0,0,L 0,0,I 0,0,G 0,0,H 1,1,A
0,0,S 0,0,T 0,0,A 0,0,R 0,0,_

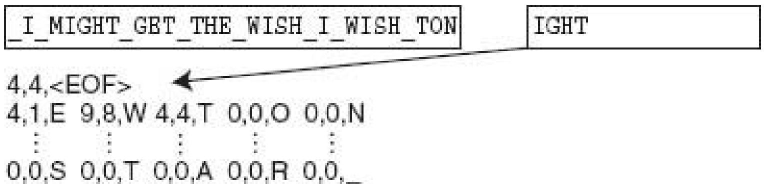
After we match *STAR_* with the characters at position 0 of the text, the six characters, *STAR_I*, shift out of the buffer and into the text window. In order to accommodate all six characters, the text window must shift to the right by three characters after we process *STAR_*.


STAR_LIGHT_STAR_BRIGHT_FIRST_
STAR_I_SEE_TONIG
0,5,I
0,5,B 3,1,I 7,4,F 6,1,R 0,2,_
0,0,L 0,0,I 0,0,G 0,0,H 1,1,A
0,0,S 0,0,T 0,0,A 0,0,R 0,0,_

After writing the code for *STAR_I* and shifting the windows, *_S* is at the beginning of the buffer. These characters match with the text at position 7.


R_LIGHT_STAR_BRIGHT_FIRST_STAR_I
_SEE_TONIGHT_I_W
0,5,I 7,2,E
0,5,B 3,1,I 7,4,F 6,1,R 0,2,_
0,0,L 0,0,I 0,0,G 0,0,H 1,1,A
0,0,S 0,0,T 0,0,A 0,0,R 0,0,_

Continuing in this manner, we ultimately come to the end of the text. The last characters to be processed are *IGHT*. These match the text at position 4. Because there are no characters after *IGHT* in the buffer, the last triple written is tagged with an end of file character, <*EOF*>.


_I_MIGHT_GET_THE_WISH_I_WISH_TON
IGHT
4,4,<EOF>
4,1,E 9,8,W 4,4,T 0,0,O 0,0,N
0,0,S 0,0,T 0,0,A 0,0,R 0,0,_

A total of 36 triples are written to the output in this example. Using a text window of 32 bytes, the index needs only 5 bits to point to any text character. Because the look-ahead buffer is 16 bytes wide, the longest string that we can match is 16 bytes, so we require a maximum of 4 bits to store the length. Using 5 bits for the index, 4 bits for the string length, and 7 bits for each ASCII character, each triple requires 16 bits, or 2 bytes. The rhyme contains 103 characters, which would have been stored in 103 uncompressed bytes on disk. The compressed message requires only 72 bytes, giving us a compression factor of $(1 - (\frac{72}{103})) \times 100 = 30\%$.

It stands to reason that if we make the text window larger, we increase the likelihood of finding matches with the characters in the look-ahead buffer. For example, the string *_TONIGHT* occurs at the forty-first position of the rhyme and again at position 96. Because there are 48 characters between the two occurrences of *_TONIGHT*, the first occurrence cannot possibly be used as a dictionary entry for the second occurrence if we use a 32-character text window. Enlarging the text window to 64 bytes allows the first *_TONIGHT* to be used to encode the second one, and it would add only one bit to each coded triple. In this example, however, an expanded 64-byte text window decreases the output by only two triples: from 36 to 34. Because the text window requires 7 bits for the index, each triple would consist of 17 bits. The compressed message then occupies a total of 17 × 34 = 578 bits, or about 73 bytes. Therefore, the larger text window actually costs us a few bits in this example.

A degenerate condition occurs when there are no matches whatsoever between the text and the buffer during the compression process. For instance, if we would have used a 36-character string consisting of all the letters of the alphabet and the digits 0 through 9, ABC … XYZ012 … 9, we would have had no matches in our example. The output of the algorithm would have been 36 triples of the form, 0,0,?. We would have ended up with an output triple the size of the original string, or an *expansion* of 200%.

Fortunately, exceptional cases like the one just cited happen rarely in practice. Variants of LZ77 can be found in a number of popular compression utilities, including the ubiquitous PKZIP. Several brands of tape and disk drives implement LZ77 directly in the disk control circuitry. This compression takes place at hardware speeds, making it completely transparent to users.

Dictionary-based compression has been an active area of research since Ziv and Lempel published their algorithm in 1977. One year later, Ziv and Lempel improved on their own work when they published their second dictionary-based algorithm, now known as LZ78. LZ78 differs from LZ77 in that it removes the limitation of the fixed-size text window. Instead, it creates a special tree data structure called a **trie**, which is populated by tokens as they are read from the input. (Each interior node of the tree can have as many children as it needs.) Instead of writing characters to the disk as in LZ77, LZ78 writes pointers to the tokens in the trie. The entire trie is written to disk following the encoded message, and read first before decoding the message. (See Appendix A for more information regarding tries.)

7A.4 GIF AND PNG COMPRESSION

Efficient management of the trie of tokens is the greatest challenge for LZ78 implementations. If the dictionary gets too large, the pointers can become larger than the original data. A number of solutions to this problem have been found, one of which has been the source of acrimonious debate and legal action.

