

Computer	Bits/cell
Burroughs B1700	1
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

Figure 2-10. Number of bits per cell for some historically interesting commercial computers.

whereas a 64-bit machine will have 64-bit registers and instructions for moving, adding, subtracting, and otherwise manipulating 64-bit words.

2.2.3 Byte Ordering

The bytes in a word can be numbered from left to right or right to left. At first it might seem that this choice is unimportant, but as we shall see shortly, it has major implications. Figure 2-11(a) depicts part of the memory of a 32-bit computer whose bytes are numbered from left to right, such as the SPARC or the big IBM mainframes. Figure 2-11(b) gives the analogous representation of a 32-bit computer using right-to-left numbering, such as the Intel family. The former system, where the numbering begins at the “big” (i.e., high-order) end is called a **big endian** computer, in contrast to the **little endian** of Fig. 2-11(b). These terms are due to Jonathan Swift, whose *Gulliver’s Travels* satirized politicians who made war over their dispute about whether eggs should be broken at the big end or the little end. The term was first used in computer architecture in a delightful article by Cohen (1981).

It is important to understand that in both the big endian and little endian systems, a 32-bit integer with the numerical value of, say, 6, is represented by the bits 110 in the rightmost (low-order) 3 bits of a word and zeros in the leftmost 29 bits. In the big endian scheme, the 110 bits are in byte 3 (or 7, or 11, etc.), whereas in the little endian scheme they are in byte 0 (or 4, or 8, etc.). In both cases, the word containing this integer has address 0.

If computers stored only integers, there would be no problem. However, many applications require a mixture of integers, character strings, and other data types. Consider, for example, a simple personnel record consisting of a string (employee

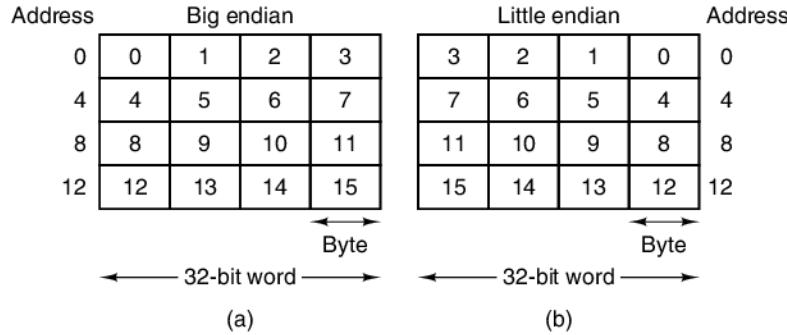


Figure 2-11. (a) Big endian memory. (b) Little endian memory.

name) and two integers (age and department number). The string is terminated with 1 or more 0 bytes to fill out a word. The big endian representation is shown in Fig. 2-12(a); the little endian representation is shown in Fig. 2-12(b) for Jim Smith, age 21, department 260 ($1 \times 256 + 4 = 260$).

The diagram shows four tables labeled (a) through (d). Table (a) is a 'Big endian' personnel record with fields J, I, M, S, T, H, 0, 0, 0, 21, 0, 0, 1, 4. Table (b) is the same record in 'Little endian' format: M, I, J, S, T, I, M, S, 0, 0, 0, H, 0, 0, 0, 21, 0, 0, 1, 4. Table (c) shows the result of 'Transfer from big endian to little endian', where the bytes are swapped: J, I, M, S, T, I, M, S, 0, 0, 0, H, 21, 0, 0, 0, 4, 1, 0, 0. Table (d) shows the result of 'Transfer and swap', which is identical to table (c): J, I, M, S, T, I, M, S, 0, 0, 0, H, 21, 0, 0, 0, 4, 1, 0, 0.

Figure 2-12. (a) A personnel record for a big endian machine. (b) The same record for a little endian machine. (c) The result of transferring the record from a big endian to a little endian. (d) The result of byte swapping (c).

Both of these representations are fine and internally consistent. The problems begin when one of the machines tries to send the record to the other one over a network. Let us assume that the big endian sends the record to the little endian one byte at a time, starting with byte 0 and ending with byte 19. (We will be optimistic and assume the bits of the bytes are not reversed by the transmission, as we have enough problems as is.) Thus the big endian's byte 0 goes into the little endian's memory at byte 0, and so on, as shown in Fig. 2-12(c).

When the little endian tries to print the name, it works fine, but the age comes out as 21×2^{24} and the department is just as garbled. This situation arises because

the transmission has reversed the order of the characters in a word, as it should, but it has also reversed the bytes in an integer, which it should not.

An obvious solution is to have the software reverse the bytes within a word after the copy has been made. Doing this leads to Fig. 2-12(d) which makes the two integers fine but turns the string into “MIJTIMS” with the “H” hanging in the middle of nowhere. This reversal of the string occurs because when reading it, the computer first reads byte 0 (a space), then byte 1 (M), and so on.

There is no simple solution. One way that works, but is inefficient, is to include a header in front of each data item telling what kind of data follows (string, integer, or other) and how long it is. This allows the receiver to perform only the necessary conversions. In any event, it should be clear that the lack of a standard for byte ordering is a big nuisance when moving data between different machines.

2.2.4 Error-Correcting Codes

Computer memories occasionally make errors due to voltage spikes on the power line, cosmic rays, or other causes. To guard against such errors, some memories use error-detecting or error-correcting codes. When these codes are used, extra bits are added to each memory word in a special way. When a word is read out of memory, the extra bits are checked to see if an error has occurred.

To understand how errors can be handled, it is necessary to look closely at what an error really is. Suppose that a memory word consists of m data bits to which we will add r redundant, or check, bits. Let the total length be n (i.e., $n = m + r$). An n -bit unit containing m data and r check bits is often referred to as an *n-bit codeword*.

Given any two codewords, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ. In this case, 3 bits differ. To determine how many bits differ, just compute the bitwise Boolean EXCLUSIVE OR of the two codewords and count the number of 1 bits in the result. The number of bit positions in which two codewords differ is called the **Hamming distance** (Hamming, 1950). Its main significance is that if two codewords are a Hamming distance d apart, it will require d single-bit errors to convert one into the other. For example, the codewords 11110001 and 00110000 are a Hamming distance 3 apart because it takes 3 single-bit errors to convert one into the other.

With an m -bit memory word, all 2^m bit patterns are legal, but due to the way the check bits are computed, only 2^m of the 2^n codewords are valid. If a memory read turns up an invalid codeword, the computer knows that a memory error has occurred. Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list find the two codewords whose Hamming distance is minimum. This distance is the Hamming distance of the complete code.

The error-detecting and error-correcting properties of a code depend on its Hamming distance. To detect d single-bit errors, you need a distance $d + 1$ code

because with such a code there is no way that d single-bit errors can change a valid codeword into another valid codeword. Similarly, to correct d single-bit errors, you need a distance $2d + 1$ code because that way the legal codewords are so far apart that even with d changes, the original codeword is still closer than any other codeword, so it can be uniquely determined.

As a simple example of an error-detecting code, consider a code in which a single **parity bit** is appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). Such a code has a distance 2, since any single-bit error produces a codeword with the wrong parity. In other words, it takes two single-bit errors to go from a valid codeword to another valid codeword. It can be used to detect single errors. Whenever a word containing the wrong parity is read from memory, an error condition is signaled. The program cannot continue, but at least no incorrect results are computed.

As a simple example of an error-correcting code, consider a code with only four valid codewords:

0000000000, 0000011111, 1111100000, and 1111111111

This code has a distance 5, which means that it can correct double errors. If the codeword 0000000111 arrives, the receiver knows that the original must have been 0000011111 (if there was no more than a double error). If, however, a triple error changes 0000000000 into 0000000111, the error cannot be corrected.

Imagine that we want to design a code with m data bits and r check bits that will allow all single-bit errors to be corrected. Each of the 2^m legal memory words has n illegal codewords at a distance 1 from it. These are formed by systematically inverting each of the n bits in the n -bit codeword formed from it. Thus each of the 2^m legal memory words requires $n + 1$ bit patterns dedicated to it (for the n possible errors and correct pattern). Since the total number of bit patterns is 2^n , we must have $(n + 1)2^m \leq 2^n$. Using $n = m + r$, this requirement becomes $(m + r + 1) \leq 2^r$. Given m , this puts a lower limit on the number of check bits needed to correct single errors. Figure 2-13 shows the number of check bits required for various memory word sizes.

Word size	Check bits	Total size	Percent overhead
8	4	12	50
16	5	21	31
32	6	38	19
64	7	71	11
128	8	136	6
256	9	265	4
512	10	522	2

Figure 2-13. Number of check bits for a code that can correct a single error.

This theoretical lower limit can be achieved using a method due to Richard Hamming (1950). Before taking a look at Hamming's algorithm, let us look at a simple graphical representation that clearly illustrates the idea of an error-correcting code for 4-bit words. The Venn diagram of Fig. 2-14(a) contains three circles, A, B, and C, which together form seven regions. As an example, let us encode the 4-bit memory word 1100 in the regions AB, ABC, AC, and BC, 1 bit per region (in alphabetical order). This encoding is shown in Fig. 2-14(a).

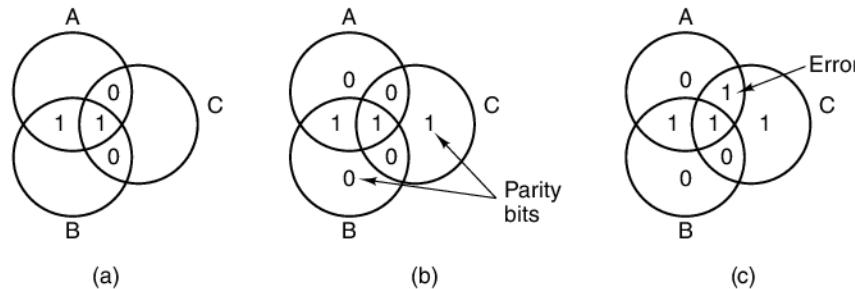


Figure 2-14. (a) Encoding of 1100. (b) Even parity added. (c) Error in AC.

Next we add a parity bit to each of the three empty regions to produce even parity, as illustrated in Fig. 2-14(b). By definition, the sum of the bits in each of the three circles, A, B, and C, is now an even number. In circle A, we have the four numbers 0, 0, 1, and 1, which add up to 2, an even number. In circle B, the numbers are 1, 1, 0, and 0, which also add up to 2, an even number. Finally, in circle C, we have the same thing. In this example all the circles happen to be the same, but sums of 0 and 4 are also possible in other examples. This figure corresponds to a codeword with 4 data bits and 3 parity bits.

Now suppose that the bit in the AC region goes bad, changing from a 0 to a 1, as shown in Fig. 2-14(c). The computer can now see that circles A and C have the wrong (odd) parity. The only single-bit change that corrects them is to restore AC back to 0, thus correcting the error. In this way, the computer can repair single-bit memory errors automatically.

Now let us see how Hamming's algorithm can be used to construct error-correcting codes for any size memory word. In a Hamming code, r parity bits are added to an m -bit word, forming a new word of length $m + r$ bits. The bits are numbered starting at 1, not 0, with bit 1 the leftmost (high-order) bit. All bits whose bit number is a power of 2 are parity bits; the rest are used for data. For example, with a 16-bit word, 5 parity bits are added. Bits 1, 2, 4, 8, and 16 are parity bits, and all the rest are data bits. In all, the memory word has 21 bits (16 data, 5 parity). We will (arbitrarily) use even parity in this example.

Each parity bit checks specific bit positions; the parity bit is set so that the total number of 1s in the checked positions is even. The bit positions checked by the parity bits are

Bit 1 checks bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.

Bit 2 checks bits 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.

Bit 4 checks bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21.

Bit 8 checks bits 8, 9, 10, 11, 12, 13, 14, 15.

Bit 16 checks bits 16, 17, 18, 19, 20, 21.

In general, bit b is checked by those bits b_1, b_2, \dots, b_j such that $b_1 + b_2 + \dots + b_j = b$. For example, bit 5 is checked by bits 1 and 4 because $1 + 4 = 5$. Bit 6 is checked by bits 2 and 4 because $2 + 4 = 6$, and so on.

Figure 2-15 shows construction of a Hamming code for the 16-bit memory word 1111000010101110. The 21-bit codeword is 00101110000101101110. To see how error correction works, consider what would happen if bit 5 were inverted by an electrical surge on the power line. The new codeword would be 001001100000101101110 instead of 001011100000101101110. The 5 parity bits will be checked, with the following results:

Parity bit 1 incorrect (1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21 contain five 1s).

Parity bit 2 correct (2, 3, 6, 7, 10, 11, 14, 15, 18, 19 contain six 1s).

Parity bit 4 incorrect (4, 5, 6, 7, 12, 13, 14, 15, 20, 21 contain five 1s).

Parity bit 8 correct (8, 9, 10, 11, 12, 13, 14, 15 contain two 1s).

Parity bit 16 correct (16, 17, 18, 19, 20, 21 contain four 1s).

The total number of 1s in bits 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, and 21 should be an even number because even parity is being used. The incorrect bit must be one of the bits checked by parity bit 1—namely, bit 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, or 21. Parity bit 4 is incorrect, meaning that one of bits 4, 5, 6, 7, 12, 13, 14, 15, 20, or 21 is incorrect. The error must be one of the bits in both lists, namely, 5, 7, 13, 15, or 21. However, bit 2 is correct, eliminating 7 and 15. Similarly, bit 8 is correct, eliminating 13. Finally, bit 16 is correct, eliminating 21. The only bit left is bit 5, which is the one in error. Since it was read as a 1, it should be a 0. In this manner, errors can be corrected.

A simple method for finding the incorrect bit is first to compute all the parity bits. If all are correct, there was no error (or more than one). Then add up all the incorrect parity bits, counting 1 for bit 1, 2 for bit 2, 4 for bit 4, and so on. The resulting sum is the position of the incorrect bit. For example, if parity bits 1 and 4 are incorrect but 2, 8, and 16 are correct, bit 5 ($1 + 4$) has been inverted.

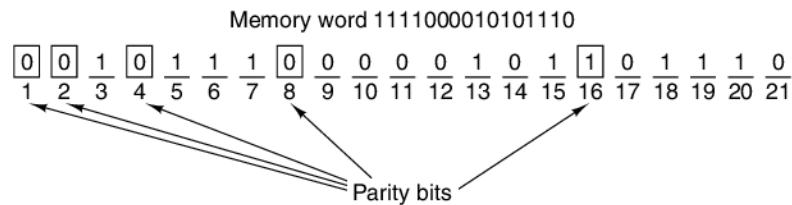


Figure 2-15. Construction of the Hamming code for the memory word 1111000010101110 by adding 5 check bits to the 16 data bits.

2.2.5 Cache Memory

Historically, CPUs have always been faster than memories. As memories have improved, so have CPUs, preserving the imbalance. In fact, as it becomes possible to put more and more circuits on a chip, CPU designers are using these new facilities for pipelining and superscalar operation, making CPUs go even faster. Memory designers have usually used new technology to increase the capacity of their chips, not the speed, so the problem appears to be getting worse over time. What this imbalance means in practice is that after the CPU issues a memory request, it will not get the word it needs for many CPU cycles. The slower the memory, the more cycles the CPU will have to wait.

As we pointed out above, there are two ways to deal with this problem. The simplest way is to just start memory READs when they are encountered but continue executing and stall the CPU if an instruction tries to use the memory word before it has arrived. The slower the memory, the greater the penalty when it does occur. For example, if one instruction in five touches memory and the memory access time is five cycles, execution time will be twice what it would have been with instantaneous memory. But if the memory access time is 50 cycles, then execution time will be up by a factor of 11 (5 cycles for executing instructions plus 50 cycles for waiting for memory).

The other solution is to have machines that do not stall but instead require the compilers not to generate code to use words before they have arrived. The trouble is that this approach is far easier said than done. Often after a LOAD there is nothing else to do, so the compiler is forced to insert NOP (no operation) instructions, which do nothing but occupy a slot and waste time. In effect, this approach is a software stall instead of a hardware stall, but the performance degradation is the same.