In 1984, Terry Welsh, an employee of the Sperry Computer Corporation (now Unisys), published a paper describing an effective algorithm for managing an LZ78-style dictionary. His solution, which involves controlling the sizes of the tokens used in the trie, is called **LZW data compression**, for Lempel-Ziv-Welsh. LZW compression is the fundamental algorithm behind the **graphics interchange format**, **GIF** (pronounced "jiff"), developed by CompuServe engineers and popularized by the World Wide Web. Because Welsh devised his algorithm as part of his official duties at Sperry, Unisys exercised its right to place a patent on it. It has subsequently requested small royalties each time a GIF is used by service providers or high-volume users.[1] LZW is not specific to GIF. It is also used in the tagged image file format (TIFF), other compression programs (including Unix Compress), various software applications (such as PostScript and PDF), and hardware devices (most notably modems). Not surprisingly, the royalty request of Unisys has not been well received within the Web community, some sites blazing with vows to boycott GIFs in perpetuity. Cooler heads have simply circumvented the issue by producing better (or at least different) algorithms, one of which is **PNG**, **portable network graphics**.

Royalty disputes alone did not "cause" PNG (pronounced "ping") to come into being, but they undoubtedly hastened its development. In a matter of months in 1995, PNG went from a draft to an accepted international standard. Amazingly, the PNG specification has had only two minor revisions as of 2005.

PNG offers many improvements over GIF, including:

- User-selectable compression modes: "Faster" or "better" on a scale of 0 to 3, respectively
- Improved compression ratios over GIF, typically 5% to 25% better
- Error detection provided by a 32-bit CRC (ISO 3309/ITU-142)
- Faster initial presentation in progressive display mode
- An open international standard, freely available and sanctioned by the World Wide Web Consortium (W3C) and many other organizations and businesses

PNG uses two levels of compression: First, information is reduced using Huffman coding. The Huffman code is then followed by LZ77 compression using a 32KB text window.

GIF can do one thing that PNG cannot: support multiple images in the same file, giving the illusion of animation (albeit stiffly). To correct this limitation, the Internet community produced the **multiple-image network graphics** algorithm (or **MNG**, pronounced "ming"). MNG is an extension of PNG that allows multiple images to be compressed into one file. These files can be of any type, such as gray-scale, true color, or even JPEGs (see the next section). Version 1.0 of MNG was released in January 2001, with refinements and enhancements sure to follow. With PNG and MNG both freely available (with source code!) over the Internet, one is inclined to think it is only a matter of time before the GIF issue becomes passé.

7A.5 JPEG COMPRESSION

When we see a graphic image such as a photograph on a printed page or computer screen, what we are really looking at is a collection of small dots called **pixels** or **picture elements**. Pixels are particularly noticeable in low-image-quality media like newspapers and comic books. When pixels are small and packed closely together, our eyes perceive a "good quality" image. "Good quality," being a subjective measure, starts at about 300 pixels per inch (120 pixels/cm). On the high end, most people would agree that an image of 1600 pixels per inch (640 pixels/cm) is "good," if not excellent.

Pixels contain the binary coding for the image in a form that can be interpreted by display and printer hardware. Pixels can be coded using any number of bits. If, for example, we are producing a black-and-white line drawing, we can do so using one bit per pixel. The bit is either black (pixel = 0) or white (pixel = 1). If we decide that we'd rather have a grayscale image, we need to think about how many shades of gray will suffice. If we want to have eight shades of gray, we need three bits per pixel. Black would be 000, white 111. Anything in between would be some shade of gray.

Color pixels are produced using a combination of red, green, and blue components. If we want to render an image using eight different shades each of red, green, and blue, we must use three bits for *each color component*. Hence, we need nine bits per pixel, giving $2^9 – 1$ different colors. Black would still be all zeros: R = 000, G = 000, B = 000; white would still be all ones: R = 111, G = 111, B = 111. "Pure" green would be R = 000, G = 111, B = 000. R = 011, G = 000, B = 101 would give us some shade of purple. Yellow would be produced by R = 111, G = 111, B = 000. The more bits we use to represent each color, the closer we get to the "true color" that we see around us. Many computer systems approximate true color using eight bits per color—rendering 256 different shades of each. These 24-bit pixels can display about 16 million different colors.

Let's say that we wish to store a 4in. × 6in. (10 cm × 15 cm) photographic image in such a way as to give us "reasonably" good quality when it is viewed or printed. Using 24 bits (3 bytes) per pixel at 300 pixels per inch, we need 300 × 300 × 6 × 4 × 3 = 6.48MB to store the image. If this 4in. × 6in. photo is part of a sales brochure posted on the Web, we would risk losing customers with dial-up modems once they realize that 20 minutes have passed and they still have not completed downloading the brochure. At 1,600 pixels per inch, storage balloons to just under 1.5GB, which is practically impossible to download and store.

JPEG is a compression algorithm designed specifically to address this problem. Fortunately, photographic images contain a considerable amount of redundant information. Furthermore, some of the information having high theoretical entropy is often of no consequence to the integrity of the image. With these ideas in mind, the ISO and ITU together commissioned a group to formulate an international image compression standard. This group is called the **Joint Photographic Experts Group**, or **JPEG**, pronounced "jay-peg." The first JPEG standard, 10928-1, was finalized in 1992. Major revisions and enhancements to this standard were begun in 1997. The new standard is called JPEG2000 and was finalized in December 2000.