Actually, the problem is not technology, but economics. Engineers know how to build memories that are as fast as CPUs, but to run them at full speed, they have to be located on the CPU chip (because going over the bus to memory is very slow). Putting a large memory on the CPU chip makes it bigger, which makes it more expensive, and even if cost were not an issue, there are limits to how big a

CPU chip can be made. Thus the choice comes down to having a small amount of fast memory or a large amount of slow memory. What we would prefer is a large amount of fast memory at a low price.

Interestingly enough, techniques are known for combining a small amount of fast memory with a large amount of slow memory to get the speed of the fast memory (almost) and the capacity of the large memory at a moderate price. The small, fast memory is called a **cache** (from the French *cacher*, meaning to hide, and pronounced “cash”). Below we will briefly describe how caches are used and how they work. A more detailed description will be given in Chap. 4.

The basic idea behind a cache is simple: the most heavily used memory words are kept in the cache. When the CPU needs a word, it first looks in the cache. Only if the word is not there does it go to main memory. If a substantial fraction of the words are in the cache, the average access time can be greatly reduced.

Success or failure thus depends on what fraction of the words are in the cache. For years, people have known that programs do not access their memories completely at random. If a given memory reference is to address A , it is likely that the next memory reference will be in the general vicinity of A . A simple example is the program itself. Except for branches and procedure calls, instructions are fetched from consecutive locations in memory. Furthermore, most program execution time is spent in loops, in which a limited number of instructions are executed over and over. Similarly, a matrix manipulation program is likely to make many references to the same matrix before moving on to something else.

The observation that the memory references made in any short time interval tend to use only a small fraction of the total memory is called the **locality principle** and forms the basis for all caching systems. The general idea is that when a word is referenced, it and some of its neighbors are brought from the large slow memory into the cache, so that the next time it is used, it can be accessed quickly. A common arrangement of the CPU, cache, and main memory is illustrated in Fig. 2-16. If a word is read or written k times in a short interval, the computer will need 1 reference to slow memory and $k - 1$ references to fast memory. The larger k is, the better the overall performance.

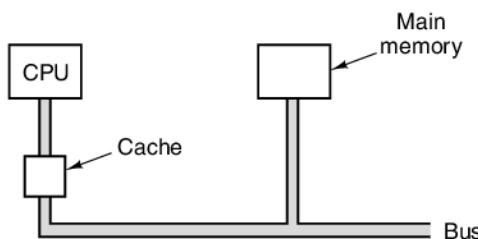


Figure 2-16. The cache is logically between the CPU and main memory. Physically, there are several possible places it could be located.

We can formalize this calculation by introducing c , the cache access time, m , the main memory access time, and h , the **hit ratio**, which is the fraction of all references that can be satisfied out of the cache. In our little example of the previous paragraph, $h = (k - 1)/k$. Some authors also define the **miss ratio**, which is $1 - h$.

With these definitions, we can calculate the mean access time as follows:

$$\text{mean access time} = c + (1 - h)m$$

As $h \rightarrow 1$, all references can be satisfied out of the cache, and the access time approaches c . On the other hand, as $h \rightarrow 0$, a memory reference is needed every time, so the access time approaches $c + m$, first a time c to check the cache (unsuccessfully), and then a time m to do the memory reference. On some systems, the memory reference can be started in parallel with the cache search, so that if a cache miss occurs, the memory cycle has already been started. However, this strategy requires that the memory can be stopped in its tracks on a cache hit, making the implementation more complicated.

Using the locality principle as a guide, main memories and caches are divided up into fixed-size blocks. When talking about these blocks inside the cache, we will often refer to them as **cache lines**. When a cache miss occurs, the entire cache line is loaded from the main memory into the cache, not just the word needed. For example, with a 64-byte line size, a reference to memory address 260 will pull the line consisting of bytes 256 to 319 into one cache line. With a little bit of luck, some of the other words in the cache line will be needed shortly. Operating this way is more efficient than fetching individual words because it is faster to fetch k words all at once than one word k times. Also, having cache entries be more than one word means there are fewer of them, hence a smaller overhead is required. Finally, many computers can transfer 64 or 128 bits in parallel on a single bus cycle, even on 32-bit machines.

Cache design is an increasingly important subject for high-performance CPUs. One issue is cache size. The bigger the cache, the better it performs, but also the slower it is to access and the more it costs. A second issue is the size of the cache line. A 16-KB cache can be divided up into 1024 lines of 16 bytes, 2048 lines of 8 bytes, and other combinations. A third issue is how the cache is organized, that is, how does the cache keep track of which memory words are currently being held? We will examine caches in detail in Chap. 4.

A fourth design issue is whether instructions and data are kept in the same cache or different ones. Having a **unified cache** (instructions and data use the same cache) is a simpler design and automatically balances instruction fetches against data fetches. Nevertheless, the trend these days is toward a **split cache**, with instructions in one cache and data in the other. This design is also called a **Harvard architecture**, the reference going all the way back to Howard Aiken's Mark III computer, which had different memories for instructions and data. The force driving designers in this direction is the widespread use of pipelined CPUs. The instruction fetch unit needs to access instructions at the same time the operand

fetch unit needs access to data. A split cache allows parallel accesses; a unified one does not. Also, since instructions are not modified during execution, the contents of the instruction cache never has to be written back into memory.

Finally, a fifth issue is the number of caches. It is common these days to have chips with a primary cache on chip, a secondary cache off chip but in the same package as the CPU chip, and a third cache still further away.

2.2.6 Memory Packaging and Types

From the early days of semiconductor memory until the early 1990s, memory was manufactured, bought, and installed as single chips. Chip densities went from 1K bits to 1M bits and beyond, but each chip was sold as a separate unit. Early PCs often had empty sockets into which additional memory chips could be plugged, if and when the purchaser needed them.

Since the early 1990s, a different arrangement has been used. A group of chips, typically 8 or 16, is mounted on a printed circuit board and sold as a unit. This unit is called a **SIMM (Single Inline Memory Module)** or a **DIMM (Dual Inline Memory Module)**, depending on whether it has a row of connectors on one side or both sides of the board. SIMMs have one edge connector with 72 contacts and transfer 32 bits per clock cycle. They are rarely used these days. DIMMs usually have edge connectors with 120 contacts on each side of the board, for a total of 240 contacts, and transfer 64 bits per clock cycle. The most common ones at present are DDR3 DIMMS, which is the third version of the double data-rate memories. A typical DIMM is illustrated in Fig. 2-17.

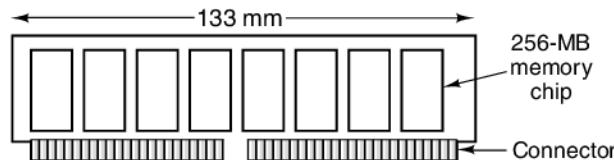


Figure 2-17. Top view of a DIMM holding 4 GB with eight chips of 256 MB on each side. The other side looks the same.

A typical DIMM configuration might have eight data chips with 256 MB each. The entire module would then hold 2 GB. Many computers have room for four modules, giving a total capacity of 8 GB when using 2-GB modules and more when using larger ones.

A physically smaller DIMM, called an **SO-DIMM (Small Outline DIMM)**, is used in notebook computers. DIMMS can have a parity bit or error correction added, but since the average error rate of a module is one error every 10 years, for most garden-variety computers, error detection and correction are omitted.

2.3 SECONDARY MEMORY

No matter how big the main memory is, it is always way too small. People always want to store more information than it can hold, primarily because as technology improves, people begin thinking about storing things that were previously entirely in the realm of science fiction. For example, as the U.S. government's budget discipline forces government agencies to generate their own revenue, one can imagine the Library of Congress deciding to digitize and sell its full contents as a consumer article ("All of human knowledge for only \$299.95"). Roughly 50 million books, each with 1 MB of text and 1 MB of compressed pictures, requires storing 10^{14} bytes or 100 terabytes. Storing all 50,000 movies ever made is also in this general ballpark. This amount of information is not going to fit in main memory, at least not for a few decades.

2.3.1 Memory Hierarchies

The traditional solution to storing a great deal of data is a memory hierarchy, as illustrated in Fig. 2-18. At the top are the CPU registers, which can be accessed at full CPU speed. Next comes the cache memory, which is currently on the order of 32 KB to a few megabytes. Main memory is next, with sizes currently ranging from 1 GB for entry-level systems to hundreds of gigabytes at the high end. After that come solid-state and magnetic disks, the current workhorses for permanent storage. Finally, we have magnetic tape and optical disks for archival storage.

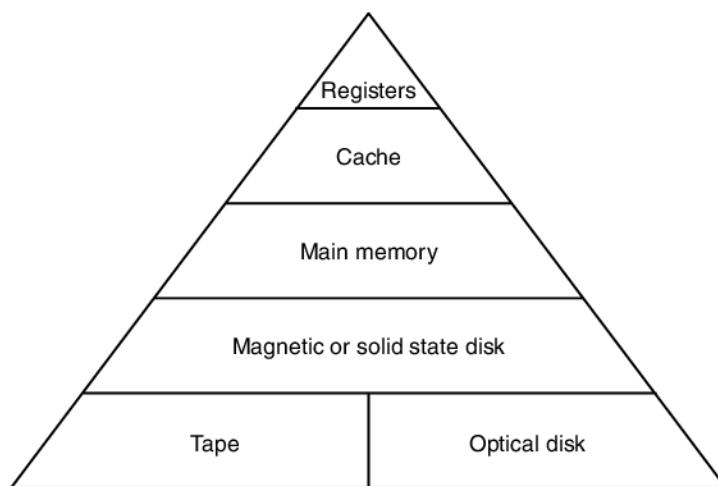


Figure 2-18. A five-level memory hierarchy.

As we move down the hierarchy, three key parameters increase. First, the access time gets bigger. CPU registers can be accessed in a nanosecond or less.

Cache memories take a small multiple of CPU registers. Main memory accesses are typically 10 nanoseconds. Now comes a big gap, as disk access times are at least 10 times slower for solid-state disks and hundreds of times slower for magnetic disks. Tape and optical disk access can be measured in seconds if the media have to be fetched and inserted into a drive.

Second, the storage capacity increases as we go downward. CPU registers are good for perhaps 128 bytes, caches for tens of megabytes, main memories for a few gigabytes, solid-state disks for hundreds of gigabytes, and magnetic disks for terabytes. Tapes and optical disks are usually kept off-line, so their capacity is limited only by the owner's budget.

Third, the number of bits you get per dollar spent increases down the hierarchy. Although the actual prices change rapidly, main memory is measured in dollars/megabyte, solid-state disk in dollars/gigabyte, and magnetic disk and tape storage in pennies/gigabyte.

We have already looked at registers, cache, and main memory. In the following sections we will look at magnetic and solid-state disks; after that, we will study optical ones. We will not study tapes because they are rarely used except for backup, and there is not a lot to say about them anyway.

2.3.2 Magnetic Disks

A magnetic disk consists of one or more aluminum platters with a magnetizable coating. Originally these platters were as much as 50 cm in diameter, but at present they are typically 3 to 9 cm, with disks for notebook computers already under 3 cm and still shrinking. A disk head containing an induction coil floats just over the surface, resting on a cushion of air. When a positive or negative current passes through the head, it magnetizes the surface just beneath the head, aligning the magnetic particles facing left or facing right, depending on the polarity of the drive current. When the head passes over a magnetized area, a positive or negative current is induced in the head, making it possible to read back the previously stored bits. Thus as the platter rotates under the head, a stream of bits can be written and later read back. The geometry of a disk track is shown in Fig. 2-19.

The circular sequence of bits written as the disk makes a complete rotation is called a **track**. Each track is divided up into some number of fixed-length **sectors**, typically containing 512 data bytes, preceded by a **preamble** that allows the head to be synchronized before reading or writing. Following the data is an Error-Correcting Code (ECC), either a Hamming code or, more commonly, a code that can correct multiple errors called a **Reed-Solomon code**. Between consecutive sectors is a small **intersector gap**. Some manufacturers quote their disks' capacities in unformatted state (as if each track contained only data), but a more honest measurement is the formatted capacity, which does not count the preambles, ECCs, and gaps as data. The formatted capacity is typically about 15 percent lower than the unformatted capacity.

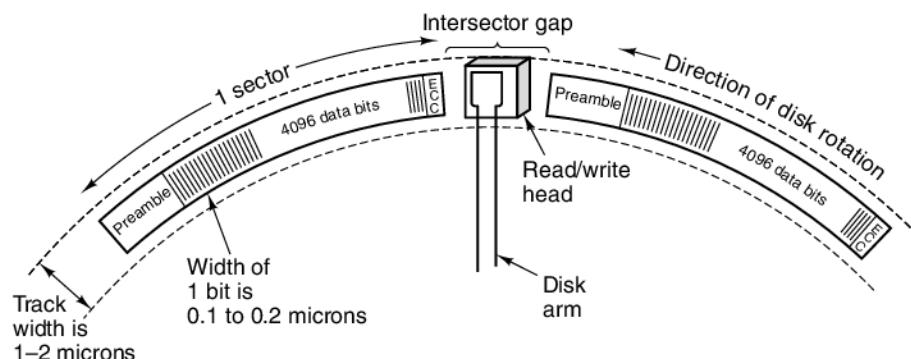


Figure 2-19. A portion of a disk track. Two sectors are illustrated.

All disks have movable arms that are capable of moving in and out to different radial distances from the spindle about which the platter rotates. At each radial distance, a different track can be written. The tracks are thus a series of concentric circles about the spindle. The width of a track depends on how large the head is and how accurately the head can be positioned radially. With current technology, disks have around 50,000 tracks per centimeter, giving track widths in the 200-nanometer range (1 nanometer = 1/1,000,000 mm). It should be noted that a track is not a physical groove in the surface, but simply an annulus (ring) of magnetized material, with small guard areas separating it from the tracks inside and outside it.

The linear bit density around the circumference of the track is different from the radial one. In other words, the number of bits per millimeter measured going around a track is different from the number of bits per millimeter starting from the center and moving outward. The density along a track is determined largely by the purity of the surface and air quality. Current disks achieve densities of 25 gigabits/cm. The radial density is determined by how accurately the arm can be made to seek to a track. Thus a bit is many times larger in the radial direction as compared to the circumference, as suggested by Fig. 2-19.

Ultrahigh density disks utilize a recording technology in which the “long” dimension of the bits is not along the circumference of the disk, but vertically, down into the iron oxide. This technique is called **perpendicular recording**, and it has been demonstrated to provide data densities of up to 100 gigabits/cm. It is likely to become the dominant technology in the coming years.

In order to achieve high surface and air quality, most disks are sealed at the factory to prevent dust from getting in. Such drives were originally called **Winchester disks** because the first such drives (created by IBM) had 30 MB of sealed, fixed storage and 30 MB of removable storage. Supposedly, these 30-30 disks reminded people of the Winchester 30-30 rifles that played a great role in opening the American frontier, and the name “Winchester” stuck. Now they are just called

hard disks to differentiate them from the long-vanished **floppy disks** used on the very first personal computers. In this business, it is difficult to pick a name for anything and not have it be ridiculous 30 years later.

Most disks consist of multiple platters stacked vertically, as depicted in Fig. 2-20. Each surface has its own arm and head. All the arms are ganged together so they move to different radial positions all at once. The set of tracks at a given radial position is called a **cylinder**. Current PC and server disks typically have 1 to 12 platters per drive, giving 2 to 24 recording surfaces. High-end disks can store 1 TB on a single platter and that limit is sure to grow with time.