JPEG is a collection of algorithms that provides excellent compression at the expense of some image information loss. Up to this point, we have been describing **lossless data compression**: The data restored from the compressed sequence is precisely the same as it was before compression, barring any computational or media errors. Sometimes we can achieve much better compression if a little information loss can be tolerated. Photographic images lend themselves particularly well to **lossy data compression** because of the human eye's ability to compensate for minor imperfections in graphical images. Of course, some images carry real information and should be subjected to lossy compression only after "quality" has been carefully defined. Medical diagnostic images such as x-rays and electrocardiograms fall into this class. Family album and sales brochure photographs, however, are the kinds of images that can lose considerable “information,” while retaining their illusion of visual “quality.”

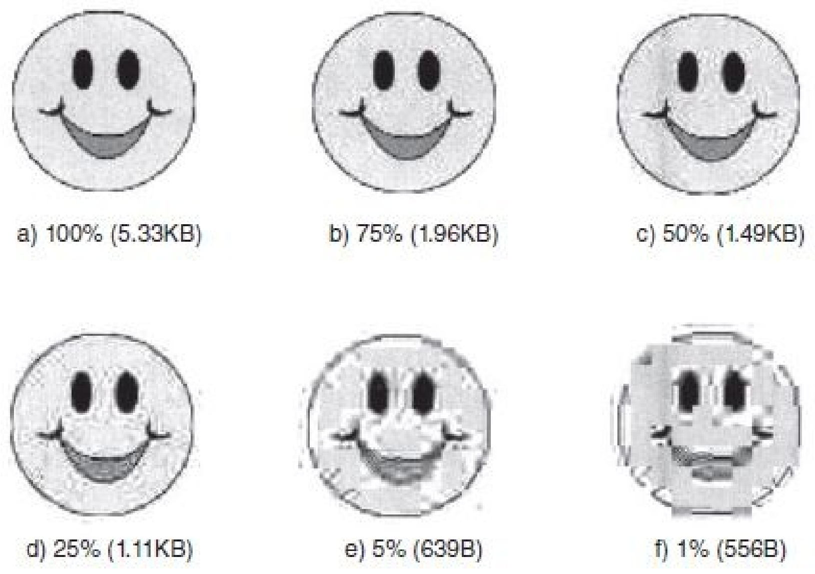


FIGURE 7A.4 JPEG Compression Using Different Quantizations on a 7.14KB Bitmap File

One of the salient features of JPEG is that the user can control the amount of information loss by supplying parameters prior to compressing the image. Even at 100% fidelity, JPEG produces remarkable compression. At 75%, the “lost” information is barely noticeable and the image file is a small fraction of its original size. Figure 7A.4 shows a grayscale image compressed using different quality parameters. (The original 7.14KB bitmap was used as input with the stated quality parameters.)

As you can see, the lossiness of JPEG becomes problematic only when using the lowest quality factors. You’ll also notice how the image takes on the appearance of a crossword puzzle when it is at its highest compression. The reason for this becomes clear once you understand how JPEG works.

When compressing color images, the first thing that JPEG does is to convert the RGB components to the domain of **luminance** and **chrominance**, where luminance is the brightness of the color and chrominance is the color itself. The human eye is less sensitive to chrominance than luminance, so the resulting code is constructed so that the luminance component is least likely to be lost during the subsequent compression steps. Grayscale images do not require this step.

Next, the image is divided into square blocks of eight pixels on each side. These 64-pixel blocks are converted from the spatial domain (*x*, *y*) to a frequency domain (*i*, *j*) using a discrete cosine transform (DCT) as follows:

$$\text{DCT}(i,j) = \frac{1}{4}C(i) \times C(j) \sum_{x=0}^{7}\sum_{y=0}^{7} \text{pixel}(x,y) \times \cos\left[\frac{(2x+1)i\pi}{16}\right] \times \cos\left[\frac{(2y+1)j\pi}{16}\right]$$

where

$$C(a) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } a = 0, \\ 1 & \text{otherwise.} \end{cases}$$

The output of this transform is an 8×8 matrix of integers ranging from –1,024 to 1,023. The pixel at $i = 0$, $j = 0$ is called the **DC coefficient**, and it contains a weighted average of the values of the 64 pixels in the original block. The other 63 values are called **AC coefficients**. Because of the behavior of the cosine function (cos 0 = 1), the resulting frequency matrix, the (i, j) matrix, has a concentration of low numbers and zeros in the lower right-hand corner. Larger numbers collect toward the upper left-hand corner of the matrix. This pattern lends itself well to many different compression methods, but we're not quite ready for that step.

Before the frequency matrix is compressed, each value in the matrix is divided by its corresponding element in a **quantization matrix**. The purpose of the quantization step is to reduce the 11-bit output of the DCT to an 8-bit value. This is the lossy step in JPEG, the degree of which is selectable by the user. The JPEG specification gives several quantization matrices, any of which may be used at the discretion of the implementer. All of these standard matrices ensure that the frequency matrix elements containing the most information (those toward the upper left-hand corner) lose the least amount of their information during the quantization step.

Following the quantization step, the frequency matrix is **sparse**—containing more zero than nonzero entries—in the lower right-hand corner. Large blocks of identical values can be compressed easily using **run-length coding**.

Run-length coding is a simple compression method where instead of coding XXXXX, we code 5,X to indicate a run of five Xs. When we store 5,X instead of XXXXX, we save three bytes, not including any delimiters that the method might require. Clearly, the most effective way of doing this is to try to align everything so that we get as many of the zero values adjacent to each other as we can. JPEG achieves this by doing a **zig-zag scan** of the frequency matrix. The result of this step is a one-dimensional matrix (a vector) that usually contains a long run of zeros. Figure 7A.5 illustrates how the zig-zag scan works.

Each of the AC coefficients in the vector is compressed using run-length coding. The DC coefficient is coded as the arithmetic difference between its original value and the DC coefficient of the previous block, if there is one.

The resulting run-length encoded vector is further compressed using either Huffman or arithmetic coding. Huffman coding is the preferred method because of a number of patents on the arithmetic algorithms.

Figure 7A.6 summarizes the steps we have just described for the JPEG algorithm. Decompression is achieved by reversing this process.

JPEG2000 offers a number of improvements over the JPEG standard of 1997. The underlying mathematics are more sophisticated, which permits greater flexibility with regard to quantization parameters and the incorporation of multiple images into one JPEG file. One of the most striking features of JPEG2000 is its ability to allow user definition of **regions of interest**. A region of interest is an area within an image that serves as a focal point, and would not be subjected to the same degree of lossy compression as the rest of the image. If, for example, you had a photograph of a friend standing on the shore of a lake, you would tell JPEG2000 that your friend is a region of interest. The background of the lake and the trees could be compressed to a great degree before they would lose definition. The image of your friend, however, would remain clear—if not enhanced—by a lower-quality background.

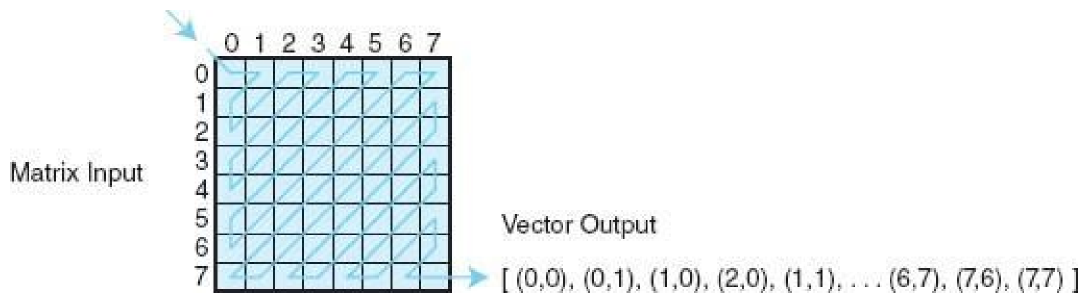


FIGURE 7A.5 A Zig-Zag Scan of a JPEG Frequency Matrix

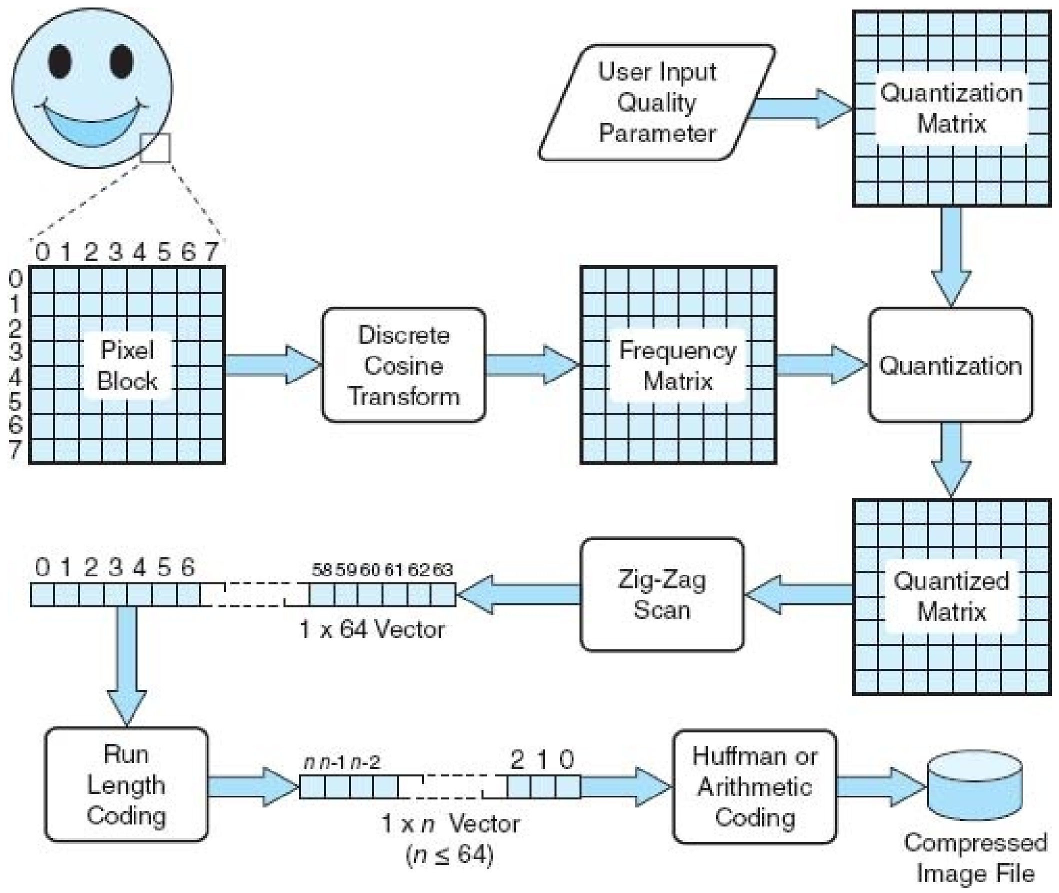


FIGURE 7A.6 The JPEG Compression Algorithm

JPEG2000 replaces the discrete cosine transform of JPEG with a wavelet transform. (Wavelets are a different way of sampling and encoding an image or any set of signals.) Quantization in JPEG2000 uses a sine function rather than the simple division of the earlier version. These more sophisticated mathematical manipulations require substantially more processor resources than JPEG, causing noticeable performance degradation in some applications. Legal concerns have been raised about patent rights to some of the JPEG2000 component algorithms. Thus, many software suppliers are reluctant to incorporate JPEG2000 into their products. It will surely take time for JPEG2000 to realize the popularity of JPEG.