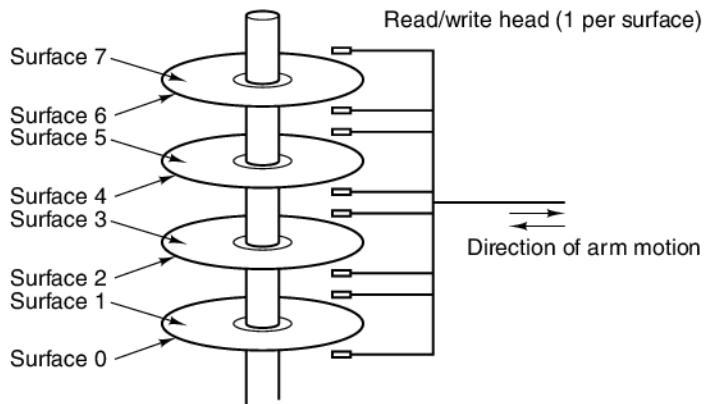


Figure 2-20. A disk with four platters.

Disk performance depends on a variety of factors. To read or write a sector, first the arm must be moved to the right radial position. This action is called a **seek**. Average seek times (between random tracks) range in the 5- to 10-msec range, although seeks between consecutive tracks are now down below 1 msec. Once the head is positioned radially, there is a delay, called the **rotational latency**, until the desired sector rotates under the head. Most disks rotate at 5400 RPM, 7200 RPM, or 10,800 RPM, so the average delay (half a rotation) is 3 to 6 msec. Transfer time depends on the linear density and rotation speed. With typical internal transfer rate of 150 MB/sec, a 512-byte sector takes about 3.5 μ sec. Consequently, the seek time and rotational latency dominate the transfer time. Reading random sectors all over the disk is clearly an inefficient way to operate.

It is worth mentioning that on account of the preambles, the ECCs, the inter-sector gaps, the seek times, and the rotational latencies, there is a big difference between a drive's maximum burst rate and its maximum sustained rate. The maximum burst rate is the data rate once the head is over the first data bit. The computer must be able to handle data coming in this fast. However, the drive can keep up that rate only for one sector. For some applications, such as multimedia, what

matters is the average sustained rate over a period of seconds, which has to take into account the necessary seeks and rotational delays as well.

A little thought and the use of that old high-school math formula for the circumference of a circle, $c = 2\pi r$, will reveal that the outer tracks have more linear distance around them than the inner ones do. Since all magnetic disks rotate at a constant angular velocity, no matter where the heads are, this observation creates a problem. In older drives, manufacturers used the maximum possible linear density on the innermost track, and successively lowered linear bit densities on tracks further out. If a disk had 18 sectors per track, for example, each one occupied 20 degrees of arc, no matter which cylinder it was in.

Nowadays, a different strategy is used. Cylinders are divided into zones (typically 10 to 30 per drive), and the number of sectors per track is increased in each zone moving outward from the innermost track. This change makes keeping track of information harder but increases the drive capacity, which is viewed as more important. All sectors are the same size. A disk with five zones is shown in Fig. 2-21.

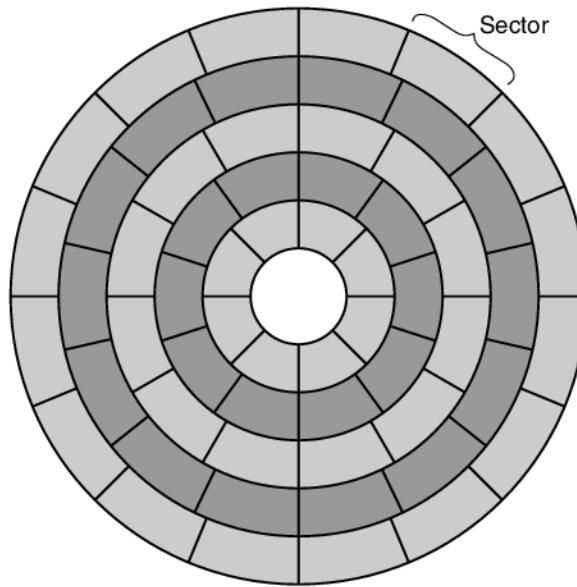


Figure 2-21. A disk with five zones. Each zone has many tracks.

Associated with each drive is a **disk controller**, a chip that controls the drive. Some controllers contain a full CPU. The controller's tasks include accepting commands from the software, such as READ, WRITE, and FORMAT (writing all the preambles), controlling the arm motion, detecting and correcting errors, and converting 8-bit bytes read from memory into a serial bit stream and vice versa. Some controllers also handle buffering of multiple sectors, caching sectors read for

potential future use, and remapping bad sectors. This latter function is caused by the existence of sectors with a bad (permanently magnetized) spot. When the controller discovers a bad sector, it replaces it by one of the spare sectors reserved for this purpose within each cylinder or zone.

2.3.3 IDE Disks

Modern personal computer disks evolved from the one in the IBM PC XT, which was a 10-MB Seagate disk controlled by a Xebec disk controller on a plug-in card. The Seagate disk had 4 heads, 306 cylinders, and 17 sectors/track. The controller was capable of handling two drives. The operating system read from and wrote to a disk by putting parameters in CPU registers and then calling the **BIOS (Basic Input Output System)**, located in the PC's built-in read-only memory. The BIOS issued the machine instructions to load the disk controller registers that initiated transfers.

The technology evolved rapidly from having the controller on a separate board, to having it closely integrated with the drives, starting with **IDE (Integrated Drive Electronics)** drives in the mid 1980s. However, the BIOS calling conventions were not changed for reasons of backward compatibility. These calling conventions addressed sectors by giving their head, cylinder, and sector numbers, with the heads and cylinders numbered starting at 0 and the sectors starting at 1. This choice was probably due to a mistake on the part of the original BIOS programmer, who wrote his masterpiece in 8088 assembler. With 4 bits for the head, 6 bits for the sector, and 10 bits for the cylinder, the maximum drive could have 16 heads, 63 sectors, and 1024 cylinders, for a total of 1,032,192 sectors. Such a maximum drive has a capacity of 504 MB, which probably seemed like infinity at the time but certainly does not today. (Would you fault a new machine today that could not handle drives bigger than 1000 TB?)

Unfortunately, before too long, drives below 504 MB appeared but with the wrong geometry (e.g., 4 heads, 32 sectors, 2000 cylinders is 256,000 sectors). There was no way for the operating system to address them due to the long-frozen BIOS calling conventions. As a result, disk controllers began to lie, pretending that the geometry was within the BIOS limits but actually remapping the virtual geometry onto the real geometry. Although this approach worked, it wreaked havoc with operating systems that carefully placed data to minimize seek times.

Eventually, IDE drives evolved into **EIDE** drives (**Extended IDE**), which also support a second addressing scheme called **LBA (Logical Block Addressing)**, which just numbers the sectors starting at 0 up until a maximum of $2^{28} - 1$. This scheme requires the controller to convert LBA addresses to head, sector, and cylinder addresses, but at least it does get beyond the 504-MB limit. Unfortunately, it created a new bottleneck at $2^{28} \times 2^9$ bytes (128 GB). In 1994, when the EIDE standard was adopted, nobody in their wildest imagination could imagine 128-GB

disks. Standards committees, like politicians, have a tendency to push problems forward in time so the next committee has to solve them.

EIDE drives and controllers also had other improvements as well. For example, EIDE controllers could have two channels, each with a primary and a secondary drive. This arrangement allowed a maximum of four drives per controller. CD-ROM and DVD drives were also supported, and the transfer rate was increased from 4 MB/sec to 16.67 MB/sec.

As disk technology continued to improve, the EIDE standard continued to evolve, but for some reason the successor to EIDE was called **ATA-3 (AT Attachment)**, a reference to the IBM PC/AT (where AT referred to the then-Advanced Technology of a 16-bit CPU running at 8 MHz). In the next edition, the standard was called **ATAPI-4 (ATA Packet Interface)** and the speed was increased to 33 MB/sec. In ATAPI-5 it went to 66 MB/sec.

By this time, the 128-GB limit imposed by the 28-bit LBA addresses was looming larger and larger, so ATAPI-6 changed the LBA size to 48 bits. The new standard will run into trouble when disks reach $2^{48} \times 2^9$ bytes (128 PB). With a 50% annual increase in capacity, the 48-bit limit will probably last until about 2035. To find out how the problem was solved, please consult the 11th edition of this book. The smart money is betting on increasing the LBA size to 64 bits. The ATAPI-6 standard also increased the transfer rate to 100 MB/sec and addressed the issue of disk noise for the first time.

The ATAPI-7 standard is a radical break with the past. Instead of increasing the size of the drive connector (to increase the data rate), this standard uses what is called **serial ATA** to transfer 1 bit at a time over a 7-pin connector at speeds starting at 150 MB/sec and expected to rise over time to 1.5 GB/sec. Replacing the old 80-wire flat cable with a round cable only a few mm thick improves airflow within the computer. Also, serial ATA uses 0.5 volts for signaling (compared to 5 volts on ATAPI-6 drives), which reduces power consumption. It is likely that within a few years, all computers will use serial ATA. The issue of power consumption by disks is an increasingly important one, both at the high end, where data centers have vast disk farms, and at the low end, where notebooks are power limited (Gurumurthi et al., 2003).

2.3.4 SCSI Disks

SCSI disks are not different from IDE disks in terms of how their cylinders, tracks, and sectors are organized, but they have a different interface and much higher transfer rates. SCSI traces its history back to Howard Shugart, the inventor of the floppy disk, which was used on the first personal computers in the 1980s. His company introduced the SASI (Shugart Associates System Interface) disk in 1979. After some modification and quite a bit of discussion, ANSI standardized it in 1986 and changed the name to **SCSI (Small Computer System Interface)**. SCSI is pronounced “scuzzy.” Since then, increasingly higher bandwidth versions

have been standardized under the names Fast SCSI (10 MHz), Ultra SCSI (20 MHz), Ultra2 SCSI (40 MHz), Ultra3 SCSI (80 MHz), Ultra4 SCSI (160 MHz), and Ultra5 SCSI (320 MHz). Each of these has a wide (16-bit) version as well. In fact, the recent ones have only a wide version. The main combinations are shown in Fig. 2-22.

Name	Data bits	Bus MHz	MB/sec
SCSI-1	8	5	5
Fast SCSI	8	10	10
Wide Fast SCSI	16	10	20
Ultra SCSI	8	20	20
Wide Ultra SCSI	16	20	40
Ultra2 SCSI	8	40	40
Wide Ultra2 SCSI	16	40	80
Wide Ultra3 SCSI	16	80	160
Wide Ultra4 SCSI	16	160	320
Wide Ultra5 SCSI	16	320	640

Figure 2-22. Some of the possible SCSI parameters.

Because SCSI disks have high transfer rates, they are the standard disk in many high-end workstations and servers, especially those that run RAID configurations (see below).

SCSI is more than just a hard-disk interface. It is a bus to which a SCSI controller and up to seven devices can be attached. These can include one or more SCSI hard disks, CD-ROMs, CD recorders, scanners, tape units, and other SCSI peripherals. Each SCSI device has a unique ID, from 0 to 7 (15 for wide SCSI). Each device has two connectors: one for input and one for output. Cables connect the output of one device to the input of the next one, in series, like a string of cheap Christmas tree lamps. The last device in the string must be terminated to prevent reflections from the ends of the SCSI bus from interfering with other data on the bus. Typically, the controller is on a plug-in card and the start of the cable chain, although this configuration is not strictly required by the standard.

The most common cable for 8-bit SCSI has 50 wires, 25 of which are grounds paired one-to-one with the other 25 wires to provide the excellent noise immunity needed for high-speed operation. Of the 25 wires, 8 are for data, 1 is for parity, 9 are for control, and the remainder are for power or are reserved for future use. The 16-bit (and 32-bit) devices need a second cable for the additional signals. The cables may be several meters long, allowing for external drives, scanners, etc.

SCSI controllers and peripherals can operate either as initiators or as targets. Usually, the controller, acting as initiator, issues commands to disks and other peripherals acting as targets. These commands are blocks of up to 16 bytes telling the target what to do. Commands and responses occur in phases, using various

control signals to delineate the phases and arbitrate bus access when multiple devices are trying to use the bus at the same time. This arbitration is important because SCSI allows all the devices to run at once, potentially greatly improving performance in an environment with multiple processes active at once. IDE and EIDE allow only one active device at a time.

2.3.5 RAID

CPU performance has been increasing exponentially over the past decade, roughly doubling every 18 months. Not so with disk performance. In the 1970s, average seek times on minicomputer disks were 50 to 100 msec. Now seek times are 10 msec. In most technical industries (say, automobiles or aviation), a factor of 5 to 10 performance improvement in two decades would be major news, but in the computer industry it is an embarrassment. Thus the gap between CPU performance and disk performance has become much larger over time.

As we have seen, parallel processing is often used to speed up CPU performance. It has occurred to various people over the years that parallel I/O might be a good idea, too. In their 1988 paper, Patterson et al. suggested six specific disk organizations that could be used to improve disk performance, reliability, or both (Patterson et al., 1988). These ideas were quickly adopted by industry and have led to a new class of I/O device called a **RAID**. Patterson et al. defined **RAID** as **Redundant Array of Inexpensive Disks**, but industry redefined the I to be “Independent” rather than “Inexpensive” (maybe so they could use expensive disks?). Since a villain was also needed (as in RISC versus CISC, also due to Patterson), the bad guy here was the **SLED (Single Large Expensive Disk)**.

The idea behind a RAID is to install a box full of disks next to the computer, typically a large server, replace the disk controller card with a RAID controller, copy the data over to the RAID, and then continue normal operation. In other words, a RAID should look like a SLED to the operating system but have better performance and better reliability. Since SCSI disks have good performance, low price, and the ability to have up to 7 drives on a single controller (15 for wide SCSI), it is natural that many RAIDs consist of a RAID SCSI controller plus a box of SCSI disks that appear to the operating system as a single large disk. In this way, no software changes are required to use the RAID, a big selling point for many system administrators.

In addition to appearing like a single disk to the software, all RAIDs have the property that the data are distributed over the drives, to allow parallel operation. Several different schemes for doing this were defined by Patterson et al., and they are now known as RAID level 0 through RAID level 5. In addition, there are a few other minor levels that we will not discuss. The term “level” is something of a misnomer since there is no hierarchy involved; there are simply six different organizations, each with a different mix of reliability and performance characteristics.

RAID level 0 is illustrated in Fig. 2-23(a). It consists of viewing the virtual disk simulated by the RAID as being divided up into strips of k sectors each, with sectors 0 to $k - 1$ being strip 0, sectors k to $2k - 1$ as strip 1, and so on. For $k = 1$, each strip is a sector; for $k = 2$ a strip is two sectors, etc. The RAID level 0 organization writes consecutive strips over the drives in round-robin fashion, as depicted in Fig. 2-23(a) for a RAID with four disk drives. Distributing data over multiple drives like this is called **striping**. For example, if the software issues a command to read a data block consisting of four consecutive strips starting at a strip boundary, the RAID controller will break this command up into four separate commands, one for each of the four disks, and have them operate in parallel. Thus we have parallel I/O without the software knowing about it.

RAID level 0 works best with large requests, the bigger the better. If a request is larger than the number of drives times the strip size, some drives will get multiple requests, so that when they finish the first request they start the second one. It is up to the controller to split the request up and feed the proper commands to the proper disks in the right sequence and then assemble the results in memory correctly. Performance is excellent and the implementation is straightforward.

RAID level 0 works worst with operating systems that habitually ask for data one sector at a time. The results will be correct, but there is no parallelism and hence no performance gain. Another disadvantage of this organization is that the reliability is potentially worse than having a SLED. If a RAID consists of four disks, each with a mean time to failure of 20,000 hours, about once every 5000 hours a drive will fail and all the data will be completely lost. A SLED with a mean time to failure of 20,000 hours would be four times more reliable. Because no redundancy is present in this design, it is not really a true RAID.

The next option, RAID level 1, shown in Fig. 2-23(b), is a true RAID. It duplicates all the disks, so there are four primary disks and four backup disks in this example, although any other even number of disks is also possible. On a write, every strip is written twice. On a read, either copy can be used, distributing the load over more drives. Consequently, write performance is no better than for a single drive, but read performance can be up to twice as good. Fault tolerance is excellent: if a drive crashes, the copy is simply used instead. Recovery consists of simply installing a new drive and copying the entire backup drive to it.