7A.6 MP3 COMPRESSION

Although there are many more powerful audio file compression algorithms, such as AAC, OGG, and WMA, MP3 is universally used and supported. MP3 employs several different compression techniques as well as a bit of human physiology. Although many others helped in its refinement and implementation, German doctoral student, Karlheinz Brandenburg at Erlangen-Nuremberg University is widely recognized as the chief architect of MP3. In 1987, the University and the Fraunhofer Institute for Integrated Circuits formed an alliance to implement Brandenburg’s ideas in electronic circuits. Fraunhofer-Gesellschaft’s refinements of Brandenburg’s work were subsequently adopted by the **Moving Picture Experts Group** (**MPEG**) for compression of the audio component of motion pictures. One of several related audio compression standards, **MP3** is officially known as **MPEG-1 Audio Layer III** or **MPEG-1 Part 3**. It was adopted in 1993 by the International Organization for Standardization as **ISO/IEC 11172-3-1993**. It is important to note that the standard describes only the compression methods. Its implementation—the electronics of the coders and decoders—is left to the creativity of the manufacturers. To fully appreciate the power of MP3 compression, we must first understand how audio signals are recorded onto digital media.

Sound is produced by disturbing air in some way to create a vibration that is ultimately detected by small structures in the human ear. These vibrations are analog waves that have both frequency and amplitude. Converting analog waves to a digital format is achieved by sampling the analog waves at specified intervals. The more frequent the sampling, and the greater number of bits used for the encoding, the more accurately the sound approximates the analog wave.

We illustrate this concept in Figure 7A.7a–c. The *y*-axes in the figures consist of eight values so that the amplitude of the wave can be encoded—modulated to a digital signal—using 3 bits. You can see that the axis is not evenly divided, but has more divisions in the middle of the range. The analog waveform is matched, or quantized, to its nearest integral value. Thus, we get better resolution, or a closer fit to the waveform in areas where most sound occurs. This manner of converting an analog signal to digital is known as **pulse code modulation (PCM)**. Standard audio pulse code modulation uses 16 bits along the *y*-axis.

The next matter to address is the sampling rate. In Figure 7A.7a, the vertical bars represent the sampling intervals taken over the example waveform. It is easy to see that a great deal of the waveform lies outside the rectangles. If we double the sampling rate, as in Figure 7A.7b, we get closer to the waveform. With tripling the sampling rate, we get even closer, as in Figure 7A.7c. As any student of calculus knows, we can keep making these rectangles smaller (by increasing the sampling rate) until the difference between the digital signal and the analog signal becomes too small to measure. However, this “perfection” is costly, as each rectangle requires 2 bytes to encode in PCM. At some point, the encoded audio file becomes too large to be of any practical use.

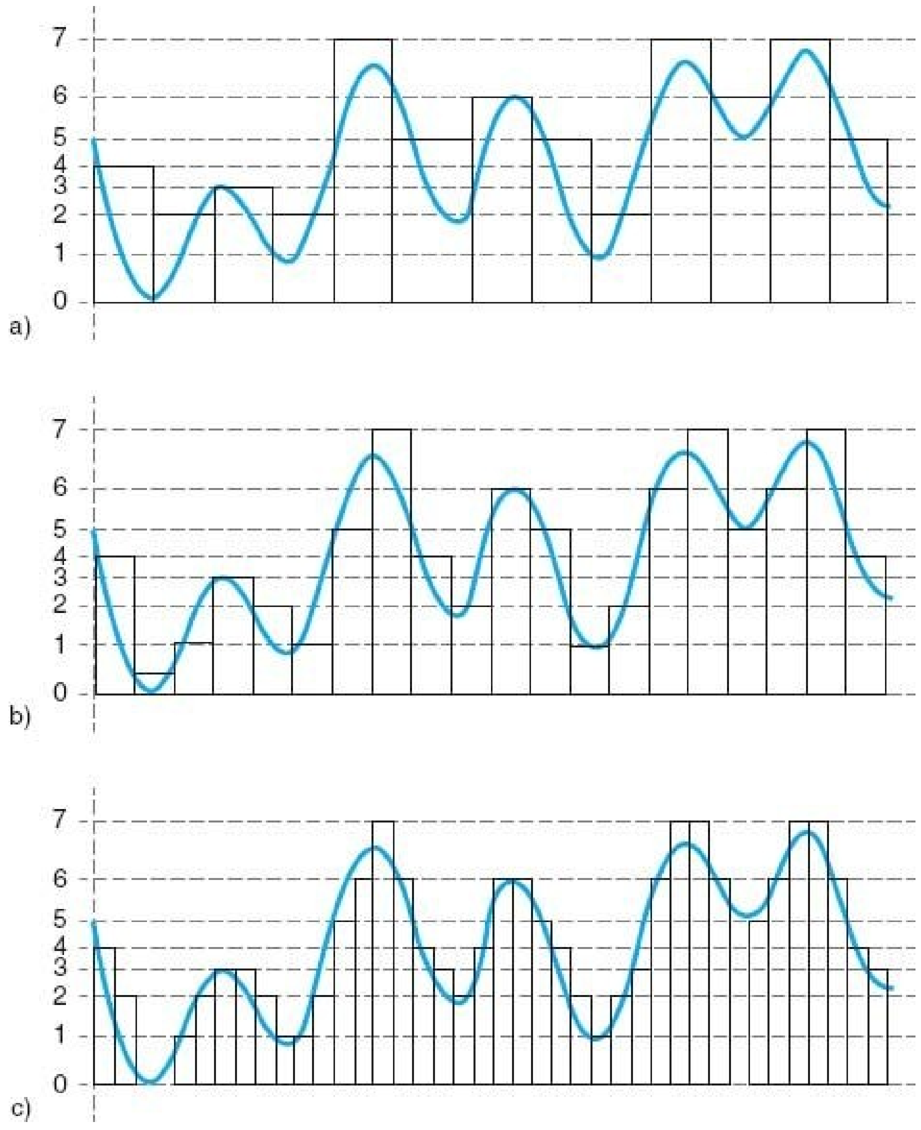


FIGURE 7A.7 Sound Wave Sampling
a) 1× Samples per Second
b) 2× Samples per Second
c) 3× Samples per Second

In the late 1970s, digital recording established a sampling rate of 44,100 samples per second (44.1kHz). At 44.1kHz, an audio signal can be conveyed with sufficient accuracy using electronic circuits that are practical to implement. Using a 44.1kHz sampling rate, and encoding each pulse using 2 bytes, the resulting bit rate is 44,100 samples/s × 16 bits/sample = 705,600 bits/s for each stereo channel. This gives us a total of 705,600 × 2 = 1.41Mb/s for a standard PCM digital signal. Thus, a 3-minute song (in stereo) rendered as a PCM signal produces a 32MB stream (with no error detection or correction applied to the stream). While this file size might be tolerable for CDs, it is completely unacceptable for transmission over data networks. In 1988, the Moving Picture Experts Group was formed under the auspices of the ISO to find a method for compressing audio and video files. Several compression standards emerged from this work, MP3 being the most powerful and widely known. MP3 can provide the approximate quality of CD stereo using less than 2 bits per PCM sample. Our 31MB file can be reduced by a factor of 8, giving a much more manageable file size of about 4MB.[2]

Karlheinz Brandenburg's most brilliant insight in his compression algorithm is his utilization of **psychoacoustic coding**: taking advantage of the imperfect manner in which the human ear perceives sound. The encoding process needs to identify the sounds that the human ear won't perceive and discard them. Thus, MP3 can be very lossy without having any noticeable effects to the untrained listener.

Discarded sounds include those that are at the edges of auditory perception and sounds that are masked (dominated) by other sounds. In addition, the thresholds of perception of sound vary according to frequency as well as volume. Low-frequency tones must be louder than high-frequency sounds. Low-volume, low-frequency sounds can be discarded because the listener can't hear them anyway.

Psychoacoustic coding is just one piece—albeit the most powerful piece—of the highly complex process of MP3 compression. Figure 7A.8 provides a high-level depiction of MP3 encoding. Although a detailed discussion of each step of this process is well beyond the scope of this text, we can describe each in broad terms. (The details can be found in various references provided at the end of this chapter.)

The input to the MP3 encoding process is a PCM audio stream that is dispatched on two parallel paths. In one of them, a **bandpass filterbank** divides the stream into 32 frequency ranges, each of which is then subdivided into 18 subbands, thus giving an output of 576 subbands to be used by the modified discrete cosine transform process.

A **fast Fourier transform** preprocesses the PCM stream to facilitate analysis by the psychoacoustic model. We know that not all of the 576 subbands are necessary for producing good-quality sound; only a subset of them thus needs to be processed. The **psychoacoustic model** determines which sounds fall outside the range of human hearing and which are masked by other sounds in nearby frequency bands. (The masking thresholds are a function of frequency: The thresholds are higher at the middle ranges of audible sound.) The psychoacoustic model is subsequently used both by a modified discrete cosine transform and by the quantization process.

The **modified discrete cosine transform** (**MDCT**) process applies the psychoacoustic model to the 576 subbands output by the bandpass filterbank. Generally speaking, the purpose of a discrete cosine transform (DCT) is to map an input in the form of image intensity or sound amplitude to the frequency domain expressed in terms of the cosine function. The DCT of MP3 is called *modified* because, unlike a "pure" DCT, it processes the 576 subband filterbank output through a set of overlapping windows. This overlapping prevents the noise that occurs at window boundaries later in the encoding process.