Unlike levels 0 and 1, which work with strips of sectors, RAID level 2 works on a word basis, possibly even a byte basis. Imagine splitting each byte of the single virtual disk into a pair of 4-bit nibbles, then adding a Hamming code to each one to form a 7-bit word, of which bits 1, 2, and 4 were parity bits. Further imagine that the seven drives of Fig. 2-23(c) were synchronized in terms of arm position and rotational position. Then it would be possible to write the 7-bit Hamming coded word over the seven drives, one bit per drive.

The Thinking Machines CM-2 computer used this scheme, taking 32-bit data words and adding 6 parity bits to form a 38-bit Hamming word, plus an extra bit for word parity, and spread each word over 39 disk drives. The total throughput

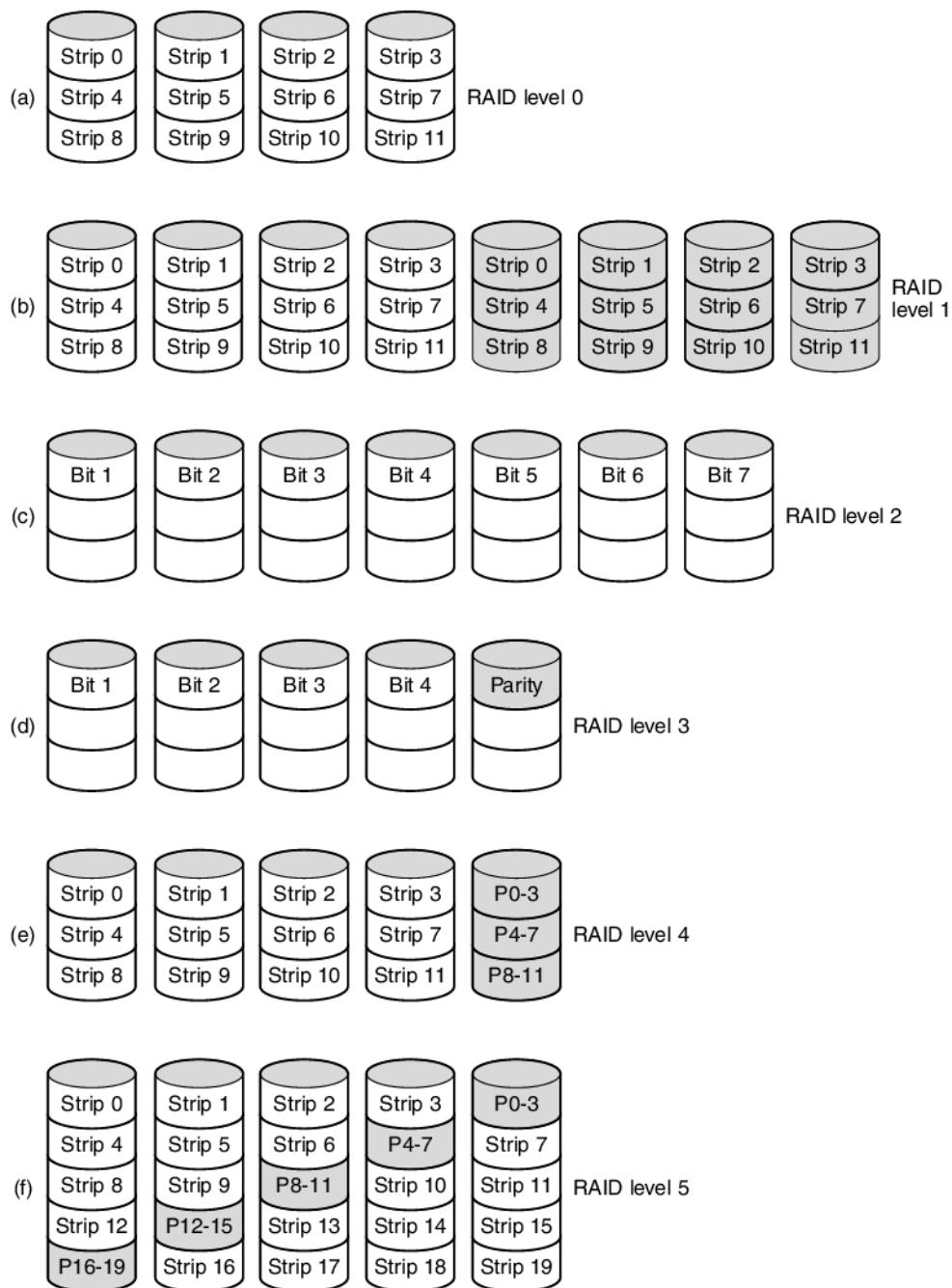


Figure 2-23. RAID levels 0 through 5. Backup and parity drives are shown shaded.

was immense, because in one sector time it could write 32 sectors worth of data. Also, losing one drive did not cause problems, because loss of a drive amounted to losing 1 bit in each 39-bit word read, something the Hamming code could handle on the fly.

On the down side, this scheme requires all the drives to be rotationally synchronized, and it makes sense only with a substantial number of drives (even with 32 data drives and 6 parity drives, the overhead is 19 percent). It also asks a lot of the controller, since it must do a Hamming checksum every bit time.

RAID level 3 is a simplified version of RAID level 2. It is illustrated in Fig. 2-23(d). Here a single parity bit is computed for each data word and written to a parity drive. As in RAID level 2, the drives must be exactly synchronized, since individual data words are spread over multiple drives.

At first thought, it might appear that a single parity bit gives only error detection, not error correction. For the case of random undetected errors, this observation is true. However, for the case of a drive crashing, it provides full 1-bit error correction since the position of the bad bit is known. If a drive crashes, the controller just pretends that all its bits are 0s. If a word has a parity error, the bit from the dead drive must have been a 1, so it is corrected. Although both RAID levels 2 and 3 offer very high data rates, the number of separate I/O requests per second they can handle is no better than for a single drive.

RAID levels 4 and 5 work with strips again, not individual words with parity, and do not require synchronized drives. RAID level 4 [see Fig. 2-23(e)] is like RAID level 0, with a strip-for-strip parity written onto an extra drive. For example, if each strip is k bytes long, all the strips are EXCLUSIVE ORed together, resulting in a parity strip k bytes long. If a drive crashes, the lost bytes can be recomputed from the parity drive.

This design protects against the loss of a drive but performs poorly for small updates. If one sector is changed, all the drives must be read in order to recalculate the parity, which then must be rewritten. Alternatively, the old user data and the old parity data can be read and the new parity recomputed from them. Even with this optimization, a small update requires two reads and two writes, clearly a bad arrangement.

As a consequence of the heavy load on the parity drive, it may become a bottleneck. This bottleneck is eliminated in RAID level 5 by distributing the parity bits uniformly over all the drives, round robin fashion, as shown in Fig. 2-23(f). However, in the event of a drive crash, reconstructing the contents of the failed drive is a complex process.

2.3.6 Solid-State Disks

Disks made from nonvolatile flash memory, often called **solid-state disks (SSDs)**, are growing in popularity as a high-speed alternative to traditional magnetic disk technologies. The invention of the SSD is a classic tale of “When they

give you lemons, make lemonade.” While modern electronics may seem totally reliable, the reality is that transistors slowly wear out as they are used. Every time they switch, they wear out a little bit more and get closer to no longer working. One likely way that a transistor will fail is due to “hot carrier injection,” a failure mechanism in which an electron charge gets embedded inside a once-working transistor, leaving it in a state where it is permanently stuck on or off. While generally thought of as a death sentence for a (likely) innocent transistor, Fujio Masuoka while working for Toshiba discovered a way to harness this failure mechanism to create a new nonvolatile memory. In the early 1980s, he invented the first flash memory.

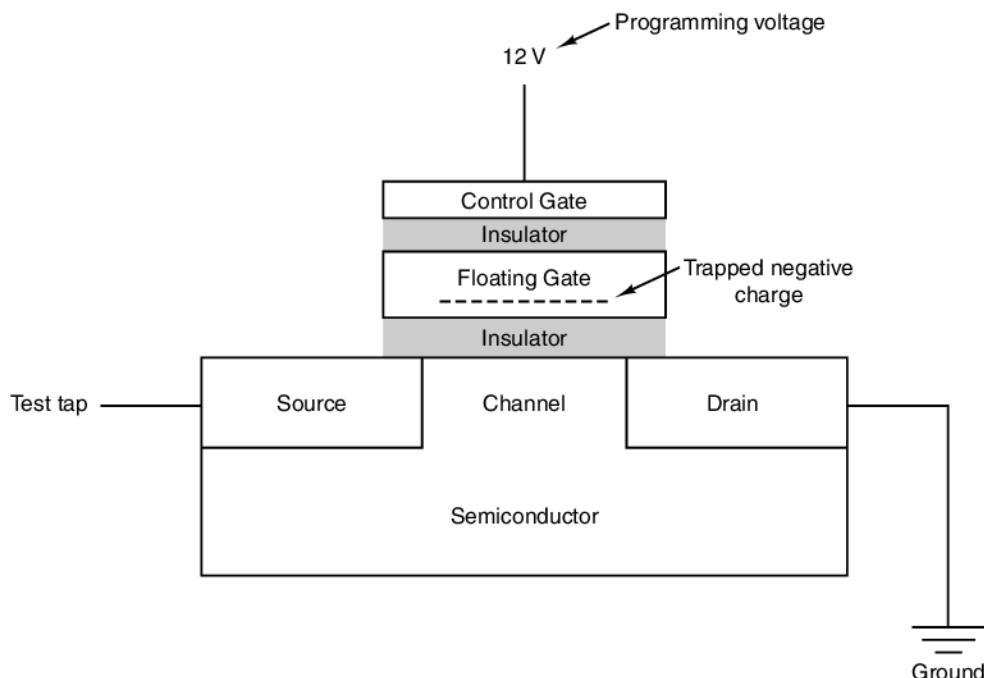


Figure 2-24. A flash memory cell.

Flash disks are made of many solid-state flash memory cells. The flash memory cells are made from a single special flash transistor. A flash memory cell is shown in Fig. 2-24. Embedded inside the transistor is a floating gate that can be charged and discharged using high voltages. Before being programmed, the floating gate does not affect the operation of the transistor, essentially acting as an extra insulator between the control gate and the transistor channel. If the flash cell is tested, it will act like a simple transistor.

To program the flash bit cell, a high voltage (in the computer world 12 V is a high voltage) is applied to the control gate, which accelerates the process of hot

carrier injection into the floating gate. Electrons become embedded into the floating gate, which places a negative charge internal to the flash transistor. The embedded negative charge increases the voltage necessary to turn on the flash transistor, and by testing whether or not the channel turns on with a high or low voltage, it is possible to determine whether the floating gate is charged or not, resulting in a 0 or 1 value for the flash cell. The embedded charge remains in the transistor, even if power is removed from the system, making the flash memory cell nonvolatile.

Because SSDs are essentially memory, they have superior performance to spinning disks and have zero seek time. While a typical magnetic disk can access data up to 100 MB/sec, a SSD can operate two to three times faster. And because the device has no moving parts, it is particularly suited for use in notebook computers, where jarring and movement will not affect its ability to access data. The downside of SSDs, compared to magnetic disks, is their cost. While magnetic disks cost pennies/gigabyte, a typical SSD will cost one to three dollars/gigabyte, making their use appropriate only for smaller drive applications or situations that are not cost sensitive. The cost of SSDs is dropping, but they still have a long way to go to catch up to cheap magnetic disks. So while SSDs are replacing magnetic disks in many computers, it will likely be a long time before the magnetic disk goes the way of the dinosaur (unless another big meteorite strikes the earth, in which cases the SSDs are probably not going to survive either).

Another disadvantage of SSDs compared with magnetic disks is their failure rate. A typical flash cell can be written only about 100,000 times before it will no longer function. The process of injecting electrons into the floating gate slowly damages it and the surrounding insulators, until it can no longer function. To increase the lifetime of SSDs, a technique called **wear leveling** is used to spread writes out to all flash cells in the disk. Every time a new disk block is written, the destination block is reassigned to a new SSD block that has not been recently written. This requires the use of a logical block map inside the flash drive, which is one of the reasons that flash drives have high internal storage overheads. Using wear leveling, a flash drive can support a number of writes equal to the number of writes a cell can sustain times the number of blocks on the disk.

Some SSDs are able to encode multiple bits per byte using multilevel flash cells. The technology carefully controls the amount of charge placed into the floating gate. An increasing sequence of voltages is then applied to the control gate to determine how much charge is stored in the floating gate. Typical multilevel cells will support four charge levels, yielding two bits per flash cell.

2.3.7 CD-ROMs

Optical disks were originally developed for recording television programs, but they can be put to more esthetic use as computer storage devices. Due to their large capacity and low price optical disks are widely used for distributing software, books, movies, and data of all kinds, as well as making backups of hard disks.

First-generation optical disks were invented by the Dutch electronics conglomerate Philips for holding movies. They were 30 cm across and marketed under the name LaserVision, but they did not catch on, except in Japan.

In 1980, Philips, together with Sony, developed the CD (Compact Disc), which rapidly replaced the 33 1/3 RPM vinyl record for music. The precise technical details for the CD were published in an official International Standard (IS 10149), popularly called the **Red Book**, after to the color of its cover. (International Standards are issued by the International Organization for Standardization, which is the international counterpart of national standards groups like ANSI, DIN, etc. Each one has an IS number.) The point of publishing the disk and drive specifications as an International Standard is to allow CDs from different music publishers and players from different electronics manufacturers to work together. All CDs are 120 mm across and 1.2 mm thick, with a 15-mm hole in the middle. The audio CD was the first successful mass-market digital storage medium. Audio CDs are supposed to last 100 years. Please check back in 2080 for an update on how well the first batch did.

A CD is prepared by using a high-power infrared laser to burn 0.8-micron diameter holes in a coated glass master disk. From this master, a mold is made, with bumps where the laser holes were. Into this mold, molten polycarbonate is injected to form a CD with the same pattern of holes as the glass master. Then a thin layer of reflective aluminum is deposited on the polycarbonate, topped by a protective lacquer and finally a label. The depressions in the polycarbonate substrate are called **pits**; the unburned areas between the pits are called **lands**.

When a CD is played back, a low-power laser diode shines infrared light with a wavelength of 0.78 micron on the pits and lands as they stream by. The laser is on the polycarbonate side, so the pits stick out in the direction of the laser as bumps in the otherwise flat surface. Because the pits have a height of one-quarter the wavelength of the laser light, light reflecting off a pit is half a wavelength out of phase with light reflecting off the surrounding surface. As a result, the two parts interfere destructively and return less light to the player's photodetector than light bouncing off a land. This is how the player tells a pit from a land. Although it might seem simplest to use a pit to record a 0 and a land to record a 1, it is more reliable to use a pit/land or land/pit transition for a 1 and its absence as a 0, so this scheme is used.

The pits and lands are written in a single continuous spiral starting near the hole and working out a distance of 32 mm toward the edge. The spiral makes 22,188 revolutions around the disk (about 600 per mm). If unwound, it would be 5.6 km long. The spiral is illustrated in Fig. 2-25.

To make the music play at a uniform rate, it is necessary for the pits and lands to stream by at a constant *linear* velocity. Consequently, the rotation rate of the CD must be continuously reduced as the reading head moves from the inside of the CD to the outside. At the inside, the rotation rate is 530 RPM to achieve the desired streaming rate of 120 cm/sec; at the outside it has to drop to 200 RPM to give

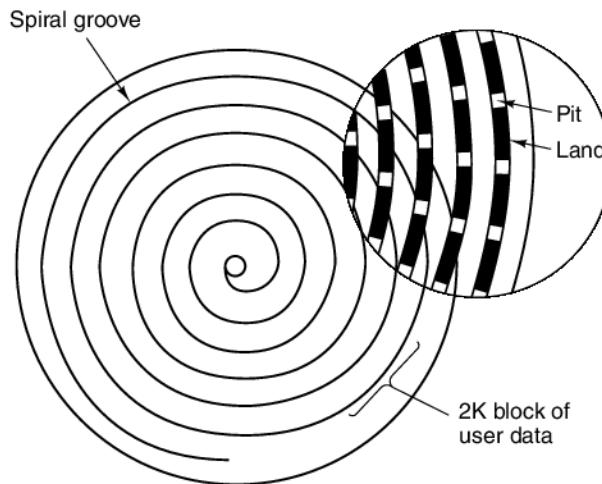


Figure 2-25. Recording structure of a Compact Disc or CD-ROM.

the same linear velocity at the head. A constant-linear-velocity drive is quite different than a magnetic-disk drive, which operates at a constant *angular* velocity, independent of where the head is currently positioned. Also, 530 RPM is a far cry from the 3600 to 7200 RPM that most magnetic disks whirl at.

In 1984, Philips and Sony realized the potential for using CDs to store computer data, so they published the **Yellow Book** defining a precise standard for what are now called **CD-ROMs (Compact Disc-Read Only Memory)**. To piggyback on the by-then already substantial audio CD market, CD-ROMs were to be the same physical size as audio CDs, mechanically and optically compatible with them, and produced using the same polycarbonate injection molding machines. The consequences of this decision were that slow variable-speed motors were required, but also that the manufacturing cost of a CD-ROM would be well under one dollar in moderate volume.

What the Yellow Book defined was the formatting of the computer data. It also improved the error-correcting abilities of the system, an essential step because although music lovers do not mind losing a bit here and there, computer lovers tend to be Very Picky about that. The basic format of a CD-ROM consists of encoding every byte in a 14-bit symbol. As we saw above, 14 bits is enough to Hamming encode an 8-bit byte with 2 bits left over. In fact, a more powerful encoding system is used. The 14-to-8 mapping for reading is done in hardware by table lookup.

At the next level up, a group of 42 consecutive symbols forms a 588-bit **frame**. Each frame holds 192 data bits (24 bytes). The remaining 396 bits are for error correction and control. This scheme is identical for audio CDs and CD-ROMs.

What the Yellow Book adds is the grouping of 98 frames into a **CD-ROM sector**, as shown in Fig. 2-26. Every CD-ROM sector begins with a 16-byte preamble, the first 12 of which are 00FFFFFFFFFFFFF00 (hexadecimal), to allow the player to recognize the start of a CD-ROM sector. The next 3 bytes contain the sector number, needed because seeking on a CD-ROM with its single data spiral is much more difficult than on a magnetic disk with its uniform concentric tracks. To seek, the software in the drive calculates approximately where to go, moves the head there, and then starts hunting around for a preamble to see how good its guess was. The last byte of the preamble is the mode.

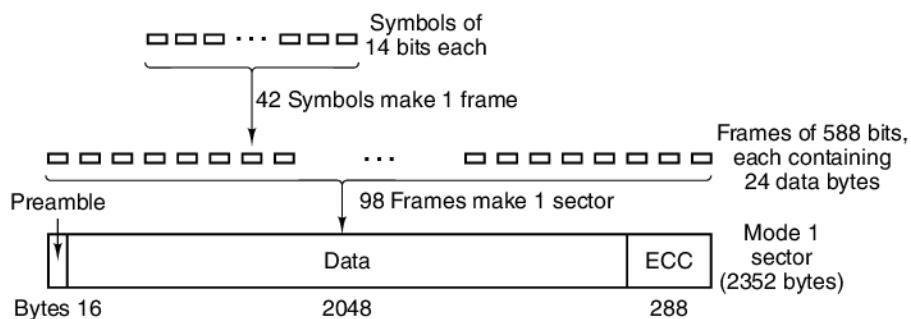


Figure 2-26. Logical data layout on a CD-ROM.

The Yellow Book defines two modes. Mode 1 uses the layout of Fig. 2-26, with a 16-byte preamble, 2048 data bytes, and a 288-byte error-correcting code (a cross-interleaved Reed-Solomon code). Mode 2 combines the data and ECC fields into a 2336-byte data field for those applications that do not need (or cannot afford the time to perform) error correction, such as audio and video. Note that to provide excellent reliability, three separate error-correcting schemes are used: within a symbol, within a frame, and within a CD-ROM sector. Single-bit errors are corrected at the lowest level, short burst errors are corrected at the frame level, and any residual errors are caught at the sector level. The price paid for this reliability is that it takes 98 frames of 588 bits (7203 bytes) to carry a single 2048-byte payload, an efficiency of only 28 percent.

Single-speed CD-ROM drives operate at 75 sectors/sec, which gives a data rate of 153,600 bytes/sec in mode 1 and 175,200 bytes/sec in mode 2. Double-speed drives are twice as fast, and so on up to the highest speed. A standard audio CD has room for 74 minutes of music, which, if used for mode 1 data, gives a capacity of 681,984,000 bytes. This figure is usually reported as 650 MB because 1 MB is 2^{20} bytes (1,048,576 bytes), not 1,000,000 bytes.

As usual, whenever a new technology comes out, some people try to push the envelope. When designing the CD-ROM, Philips and Sony were cautious and had the writing process stop well before the outer edge of the disc was reached. It did

not take long before some drive manufacturers allowed their drives to go beyond the official limit and come perilously close to the physical edge of the medium, giving about 700 MB instead of 650 MB. But as the technology improved and the blank discs were manufactured to a higher standard, 703.12 MB (360,000 2048-byte sectors instead of 333,000 sectors) became the new norm.

Note that even a 32x CD-ROM drive (4,915,200 bytes/sec) is no match for a fast SCSI-2 magnetic-disk drive at 10 MB/sec. When you realize that the seek time is often several hundred milliseconds, it should be clear that CD-ROM drives are not at all in the same performance category as magnetic-disk drives, despite their large capacity.

In 1986, Philips struck again with the **Green Book**, adding graphics and the ability to interleave audio, video, and data in the same sector, a feature essential for multimedia CD-ROMs.

The last piece of the CD-ROM puzzle is the file system. To make it possible to use the same CD-ROM on different computers, agreement was needed on CD-ROM file systems. To get this agreement, representatives of many computer companies met at Lake Tahoe in the High Sierras on the California-Nevada boundary and devised a file system that they called **High Sierra**. It later evolved into an International Standard (IS 9660). It has three levels. Level 1 uses file names of up to 8 characters optionally followed by an extension of up to 3 characters (the MS-DOS file naming convention). File names may contain only uppercase letters, digits, and the underscore. Directories may be nested up to eight deep, but directory names may not contain extensions. Level 1 requires all files to be contiguous, which is not a problem on a medium written only once. Any CD-ROM conformant to IS 9660 level 1 can be read using MS-DOS, an Apple computer, a UNIX computer, or just about any other computer. CD-ROM publishers regard this property as a big plus.

IS 9660 level 2 allows names up to 32 characters, and level 3 allows noncontiguous files. The Rock Ridge extensions (whimsically named after the town in the Mel Brooks film *Blazing Saddles*) allow very long names (for UNIX), UIDs, GIDs, and symbolic links, but CD-ROMs not conforming to level 1 will not be readable on old computers.

2.3.8 CD-Recordables

Initially, the equipment needed to produce a master CD-ROM (or audio CD, for that matter) was extremely expensive. But in the computer industry nothing stays expensive for long. By the mid 1990s, CD recorders no bigger than a CD player were a common peripheral available in most computer stores. These devices were still different from magnetic disks because once written, CD-ROMs could not be erased. Nevertheless, they quickly found a niche as a backup medium for large magnetic hard disks and also allowed individuals or startup companies to

manufacture their own small-run CD-ROMs (hundreds, not thousands) or make masters for delivery to high-volume commercial CD duplication plants. These drives are known as **CD-Rs (CD-Recordables)**.

Physically, CD-Rs start with 120-mm polycarbonate blanks that are like CD-ROMs, except that they contain a 0.6-mm-wide groove to guide the laser for writing. The groove has a sinusoidal excursion of 0.3 mm at a frequency of exactly 22.05 kHz to provide continuous feedback so the rotation speed can be accurately monitored and adjusted if need be. The first CD-Rs looked like regular CD-ROMs, except that they were colored gold on top instead of silver. The gold color came from the use of real gold instead of aluminum for the reflective layer. Unlike silver CDs, which have physical depressions, on CD-Rs the differing reflectivity of pits and lands has to be simulated. This is done by adding a layer of dye between the polycarbonate and the reflective layer, as shown in Fig. 2-27. Two kinds of dye are used: cyanine, which is green, and phthalocyanine, which is a yellowish orange. Chemists can argue endlessly about which one is better. Eventually, an aluminum reflective layer replaced the gold one.

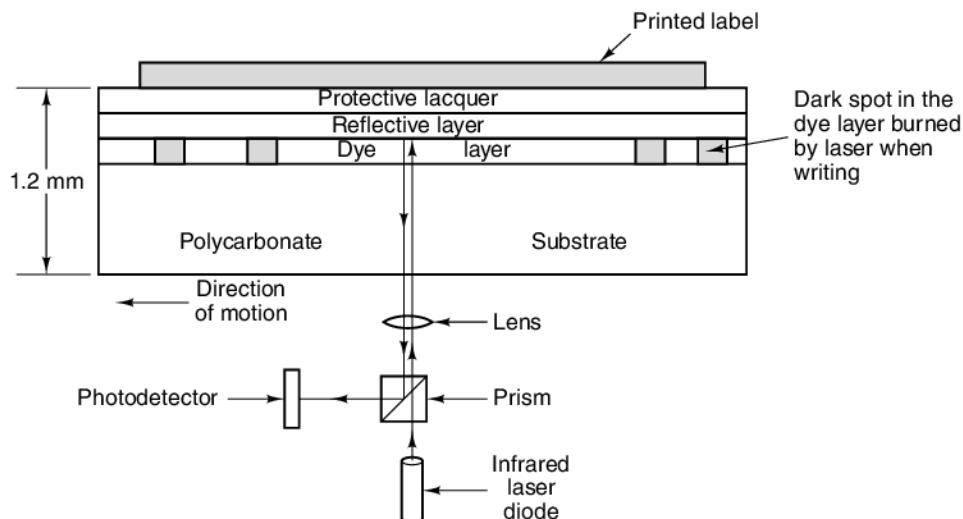


Figure 2-27. Cross section of a CD-R disk and laser (not to scale). A CD-ROM has a similar structure, except without the dye layer and with a pitted aluminum layer instead of a reflective layer.

In its initial state, the dye layer is transparent and lets the laser light pass through and reflect off the reflective layer. To write, the CD-R laser is turned up to high power (8–16 mW). When the beam hits a spot of dye, it heats up, breaking a chemical bond. This change to the molecular structure creates a dark spot. When read back (at 0.5 mW), the photodetector sees a difference between the dark spots where the dye has been hit and transparent areas where it is intact. This difference

is interpreted as the difference between pits and lands, even when read back on a regular CD-ROM reader or even on an audio CD player.

No new kind of CD could hold up its head with pride without a colored book, so CD-R has the **Orange Book**, published in 1989. This document defines CD-R and also a new format, **CD-ROM XA**, which allows CD-Rs to be written incrementally, a few sectors today, a few tomorrow, and a few next month. A group of consecutive sectors written at once is called a **CD-ROM track**.

One of the first uses of CD-R was for the Kodak PhotoCD. In this system the customer brings a roll of exposed film and his old PhotoCD to the photo processor and gets back the same PhotoCD with the new pictures added after the old ones. The new batch, which is created by scanning in the negatives, is written onto the PhotoCD as a separate CD-ROM track. Incremental writing is needed because the CD-R blanks are too expensive to provide a new one for every film roll.

However, incremental writing creates a new problem. Prior to the Orange Book, all CD-ROMs had a single **VTOC (Volume Table of Contents)** at the start. That scheme does not work with incremental (i.e., multitrack) writes. The Orange Book's solution is to give each CD-ROM track its own VTOC. The files listed in the VTOC can include some or all of the files from previous tracks. After the CD-R is inserted into the drive, the operating system searches through all the CD-ROM tracks to locate the most recent VTOC, which gives the current status of the disk. By including some, but not all, of the files from previous tracks in the current VTOC, it is possible to give the illusion that files have been deleted. Tracks can be grouped into **sessions**, leading to **multisession** CD-ROMs. Standard audio CD players cannot handle multisession CDs since they expect a single VTOC at the start.

CD-R makes it possible for individuals and companies to easily copy CD-ROMs (and audio CDs), possibly in violation of the publisher's copyright. Several schemes have been devised to make such piracy harder and to make it difficult to read a CD-ROM using anything other than the publisher's software. One of them involves recording all the file lengths on the CD-ROM as multigigabyte, thwarting any attempts to copy the files to hard disk using standard copying software. The true lengths are embedded in the publisher's software or hidden (possibly encrypted) on the CD-ROM in an unexpected place. Another scheme uses intentionally incorrect ECCs in selected sectors, in the expectation that CD copying software will "fix" the errors. The application software checks the ECCs itself, refusing to work if they are "correct." Nonstandard gaps between the tracks and other physical "defects" are also possibilities.

2.3.9 CD-Rewritables

Although people are used to other write-once media such as paper and photographic film, there is a demand for a rewritable CD-ROM. One technology now available is **CD-RW (CD-ReWritable)**, which uses the same size media as CD-R.

However, instead of cyanine or phthalocyanine dye, CD-RW uses an alloy of silver, indium, antimony, and tellurium for the recording layer. This alloy has two stable states: crystalline and amorphous, with different reflectivities.

CD-RW drives use lasers with three different powers. At high power, the laser melts the alloy, converting it from the high-reflectivity crystalline state to the low-reflectivity amorphous state to represent a pit. At medium power, the alloy melts and reforms in its natural crystalline state to become a land again. At low power, the state of the material is sensed (for reading), but no phase transition occurs.

The reason CD-RW has not replaced CD-R is that the CD-RW blanks are more expensive than the CD-R blanks. Also, for applications consisting of backing up hard disks, the fact that once written, a CD-R cannot be accidentally erased is a feature, not a bug.

2.3.10 DVD

The basic CD/CD-ROM format has been around since 1980. By the mid-1990s optical media technology had improved dramatically, so higher-capacity video disks were becoming economically feasible. At the same time Hollywood was looking for a way to replace analog video tapes with an optical disk technology that had higher quality, was cheaper to manufacture, lasted longer, took up less shelf space in video stores, and did not have to be rewound. It was looking as if the wheel of progress for optical disks was about to turn once again.

This combination of technology and demand by three immensely rich and powerful industries has led to **DVD**, originally an acronym for **Digital Video Disk**, but now officially **Digital Versatile Disk**. DVDs use the same general design as CDs, with 120-mm injection-molded polycarbonate disks containing pits and lands that are illuminated by a laser diode and read by a photodetector. What is new is the use of

1. Smaller pits (0.4 microns versus 0.8 microns for CDs).
2. A tighter spiral (0.74 microns between tracks versus 1.6 microns for CDs).
3. A red laser (at 0.65 microns versus 0.78 microns for CDs).

Together, these improvements raise the capacity sevenfold, to 4.7 GB. A 1x DVD drive operates at 1.4 MB/sec (versus 150 KB/sec for CDs). Unfortunately, the switch to the red lasers used in supermarkets means that DVD players require a second laser to read existing CDs and CD-ROMs, which adds a little to the complexity and cost.

Is 4.7 GB enough? Maybe. Using MPEG-2 compression (standardized in IS 13346), a 4.7-GB DVD disk can hold 133 minutes of full-screen, full-motion video at high resolution (720×480), as well as soundtracks in up to eight languages and subtitles in 32 more. About 92 percent of all the movies Hollywood has ever made

are under 133 minutes. Nevertheless, some applications such as multimedia games or reference works may need more, and Hollywood would like to put multiple movies on the same disk, so four formats have been defined:

1. Single-sided, single-layer (4.7 GB).
2. Single-sided, dual-layer (8.5 GB).
3. Double-sided, single-layer (9.4 GB).
4. Double-sided, dual-layer (17 GB).