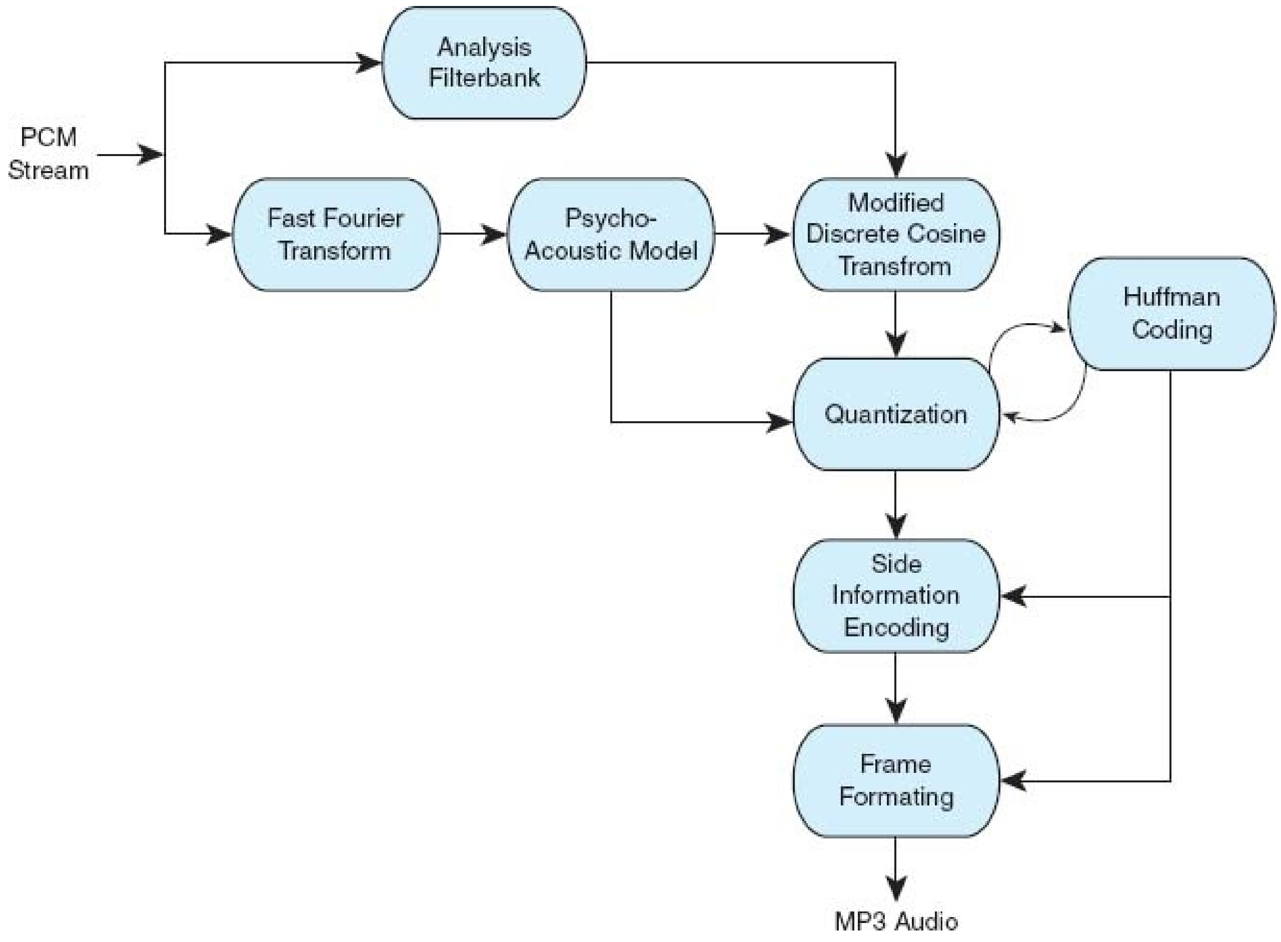


FIGURE 7A.8 Generalized MP3

The output of the MDCT in the form of cosine functions is passed to a quantization function that maps the real-valued cosine function into the integer domain. Such quantization always involves the loss of data because, as discussed in Chapter 2, real values cannot map precisely into the integer domain. MP3 can use 64 bits or 128 bits per second in its quantization, with 128 bits giving better resolution. The psychoacoustic model helps the quantizer find the best fit onto the integer domain, thus reducing the loss of important sounds. Instead of using a fixed quantization, as in the creation of the PCM code shown above, MP3 uses variable **scalefactor bands** that quantize different parts of the sound differently depending on the sensitivity of the human ear to the sound. Less sensitivity allows a wider scalefactor band. The scalefactors are compressed along with the quantized sound using a type of Huffman coding to ensure that the sound is decompressed using the same scalefactor.

Side information is a type of MP3 metadata describing the particulars of the Huffman coding process, the scalefactor bands, and other information that will be crucial to the decoding process. This information is compressed and organized into 17- or 32-byte frames for mono or stereo streams, respectively.

The final step is assembly of the MP3 frame. The MP3 frame consists of a 32-bit header, optional CRC block, side information, MP3 data payload, and optional ancillary data. The actual length of the MP3 frame varies depending on the PCM sampling bitrate and scalefactors used to quantize the data.

It is easy to see that this level of complexity requires implementation in hardware to be able to encode audio in real time and to decode it quickly enough as to render output at the 44.1kHz rate at which the original PCM stream was sampled. The MP3 encoder–decoders **(codecs)** are available inexpensively from several suppliers.

In the end, it is the marriage of advanced mathematics, superb algorithms, and digital circuits that has created the “miracle” of MP3. It has changed the entire music industry by enabling inexpensive distribution channels for all types of music. As a consequence, the hegemony over the music industry enjoyed by a few large music companies was ended, as many other companies could inexpensively enter the business. MP3 has also brought ubiquity to the podcast medium, enabling worldwide sharing of news and information. University lectures and seminars can be delivered to any portable player in a few clicks. MP3's mathematics, algorithms, and electronic circuits are indeed amazing. But even more amazing is how they have changed our world. We can't imagine our world without them.