Why so many formats? In a word: politics. Philips and Sony wanted single-sided, dual-layer disks for the high-capacity version, but Toshiba and Time Warner wanted double-sided, single-layer disks. Philips and Sony did not think people would be willing to turn the disks over, and Time Warner did not believe putting two layers on one side could be made to work. The compromise: all combinations, with the market deciding which ones will survive. Well, the market has spoken. Philips and Sony were right. Never bet against technology.

The dual-layering technology has a reflective layer at the bottom, topped with a semireflective layer. Depending on where the laser is focused, it bounces off one layer or the other. The lower layer needs slightly larger pits and lands to be read reliably, so its capacity is slightly smaller than the upper layer's.

Double-sided disks are made by taking two 0.6-mm single-sided disks and gluing them together back to back. To make the thicknesses of all versions the same, a single-sided disk consists of a 0.6-mm disk bonded to a blank substrate (or perhaps in the future, one consisting of 133 minutes of advertising, in the hope that people will be curious as to what is down there). The structure of the double-sided, dual-layer disk is illustrated in Fig. 2-28.

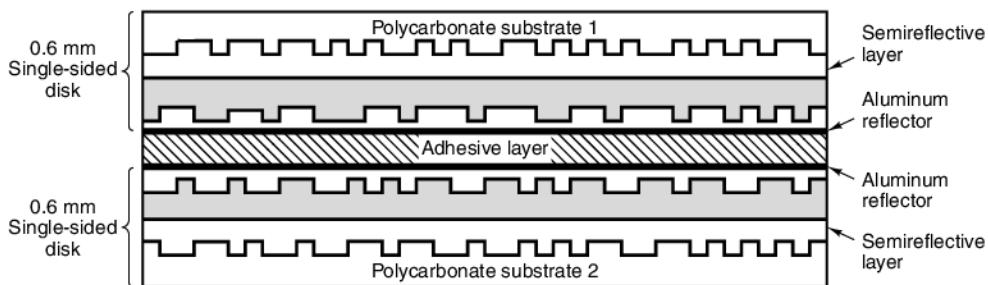


Figure 2-28. A double-sided, dual layer DVD disk.

DVD was devised by a consortium of 10 consumer electronics companies, seven of them Japanese, in close cooperation with the major Hollywood studios (some of which are owned by the Japanese electronics companies in the consortium). The computer and telecommunications industries were not invited to the

picnic, and the resulting focus was on using DVD for movie rentals. For example, standard features include real-time skipping of dirty scenes (to allow parents to turn a film rated NC17 into one safe for toddlers), six-channel sound, and support for Pan-and-Scan. The latter feature allows the DVD player to dynamically decide how to crop the left and right edges off movies (whose width:height ratio is 3:2) to fit on then-current television sets (whose aspect ratio was 4:3).

Another item the computer industry probably would not have thought of is an intentional incompatibility between disks intended for the United States and disks intended for Europe and yet other standards for other continents. Hollywood demanded this “feature” because new films are often released first in the United States and then physically shipped to Europe when the videos come out in the United States. The idea was to make sure European video stores could not buy videos in the U.S. too early, thereby reducing new movies’ European theater sales. If Hollywood had been running the computer industry, we would have had 3.5-inch floppy disks in the United States and 9-cm floppy disks in Europe.

2.3.11 Blu-ray

Nothing stands still in the computer business, certainly not storage technology. DVD was barely introduced before its successor threatened to make it obsolete. The successor to DVD is **Blu-ray**, so called because it uses a blue laser instead of the red one used by DVDs. A blue laser has a shorter wavelength than a red one, which allows it to focus more accurately and thus support smaller pits and lands. Single-sided Blu-ray disks hold about 25 GB of data; double-sided ones hold about 50 GB. The data rate is about 4.5 MB/sec, which is good for an optical disk, but still insignificant compared to magnetic disks (cf. ATAPI-6 at 100 MB/sec and wide Ultra5 SCSI at 640 MB/sec). It is expected that Blu-ray will eventually replace CD-ROMs and DVDs, but this transition will take some years.

2.4 INPUT/OUTPUT

As we mentioned at the start of this chapter, a computer system has three major components: the CPU, the memories (primary and secondary), and the **I/O (Input/Output)** equipment such as printers, scanners, and modems. So far we have looked at the CPU and the memories. Now it is time to examine the I/O equipment and how it is connected to the rest of the system.

2.4.1 Buses

Physically, most personal computers and workstations have a structure similar to the one shown in Fig. 2-29. The usual arrangement is a metal box with a large printed circuit board at the bottom or side, called the **motherboard** (parentboard,

for the politically correct). The motherboard contains the CPU chip, some slots into which DIMM modules can be clicked, and various support chips. It also contains a bus etched along its length, and sockets into which the edge connectors of I/O boards can be inserted.

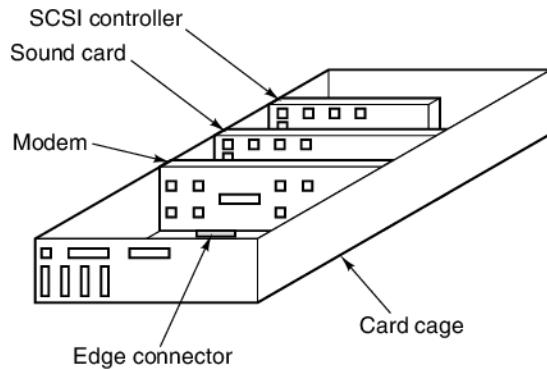


Figure 2-29. Physical structure of a personal computer.

The logical structure of a simple personal computer is shown in Fig. 2-30. This one has a single bus used to connect the CPU, memory, and I/O devices; most systems have two or more buses. Each I/O device consists of two parts: one containing most of the electronics, called the **controller**, and one containing the I/O device itself, such as a disk drive. The controller is usually integrated directly onto the motherboard or sometimes contained on a board plugged into a free bus slot. Even though the display (monitor) is not an option, the video controller is sometimes located on a plug-in board to allow the user to choose between boards with or without graphics accelerators, extra memory, and so on. The controller connects to its device by a cable attached to a connector on the back of the box.

The job of a controller is to control its I/O device and handle bus access for it. When a program wants data from the disk, for example, it gives a command to the disk controller, which then issues seeks and other commands to the drive. When the proper track and sector have been located, the drive begins outputting the data as a serial bit stream to the controller. It is the controller's job to break the bit stream up into units and write each unit into memory, as it is assembled. A unit is typically one or more words. A controller that reads or writes data to or from memory without CPU intervention is said to be performing **Direct Memory Access**, better known by its acronym **DMA**. When the transfer is completed, the controller normally causes an **interrupt**, forcing the CPU to immediately suspend running its current program and start running a special procedure, called an **interrupt handler**, to check for errors, take any special action needed, and inform the operating system that the I/O is now finished. When the interrupt handler is finished, the CPU continues with the program that was suspended when the interrupt occurred.

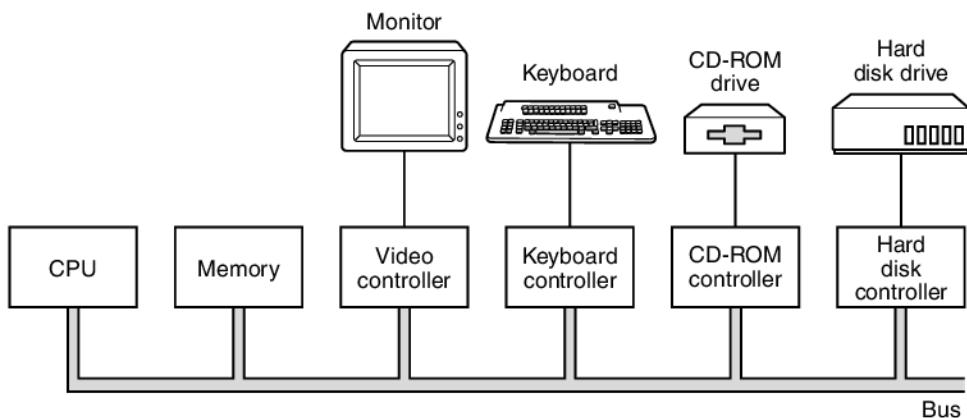


Figure 2-30. Logical structure of a simple personal computer.

The bus is used not only by the I/O controllers but also by the CPU for fetching instructions and data. What happens if the CPU and an I/O controller want to use the bus at the same time? The answer is that a chip called a **bus arbiter** decides who goes next. In general, I/O devices are given preference over the CPU, because disks and other moving devices cannot be stopped, and forcing them to wait would result in lost data. When no I/O is in progress, the CPU can have all the bus cycles for itself to reference memory. However, when some I/O device is also running, that device will request and be granted the bus when it needs it. This process is called **cycle stealing** and it slows down the computer.

This design worked fine for the first personal computers, since all the components were roughly in balance. However, as the CPUs, memories, and I/O devices got faster, a problem arose: the bus could no longer handle the load presented. On a closed system, such as an engineering workstation, the solution was to design a new and faster bus for the next model. Because nobody ever moved I/O devices from an old model to a new one, this approach worked fine.

However, in the PC world, people often upgraded their CPU but wanted to move their printer, scanner, and modem to the new system. Also, a huge industry had grown up around providing a vast range of I/O devices for the IBM PC bus, and this industry had exceedingly little interest in throwing out its entire investment and starting over. IBM learned this the hard way when it brought out the successor to the IBM PC, the PS/2 range. The PS/2 had a new, faster bus, but most clone makers continued to use the old PC bus, now called the **ISA (Industry Standard Architecture)** bus. Most disk and I/O device makers also continued to make controllers for it, so IBM found itself in the peculiar situation of being the only PC maker that was no longer IBM compatible. Eventually, it was forced back to supporting the ISA bus. Today the ISA bus has been relegated to ancient systems and computer museums, since it has been replaced by newer and faster standard bus

architectures. As an aside, please note that ISA stands for Instruction Set Architecture in the context of machine levels, whereas it stands for Industry Standard Architecture in the context of buses.

The PCI and PCIe Buses

Nevertheless, despite the market pressure not to change anything, the old bus really was too slow, so something had to be done. This situation led to other companies developing machines with multiple buses, one of which was the old ISA bus, or its backward-compatible successor, the **EISA (Extended ISA)** bus. The winner was the **PCI (Peripheral Component Interconnect)** bus. It was designed by Intel, but Intel decided to put all the patents in the public domain, to encourage the entire industry (including its competitors) to adopt it.

The PCI bus can be used in many configurations, but a typical one is illustrated in Fig. 2-31. Here the CPU talks to a memory controller over a dedicated high-speed connection. The controller talks to the memory and to the PCI bus directly, so CPU-memory traffic does not go over the PCI bus. Other peripherals connect to the PCI bus directly. A machine of this design would typically contain two or three empty PCI slots to allow customers to plug in PCI I/O cards for new peripherals.

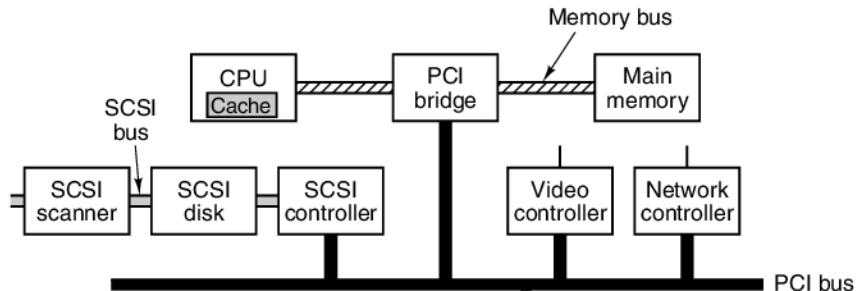


Figure 2-31. A typical PC built around the PCI bus. The SCSI controller is a PCI device.

No matter how fast something is in the computer world, a lot of people think it is too slow. This fate also befell the PCI bus, which is being replaced by **PCI Express**, abbreviated as **PCIe**. Most modern computers support both, so users can attach new, fast devices to the PCIe bus and older, slower ones to the PCI bus.

While the PCI bus was just an upgrade to the older ISA bus with higher speeds and more bits transferred in parallel, PCIe represents a radical change from the PCI bus. In fact, it is not even a bus at all. It is point-to-point network using bit-serial lines and packet switching, more like the Internet than like a traditional bus. Its architecture is shown in Fig. 2-32.

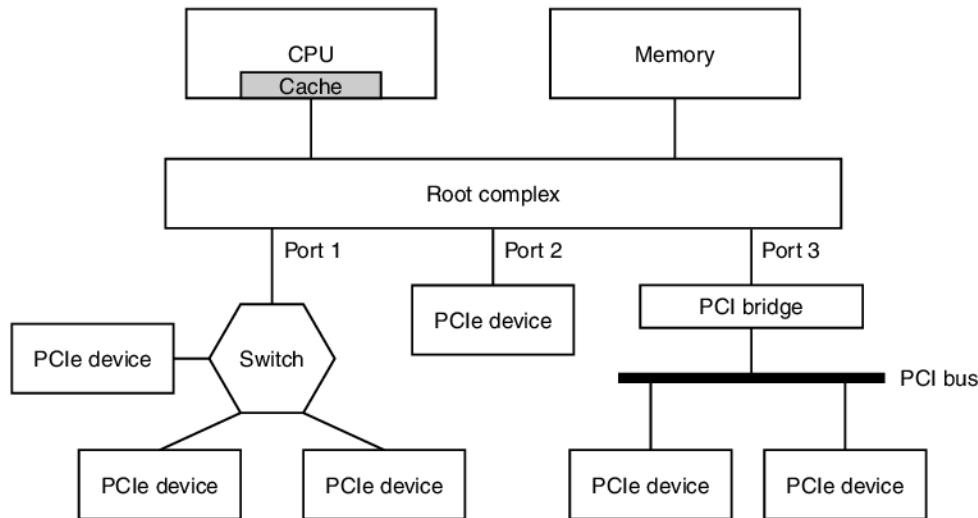


Figure 2-32. Sample architecture of a PCIe system with three PCIe ports.

Several things stand out immediately about PCIe. First, the connections between the devices are serial, that is, 1-bit wide rather than 8-, 16-, 32-, or 64-bits wide. While one might think that a 64-bit-wide connection would have a higher bandwidth than a 1-bit wide connection, in practice, differences in propagation time of the 64 bits, called the **skew**, means relatively low speeds have to be used. With a serial connection much higher speeds can be used and this more than offsets the loss of parallelism. PCI buses run at a maximum clock rate of 66 MHz. With 64 bits transferred per cycle, the data rate is 528 MB/sec. With a clock rate of 8 Gbps, even with serial transfer, the data rate of PCIe is 1 GB/sec. Furthermore, devices are not limited to a single wire pair to communicate with the root complex or a switch. A device can have up to 32 wire pairs, called **lanes**. These lanes are not synchronous, so skew is not important here. Most motherboards have a 16-lane slot for the graphics card, which in PCIe 3.0 will give the graphics card a bandwidth of 16 GB/sec, about 30 times faster than what a PCI graphics card can get. This bandwidth is necessary for increasingly demanding applications, such as 3D.

Second, all communication is point to point. When the CPU wants to talk to a device, it sends a packet to the device and generally later gets an answer. The packet goes through the root complex, which is on the motherboard, and then on to the device, possibly through a switch (or if the device is a PCI device, through the PCI bridge). This evolution from a system in which all devices listened to the same bus to one using point-to-point communications parallels the development of Ethernet (a popular local area network), which also started with a broadcast channel but now uses switches to enable point-to-point communication.