7A.7 SUMMARY

This special section has presented you with a brief survey of information theory and data compression. We have seen that compression seeks to remove redundant information from data so that only its information content is stored. This saves storage space and improves data transfer speeds for archival storage. Several popular data compression methods were presented; including statistical Huffman coding, Ziv-Lempel dictionary coding, and the compression methods of the Internet, JPG, GIF, and MP3. Dictionary systems compress data in one pass; other methods require at least two: one pass to gather information about the data to be compressed, and the second to carry out the compression process. Most importantly, you now have a sufficient foundation in each type of data compression so that you can select the best method for any particular application.

FURTHER READING

Definitive sources for the theory and application of data compression include works by Salomon (2006), Lelewer and Hirschberg (1987), and Sayood (2012). Sayood's work provides an in-depth description of the entire suite of MPEG compression algorithms. Nelson and Gailly's (1996) thorough treatment—with source code—is clear and easy to read. They make learning the arcane art of data compression a truly pleasurable experience. A wealth of information relevant to data compression can be found on the Web as well. Any good search engine will point you to hundreds of links when you search any of the key data compression terms introduced in this chapter. Wavelet theory is gaining importance in the area of data compression and data communications. If you want to delve into this heady area, you may wish to start with Vetterli and Kovačević (1995). This book also contains an exhaustive account of image compression, including JPEG and, of course, the wavelet theory behind JPEG2000.

We barely scratched the surface of the complexity of MP3 in this section. If you would like to understand the details, the master's thesis by Sripada (2006) contains a wonderfully readable account of the process. The articles by Pan (1995) and Noll (1997) also give a detailed treatment of the subject. A great deal of information concerning all manner of MPEG coding can be found at http://www.mpeg.org/MPEG/audio.htm1. The definitive history of MP3 is posted at http://www.mp3-history.com. It is well worth your time to see how the story unfolds!

REFERENCES

Lelewer, D. A. and Hirschberg, D. S. "Data Compression." *ACM Computing Surveys 19:*3, 1987, pp. 261–297.

Nelson, M., & Gailly, J. *The Data Compression Book,* 2nd ed. New York: M&T Books, 1996.

Noll, P. "MPEG Digital Audio Coding," *IEEE Signal Processing Magazine 14*:5, September 1997, pp. 59–81.

Pan, D. “A Tutorial on MPEG/Audio Compression.” *IEEE Multimedia 2*:2, Summer 1995, pp. 60–74.

Salomon, D. *Data Compression: The Complete Reference*, 4th ed. New York: Springer, 2006.

Sayood, K. *Introduction to Data Compression*, 4th ed. San Mateo, CA: Morgan Kaufmann, 2012.

Sripada, P. “MP3 Decoder in Theory and Practice.” Master’s Thesis: MEE06:09, Blekinge Tekniska Högskola, March 2006. Available at http://sea-mist.se/fou/cuppsats.nsf/all/857e49b9bfa2d753c125722700157b97/$file/Thesis%20report-%20MP3%20Decoder.pdf. Retrieved September 1, 2013.

Vetterli, M., & Kovačević, J. *Wavelets and Subband Coding*. Englewood Cliffs, NJ: Prentice Hall PTR, 1995.

Welsh, T. “A Technique for High-Performance Data Compression.” *IEEE Computer 17*:6, June 1984, pp. 8–19.

Ziv, J., & Lempel, A. “A Universal Algorithm for Sequential Data Compression.” *IEEE Transactions on Information Theory 23*:3, May 1977, pp. 337–343.

Ziv, J., & Lempel, A. “Compression of Individual Sequences via Variable-Rate Coding.” *IEEE Transactions on Information Theory 24*:5, September 1978, pp. 530–536.

EXERCISES

1. Who was the founder of the science of information theory? During which decade did he do his work?

2. What is information entropy, and how does it relate to information redundancy?

3. **a)** Name two types of statistical coding.

b) Name an advantage and a disadvantage of statistical coding.

4. Use arithmetic coding to compress your name. Can you get it back after you have compressed it?

5. Compute the compression factors for each of the JPEG images in Figure 7A.4.

6. Create a Huffman tree and assign Huffman codes for the “Star Bright” rhyme used in Section 7A.3. Use <ws> for whitespace instead of underscores.

7. Complete the LZ77 data compression illustrated in Section 7A.3.

8. JPEG is a poor choice for compressing line drawings, such as the one shown in Figure 7A.4. Why do you think this is the case? What other compression methods would you suggest? Give justification for your choice(s).

9. **a)** The LZ77 compression algorithm falls into which class of data compression algorithms?

b) Name an advantage of Huffman coding over LZ77.

c) Name an advantage of LZ77 over Huffman coding.

d) Which is better?

10. State one feature of PNG that you could use to convince someone that PNG is a better algorithm than GIF.

[1] The U.S. patent on Gif expired in 2003.

[2] Reductions in sampling rate can make the MP3 file even smaller, but the sound quality is reduced. Popular sampling rates for digital audio

include 8,000Hz, 11,025Hz, 16,000Hz, 22,050Hz, 44,100Hz, 48,000Hz, and 96,000Hz.