2.4.2 Terminals

Many kinds of I/O devices are available today. A few of the common ones are discussed below. Computer terminals consist of two parts: a keyboard and a monitor. In the mainframe world, these parts are often integrated into a single device and attached to the main computer by a serial line or over a telephone line. In the airline reservation, banking, and other mainframe-oriented industries, these devices are still in use. In the personal computer world, the keyboard and monitor are independent devices. Either way, the technology of the two parts is the same.

Keyboards

Keyboards come in several varieties. The original IBM PC came with a keyboard that had a snap-action switch under each key that gave tactile feedback and made a click when the key was depressed far enough. Nowadays, the cheaper keyboards have keys that just make mechanical contact when depressed. Better ones have a sheet of elastometric material (a kind of rubber) between the keys and the underlying printed circuit board. Under each key is a small dome that buckles when depressed far enough. A small spot of conductive material inside the dome closes the circuit. Some keyboards have a magnet under each key that passes through a coil when struck, thus inducing a current that can be detected. Various other methods, both mechanical and electromagnetic, are also in use.

On personal computers, when a key is depressed, an interrupt is generated and the keyboard interrupt handler (a piece of software that is part of the operating system) is started. The interrupt handler reads a hardware register inside the keyboard controller to get the number of the key (1 through 102) that was just depressed. When a key is released, a second interrupt is caused. Thus if a user depresses the SHIFT key, then depresses and releases the M key, then releases the SHIFT key, the operating system can see that the user wants an uppercase "M" rather than a lowercase "m." Handling of multikey sequences involving SHIFT, CTRL, and ALT is done entirely in software (including the infamous CTRL-ALT-DEL key sequence that is used to reboot PCs).

Touch Screens

While keyboards are in no danger of going the way of the manual typewriter, there is a new kid on the block when it comes to computer input: the touch screen. While these devices only became mass-market items with the introduction of Apple's iPhone in 2007, they go back much further. The first touch screen was developed at the Royal Radar Establishment in Malvern, U.K. in 1965. Even the much-heralded pinching capability of the iPhone dates back to work at the University of Toronto in 1982. Since then, many different technologies have been developed and marketed.

Touch devices fall into two categories: opaque and transparent. A typical opaque touch device is the touchpad on a notebook computer. A typical transparent device is the screen of a smart phone or tablet. We will only consider the latter here. They are usually called **touch screens**. The major types of touch screens are infrared, resistive, and capacitive.

Infrared screens have infrared transmitters, such as infrared light emitting diodes or lasers on (for example) the left and top edges of the bezel around the screen and detectors on the right and bottom edges. When a finger, stylus, or any opaque object blocks one or more beams, the corresponding detector senses the drop in signal and the hardware of the device can tell the operating system which beams have been blocked, allowing it to compute the (x, y) coordinates of the finger or stylus. While these devices have a long history and are still in use in kiosks and other applications, they are not used for mobile devices.

Another old technology consists of **resistive touch screens**. These consist of two layers, the top one of which is flexible. It contains a large number of horizontal wires. The one under it contains vertical wires. When a finger or other object depresses a point on the screen, one or more of the upper wires comes in contact with (or close to) the perpendicular wires in the lower layer. The electronics of the device make it possible to read out which area has been depressed. These screens can be built very inexpensively and are widely used in price-sensitive applications.

Both of these technologies are fine when the screen is pressed by one finger but have a problem when two fingers are used. To describe the problem, we will use the terminology of the infrared touch screen but the resistive one has the same problem. Imagine that the two fingers are at $(3, 3)$ and $(8, 8)$. As a result, the $x = 3$ and $x = 8$ vertical beams are interrupted as are the $y = 3$ and $y = 8$ horizontal beams. Now consider a different scenario with the fingers at $(3, 8)$ and $(8, 3)$, which are the opposite corners of the rectangle whose corners are $(3, 3), (8, 3), (8, 8)$, and $(3, 8)$. Precisely the same beams are blocked, so the software has no way of telling which of the two scenarios holds. This problem is called **ghosting**.

To be able to detect multiple fingers at the same time—a property required for pinching and expanding gestures—a new technology was needed. The one used on most smart phones and tablets (but not on digital cameras and other devices) is the **projected capacitive touch screen**. There are various types but the most common one is the **mutual capacitance** type. All touch screens that can detect two or more points of contact at the same time are known as **multitouch screens**. Let us now briefly see how they work.

For readers who are a bit rusty on their high-school physics, a **capacitor** is a device that can store electric charge. A simple one has two conductors separated by an insulator. In modern touch screens, a grid-like pattern of thin “wires” running vertically is separated from a horizontal grid by a thin insulating layer. When a finger touches the screen, it changes the capacitance at all the intersections touched (possibly far apart). This change can be measured. As a demonstration that a modern touch screen is not like the older infrared and resistive screens, try touching

one with a pen, pencil, paper clip, or gloved finger and you will see that nothing happens. The human body is good at storing electric charge, as anyone who has shuffled across a rug on a cold, dry day and then touched a metal doorknob can painfully testify. Plastic, wooden, and metal instruments are not nearly as good as people in terms of their capacitance.

The “wires” in a touch screen are not the usual copper wires found in normal electrical devices since they would block the light from the screen. Instead they are thin (typically 50 micron) strips of transparent, conducting indium tin oxide bonded to opposite sides of a thin glass plate, which together form the capacitors. In some newer designs, the insulating glass plate is replaced by a thin layer of silicon dioxide (sand!), with the three layers sputtered (sprayed, atom by atom) onto some substrate. Either way, the capacitors are protected from dirt and scratching by a glass plate placed above, to form the surface of the screen to be touched. The thinner the upper glass plate, the more sensitive the performance but the more fragile the device is.

In operation, voltages are applied alternately to the horizontal and vertical “wires” while the voltage values, which are affected by the capacitance of each intersection, are read off the other ones. This operation is repeated many times per second with the coordinates touched fed to the device driver as a stream of (x, y) pairs. Further processing, such as determining whether pointing, pinching, expanding, or swiping is taking place is done by the operating system. If you use all 10 fingers, and bring a friend to add some more, the operating system will have its hands full, but the multitouch hardware will be up to the job.

Flat Panel Displays

The first computer monitors used **cathode ray tubes (CRTs)**, just like old television sets. They were far too bulky and heavy to be used in notebook computers, so a more compact display technology had to be developed for their screens. The development of flat panel displays provided the compact form factor necessary for notebooks, and these devices also used less power. Today the size and power benefits of the flat panel display have all but eliminated the use of CRT monitors.

The most common flat panel display technology is the **LCD (Liquid Crystal Display)**. It is highly complex, has many variations, and is changing rapidly, so this description will, of necessity, be brief and greatly simplified.

Liquid crystals are viscous organic molecules that flow like a liquid but also have spatial structure, like a crystal. They were discovered by an Austrian botanist, Friedrich Reinitzer, in 1888 and first applied to displays (e.g., calculators, watches) in the 1960s. When all the molecules are lined up in the same direction, the optical properties of the crystal depend on the direction and polarization of the incoming light. Using an applied electric field, the molecular alignment, hence the optical properties, can be changed. In particular, by shining a light through a liquid

crystal, the intensity of the light exiting from it can be controlled electrically. This property can be exploited to construct flat panel displays.

An LCD display screen consists of two parallel glass plates between which is a sealed volume containing a liquid crystal. Transparent electrodes are attached to both plates. A light behind the rear plate (either natural or artificial) illuminates the screen from behind. The transparent electrodes attached to each plate are used to create electric fields in the liquid crystal. Different parts of the screen get different voltages, to control the image displayed. Glued to the front and rear of the screen are polarizing filters because the display technology requires the use of polarized light. The general setup is shown in Fig. 2-33(a).

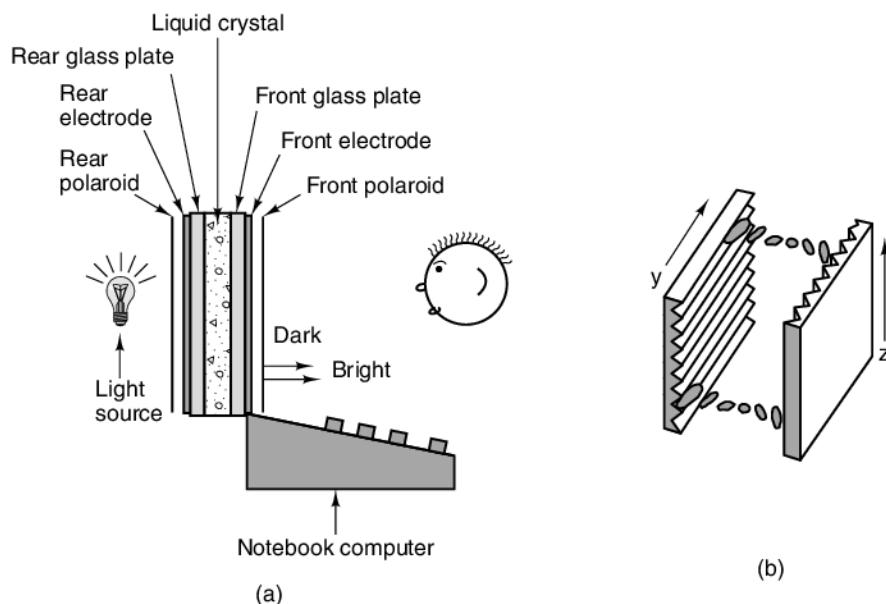


Figure 2-33. (a) The construction of an LCD screen. (b) The grooves on the rear and front plates are perpendicular to one another.

Although many kinds of LCD displays are in use, we will now consider one particular kind, the **TN (Twisted Nematic)** display, as an example. In this display, the rear plate contains tiny horizontal grooves and the front plate contains tiny vertical grooves, as illustrated in Fig. 2-33(b). In the absence of an electric field, the LCD molecules tend to align with the grooves. Since the front and rear alignments differ by 90 degrees, the molecules (and thus the crystal structure) twist from rear to front.

At the rear of the display is a horizontal polarizing filter. It allows in only horizontally polarized light. At the front of the display is a vertical polarizing filter. It allows only vertically polarized light to pass through. If there were no liquid pres-

ent between the plates, horizontally polarized light let in by the rear polarizing filter would be blocked by the front polarizing filter, making the screen black.

However the twisted crystal structure of the LCD molecules guides the light as it passes and rotates its polarization, making it come out vertically. Thus in the absence of an electric field, the LCD screen is uniformly bright. By applying a voltage to selected parts of the plate, the twisted structure can be destroyed, blocking the light in those parts.

Two schemes can be used for applying the voltage. In a (low-cost) **passive matrix display**, both electrodes contain parallel wires. In a 1920×1080 display (the size for full high-definition video), for example, the rear electrode might have 1920 vertical wires and the front one 1080 horizontal ones. By putting a voltage on one of the vertical wires and then pulsing one of the horizontal ones, the voltage at one selected pixel position can be changed, making it go dark briefly. A **pixel** (originally a picture element, is a colored dot from which all digital images are built). By repeating this pulse with the next pixel and then the next one, a dark scan line can be painted. Normally, the entire screen is painted 60 times a second to fool the eye into thinking there is a constant image there.

The other scheme in widespread use is the **active matrix display**. It is more expensive but it gives a better image. Instead of just having two sets of perpendicular wires, it has a tiny switching element at each pixel position on one of the electrodes. By turning these on and off, an arbitrary voltage pattern can be created across the screen, allowing for an arbitrary bit pattern. The switching elements are called **thin film transistors** and the flat panel displays using them are often called **TFT displays**. Most notebook computers and stand-alone flat panel displays for desktop computers use TFT technology now.

So far we have described how a monochrome display works. Suffice it to say that color displays use the same general principles as monochrome displays but the details are a great deal more complicated. Optical filters are used to separate the white light into red, green, and blue components at each pixel position so these can be displayed independently. Every color can be built up from a linear superposition of these three primary colors.

Still new screen technologies are on the horizon. One of the more promising is the **Organic Light Emitting Diode (OLED)** display. It consists of layers of electrically charged organic molecules sandwiched between two electrodes. Voltage changes cause the molecules to get excited and move to higher energy states. When they drop back to their normal state, they emit light. More detail is beyond the scope of this book (and its authors).

Video RAM

Most monitors are refreshed 60–100 times per second from a special memory, called a **video RAM**, on the display's controller card. This memory has one or more bit maps that represent the screen image. On a screen with, say, 1920×1080

pixels, the video RAM would contain 1920×1080 values, one for each pixel. In fact, it might contain many such bit maps, to allow rapid switching from one screen image to another.

On a garden-variety display, each pixel would be represented as a 3-byte RGB value, one each for the intensity of the red, green, and blue components of the pixel's color (high-end displays use 10 or more bits per color). From the laws of physics, it is known that any color can be constructed from a linear superposition of red, green, and blue light.

A video RAM with 1920×1080 pixels at 3 bytes/pixel requires over 6.2 MB to store the image and a fair amount of CPU time to do anything with it. For this reason, some computers compromise by using an 8-bit number to indicate the color desired. This number is then used as an index into a hardware table called the **color palette** that contains 256 entries, each holding a 24-bit RGB value. Such a design, called **indexed color**, reduces the video RAM memory requirements by $2/3$, but allows only 256 colors on the screen at once. Usually, each window on the screen has its own mapping, but with only one hardware color palette, often when multiple windows are present on the screen, only the current one has its colors rendered correctly. Color palettes with 2^{16} entries are also used, but the gain here is only $1/3$.

Bit-mapped video displays require a lot of bandwidth. To display full-screen, full-color multimedia on a 1920×1080 display requires copying 6.2 MB of data to the video RAM for every frame. For full-motion video, a rate of at least 25 frame/sec is needed, for a total data rate of 155 MB/sec. This load is more than the original PCI bus could handle (132 MB/sec) but PCIe can handle it with ease.

2.4.3 Mice

As time goes on, computers are being used by people with less expertise in how computers work. Computers of the ENIAC generation were used only by the people who built them. In the 1950s, computers were only used by highly skilled professional programmers. Now, computers are widely used by people who need to get some job done and do not know (or even want to know) much about how computers work or how they are programmed.

In the old days, most computers had command line interfaces, to which users typed commands. Since people who are not computer specialists often perceived command line interfaces as user-unfriendly, if not downright hostile, many computer vendors developed point-and-click interfaces, such as the Macintosh and Windows. Using this model requires having a way to point at the screen. The most common way of allowing users to point at the screen is with a mouse.

A **mouse** is a small plastic box that sits on the table next to the keyboard. When it is moved around on the table, a little pointer on the screen moves too, allowing users to point at screen items. The mouse has one, two, or three buttons on top, to allow users to select items from menus. Much blood has been spilled as a

result of arguments about how many buttons a mouse ought to have. Naive users prefer one (it is hard to push the wrong button if there is only one), but sophisticated ones like the power of multiple buttons to do fancy things.

Three kinds of mice have been produced: mechanical mice, optical mice, and optomechanical mice. The first mice had two rubber wheels protruding through the bottom, with their axles perpendicular to one another. When the mouse was moved parallel to its main axis, one wheel turned. When it is moved perpendicular to its main axis, the other one turned. Each wheel drove a variable resistor or potentiometer. By measuring changes in the resistance, it was possible to see how much each wheel had rotated and thus calculate how far the mouse had moved in each direction. Later, this design was been replaced by one in which a ball that protruded slightly from the bottom was used instead of wheels. It is shown in Fig. 2-34.

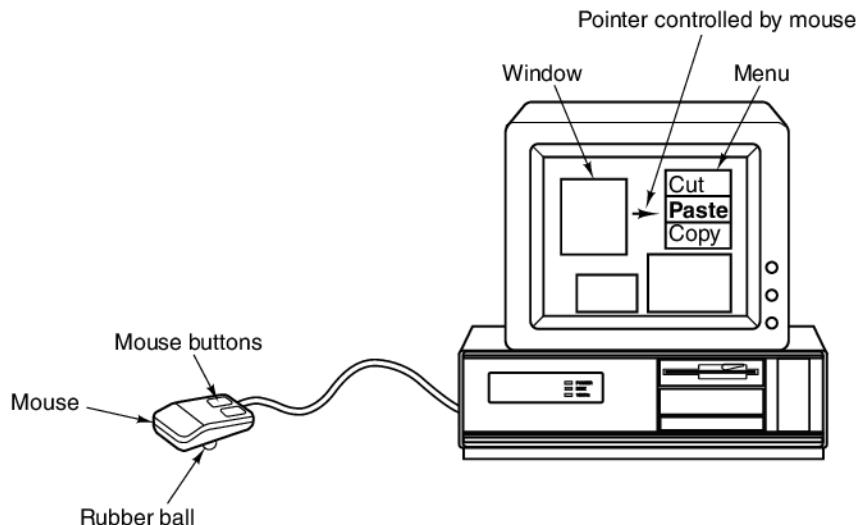


Figure 2-34. A mouse being used to point to menu items.

The second kind of mouse is the optical mouse. This kind has no wheels or ball. Instead, it has an **LED (Light Emitting Diode)** and a photodetector on the bottom. Early optical mice required a mouse pad with closely spaced lines on it to detect how many lines had been crossed and thus how far the mouse had moved. Modern optical mice contain an LED that illuminates the imperfections of the underlying surface along with a tiny video camera that records a small image (typically 18×18 pixels) up to 1000 times/sec. Consecutive images are compared to see how far the mouse has moved. Some optical mice use a laser instead of an LED for illumination. They are more accurate, but also more expensive.

The third kind of mouse is optomechanical. Like the newer mechanical mouse, it has a rolling ball that turns two shafts aligned at 90 degrees to each other.

The shafts are connected to encoders that have slits through which light can pass. As the mouse moves, the shafts rotate, and light pulses strike the detectors whenever a slit comes between an LED and its detector. The number of pulses detected is proportional to the amount of motion.

Although mice can be set up in various ways, a common arrangement is to have the mouse send a sequence of 3 bytes to the computer every time the mouse moves a certain minimum distance (e.g., 0.01 inch), sometimes called a **mickey**. Usually, these characters come in on a serial line, one bit at time. The first byte contains a signed integer telling how many units the mouse has moved in the *x*-direction since the last time. The second byte gives the same information for *y* motion. The third byte contains the current state of the mouse buttons. Sometimes 2 bytes are used for each coordinate.

Low-level software in the computer accepts this information as it comes in and converts the relative movements sent by the mouse to an absolute position. It then displays an arrow on the screen at the position corresponding to where the mouse is. When the arrow points at the proper item, the user clicks a mouse button, and the computer can then figure out which item has been selected from its knowledge of where the arrow is on the screen.

2.4.4 Game Controllers

Video games typically have heavy user I/O demands, and in the video console market specialized input devices have been developed. In this section we look at two recent developments in video game controllers, the Nintendo Wiimote and the Microsoft Kinect.

Wiimote Controller

First released in 2006 with the Nintendo Wii game console, the Wiimote controller contains traditional gamepad buttons plus a dual motion-sensing capability. All interactions with the Wiimote are sent in real time to the game console using an internal Bluetooth radio. The motion sensors in the Wiimote allow it to sense its own movement in three dimensions, plus when pointed at the television it provides a fine-grained pointing capability.

Figure 2-35 illustrates how the Wiimote implements this motion-sensing function. Tracking of the Wiimote's movement in three dimensions is accomplished with an internal 3-axis accelerometer. This device contains three small masses, each of which can move in the *x*, *y*, and *z* axis (with respect to the accelerometer chip). These masses move in proportion to the degree of acceleration in their particular axis, which changes the capacitance of the mass with respect to a fixed metal wall. By measuring these three changing capacitances, it becomes possible to sense acceleration in three dimensions. Using this technology and some classic calculus, the Wii console can track the Wiimote's movement in space. As you

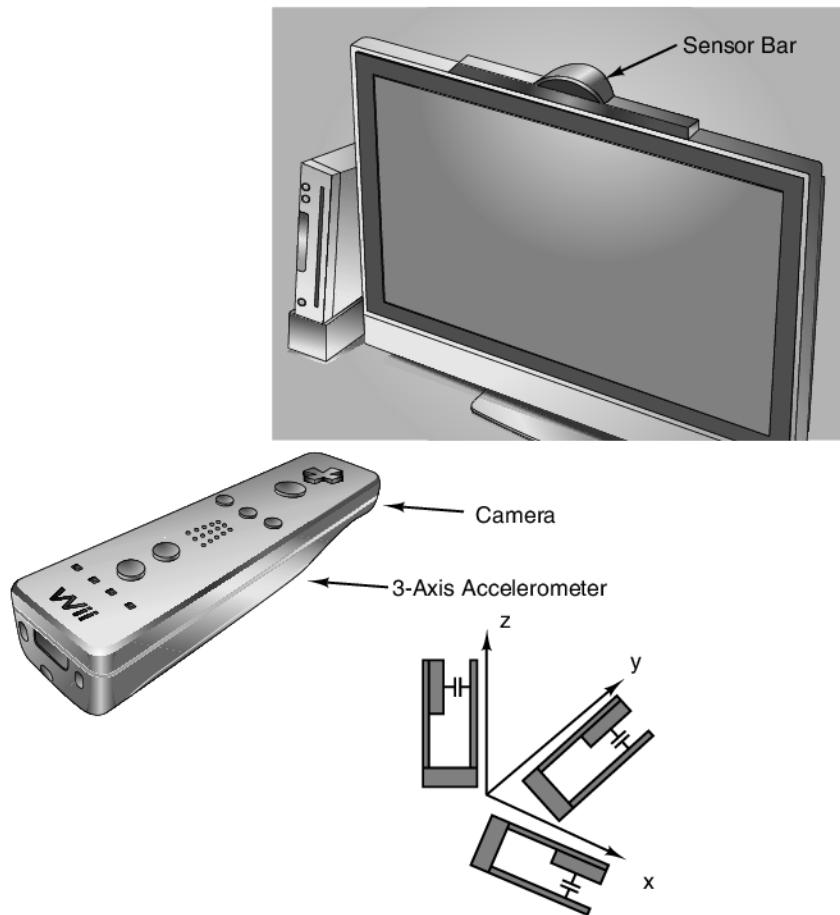


Figure 2-35. The Wiimote video game controller motion sensors.

swing the Wiimote to hit a virtual tennis ball, the motion of the Wiimote is tracked as you swing toward the ball, and if you twist your wrist at the last moment to put topspin on the ball, the Wiimote accelerometers will sense that motion as well.

While the accelerometers perform well at tracking the motion of the Wiimote as it moves in three dimensions, they cannot provide the fine-grained motion sensing necessary to control a pointer on the television screen. The accelerometers suffer from unavoidable tiny errors in their acceleration measurements, thus over time the exact location of the Wiimote (based on integration of its accelerations) will become increasingly inaccurate.

To provide fine-grained motion sensing, the Wiimote utilizes a clever computer vision technology. Sitting atop the television is a “sensor bar” which contains LEDs a fixed width apart. Contained in the Wiimote is a camera that when pointed

at the sensor bar can deduce the distance and orientation of the Wiimote with respect to the television. Since the sensor bar's LEDs are a fixed distance apart, their distance as viewed by the Wiimote is proportional to the Wiimote's distance from the sensor bar. The location of the sensor bar in the Wiimote's field of view indicates the direction that the Wiimote is pointing with respect to the television. By continuously sensing this orientation, it is possible to support a fine-grained pointing capability without the positional errors inherent to accelerometers.

Kinect Controller

The Microsoft Kinect takes the computer vision capabilities of game controllers to a whole new level. This device uses computer vision alone to determine the user's interactions with the game console. It works by sensing the user's position in the room, plus the orientation and motion of their body. Games are controlled by making predetermined motions with your hands, arms, and whatever else the game designers think you should flail in an effort to control their game.

The sensing capability of the Kinect is based on a depth camera combined with a video camera. The depth camera computes the distance of object in the Kinect's field of view. It does this by emitting a two-dimensional array of infrared laser dots, then capturing their reflections with an infrared camera. Using a computer vision technique called "structured lighting," the Kinect can determine the distance of the objects in its view based on how the stipple of infrared dots is disturbed by the lighted surfaces.

Depth information is combined with the texture information returned from the video camera to produce a textured depth map. This map can then be processed by computer vision algorithms to locate the people in the room (even recognizing their faces) and the orientation and motion of their body. After processing, information about the persons in the room is sent to the game console which uses this data to control the video game.

2.4.5 Printers

Having prepared a document or fetched a page from the World Wide Web, users often want to print it, so many computers can be equipped with a printer. In this section we will describe some of the more common kinds of printers.

Laser Printers

Probably the most exciting development in printing since Johann Gutenberg invented movable type in the fifteenth century is the **laser printer**. This device combines a high-quality image, excellent flexibility, great speed, and moderate cost into a single compact peripheral. Laser printers use almost the same technology as

photocopy machines. In fact, many companies make devices that combine copying and printing (and sometimes fax as well).

The basic technology is illustrated in Fig. 2-36. The heart of the printer is a rotating precision drum (or in some high-end systems, a belt). At the start of each page cycle, it is charged up to about 1000 volts and coated with a photosensitive material. Then light from a laser is scanned along the length of the drum by reflecting it off a rotating octagonal mirror. The light beam is modulated to produce a pattern of light and dark spots. The spots where the beam hits lose their charge.

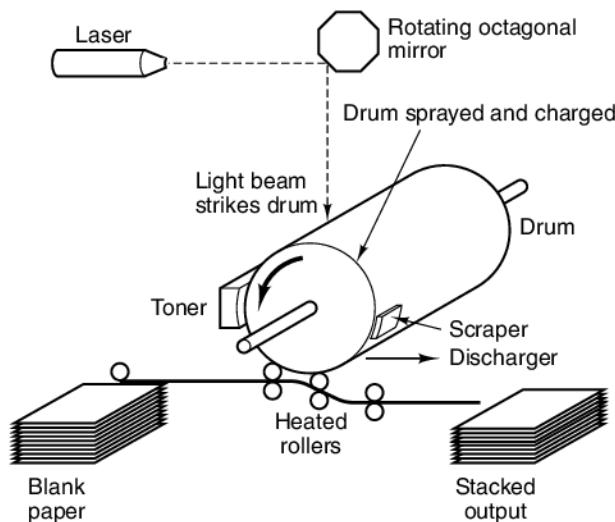


Figure 2-36. Operation of a laser printer.

After a line of dots has been painted, the drum rotates a fraction of a degree to allow the next line to be painted. Eventually, the first line of dots reaches the toner, a reservoir of an electrostatically sensitive black powder. The toner is attracted to those dots that are still charged, thus forming a visual image of that line. A little later in the transport path, the toner-coated drum is pressed against the paper, transferring the black powder to the paper. The paper is then passed through heated rollers to fuse the toner to the paper permanently, fixing the image. Later in its rotation, the drum is discharged and scraped clean of any residual toner, preparing it for being charged and coated again for the next page.

That this process is an exceedingly complex combination of physics, chemistry, mechanical engineering, and optical engineering hardly needs to be said. Nevertheless, complete assemblies, called **print engines**, are available from several vendors. Laser printer manufacturers combine the print engines with their own electronics and software to make a complete printer. The electronics consist of a fast embedded CPU along with megabytes of memory to hold a full-page bit map and numerous fonts, some of them built in and some of them downloadable. Most

printers accept commands that describe the pages to be printed (as opposed to simply accepting bit maps prepared by the main CPU). These commands are given in languages such as HP's PCL and Adobe's PostScript or PDF, which are complete, albeit specialized, programming languages.

Laser printers at 600 dpi and up can do a reasonable job of printing black and white photographs but the technology is trickier than it might at first appear. Consider a photograph scanned in at 600 dpi that is to be printed on a 600-dpi printer. The scanned image contains 600×600 pixels/inch, each one consisting of a gray value from 0 (white) to 255 (black). The printer can also print 600 dpi, but each printed pixel is either black (toner present) or white (no toner present). Gray values cannot be printed.

The usual solution to printing images with gray values is to use **halftoning**, the same as commercially printed posters. The image is broken up into halftone cells, each typically 6×6 pixels. Each cell can contain between 0 and 36 black pixels. The eye perceives a cell with many pixels as darker than one with fewer pixels. Gray values in the range 0 to 255 are represented by dividing this range into 37 zones. Values from 0 to 6 are in zone 0, values from 7 to 13 are in zone 1, and so on (zone 36 is slightly smaller than the others because 37 does not divide 256 exactly). Whenever a gray value in zone 0 is encountered, its halftone cell on the paper is left blank, as illustrated in Fig. 2-37(a). A zone-1 value is printed as 1 black pixel. A zone-2 value is printed as 2 black pixels, as shown in Fig. 2-37(b). Other zone values are shown in Fig. 2-37(c)–(f). Of course, taking a photograph scanned at 600 dpi and halftoning this way reduces the effective resolution to 100 cells/inch, called the **halftone screen frequency**, conventionally measured in **lpi (lines per inch)**.

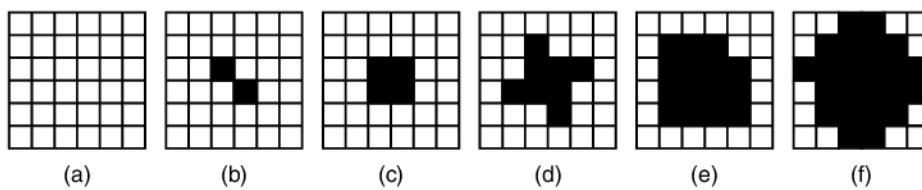


Figure 2-37. Halftone dots for various grayscale ranges. (a) 0–6. (b) 14–20. (c) 28–34. (d) 56–62. (e) 105–111. (f) 161–167.

Color Printing

Although most laser printers are monochrome, color laser printers are starting to become more common, so some explanation of color printing (also applicable to inkjet and other printers) is perhaps useful here. As you might imagine, it is not trivial. Color images can be viewed in one of two ways: transmitted light and reflected light. Transmitted-light images, such as those produced on monitors, are

built up from the linear superposition of the three additive primary colors, which are red, green, and blue.

In contrast, reflected-light images, such as color photographs and pictures in glossy magazines, absorb certain wavelengths of light and reflect the rest. These are built up from a linear superposition of the three subtractive primary colors, cyan (all red absorbed), magenta (all green absorbed), and yellow (all blue absorbed). In theory, every color can be produced by mixing cyan, yellow, and magenta ink. In practice it is difficult to get the inks pure enough to absorb all light and produce a true black. For this reason, nearly all color printing systems use four inks: cyan, magenta, yellow, and black. These systems are called **CMYK printers**. The K is sometimes attributed to blacK but it really stands for the Key plate with which the color plates are aligned in conventional four-color printing presses. Monitors, in contrast, use transmitted light and the RGB system for producing colors.

The complete set of colors that a display or printer can produce is called its **gamut**. No device has a gamut that matches the real world, since typically each color comes in 256 intensities, giving only 16,777,216 discrete colors. Imperfections in the technology reduce the total more, and the remaining ones are not always uniformly spaced over the color spectrum. Furthermore, color perception has a lot to do with how the rods and cones in the human retina work, and not just the physics of light.

As a consequence of the above observations, converting a color image that looks fine on the screen to an identical printed one is far from trivial. Among the problems are

1. Color monitors use transmitted light; color printers use reflected light.
2. Monitors have 256 intensities per color; color printers must halftone.
3. Monitors have a dark background; paper has a light background.
4. The RGB gamut of a monitor and the CMYK gamut of a printer are different.

Getting printed color images to match real life (or even to match screen images) requires hardware device calibration, sophisticated software for building and using International Color Consortium profiles, and considerable expertise on the part of the user.

Inkjet Printers

For low-cost home printing, **inkjet printers** are a favorite. The movable print head, which holds the ink cartridges, is swept horizontally across the paper by a belt while ink is sprayed from tiny nozzles. The ink droplets have a volume of about 1 picoliter, so 100 million of them fit in a single drop of water.