

8

INPUT AND OUTPUT

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

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

In this chapter we discuss the nature of communicating using busses, starting

first with simple bus fundamentals and then exploring multiple-bus architectures. We then take a look at some of the more common I/O devices that are connected to these busses.

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

8.1 Simple Bus Architectures

A computer system may contain many components that need to communicate with each other. In a worst case scenario, all N components need to simultaneously communicate with every other component, in which $N^2/2$ links are needed for N components. The number of links becomes prohibitively large for even small values of N , but fortunately, as for long distance telecommunication, not all devices need to simultaneously communicate.

A **bus** is a common pathway that connects a number of devices. An example of a bus can be found on the **motherboard** (the main circuit board that contains the central processing unit) of a personal computer, as illustrated in simplified form in Figure 8-1. (For a look at a real motherboard, see Figure 1-6.) A typical moth-

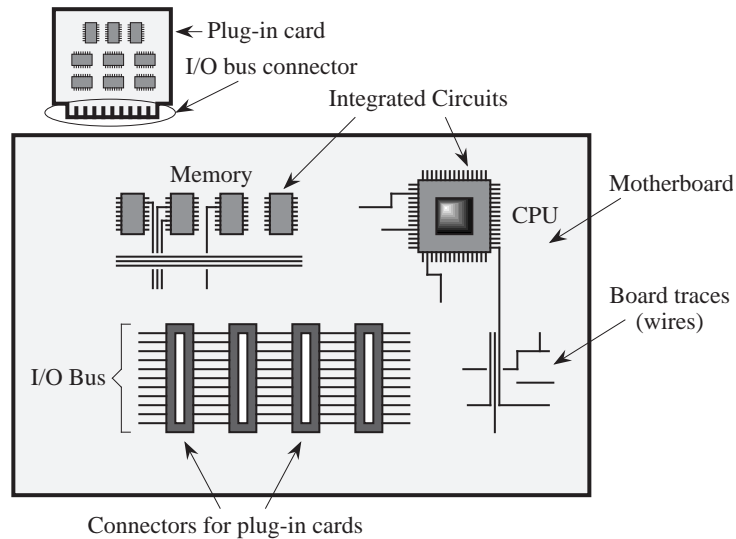


Figure 8-1 A simplified motherboard of a personal computer (top view).

erboard contains **integrated circuits** (ICs) such as the CPU chip and memory

chips, board **traces** (wires) that connect the chips, and a number of busses for chips or devices that need to communicate with each other. In Figure 8-1, an I/O bus is used for a number of cards that plug into the connectors, perpendicular to the motherboard in this example configuration.

8.1.1 BUS STRUCTURE, PROTOCOL, AND CONTROL

A bus consists of the physical parts, like connectors and wires, and a **bus protocol**. The wires can be partitioned into separate groups for control, address, data, and power as illustrated in Figure 8-2. A single bus may have a few different

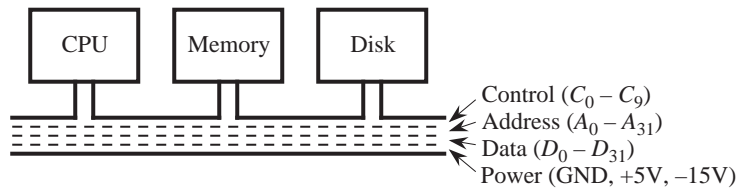


Figure 8-2 Simplified illustration of a bus.

power lines, and the example shown in Figure 8-2 has lines for ground (GND) at 0 V, and positive and negative voltages at +5 V, and -15 V, respectively.

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

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

A bus can be classified as one of two types: **synchronous** or **asynchronous**. For a synchronous bus, one of the devices that is connected to the bus contains an oscillator (a clock) that sends out a sequence of 1's and 0's at timed intervals as illustrated in Figure 8-3. The illustration shows a train of pulses that repeat at 10 ns intervals, which corresponds to a clock rate of 100 MHz. Ideally, the clock

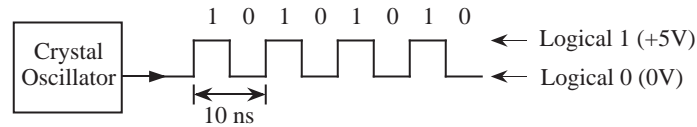


Figure 8-3 A 100 MHz bus clock.

would be a perfect square wave (instantaneous rise and fall times) as shown in the figure. In practice, the rise and fall times are approximated by a rounded, trapezoidal shape.

8.1.2 BUS CLOCKING

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

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

8.1.3 THE SYNCHRONOUS BUS

As an example of how communication takes place over a synchronous bus, consider the timing diagram shown in Figure 8-4 which is for a synchronous read of a word of memory by a CPU. At some point early in time interval T_1 , while the clock is high, the CPU places the address of the location it wants to read onto the address lines of the bus. At some later time during T_1 , after the voltages on the address lines have become stable, or “settled,” the \overline{MREQ} and \overline{RD} lines are asserted by the CPU. \overline{MREQ} informs the memory that it is selected for the transfer (as opposed to another device, like a disk). The \overline{RD} line informs the selected device to perform a read operation. The overbars on \overline{MREQ} and \overline{RD} indicate that a 0 must be placed on these lines in order to assert them.

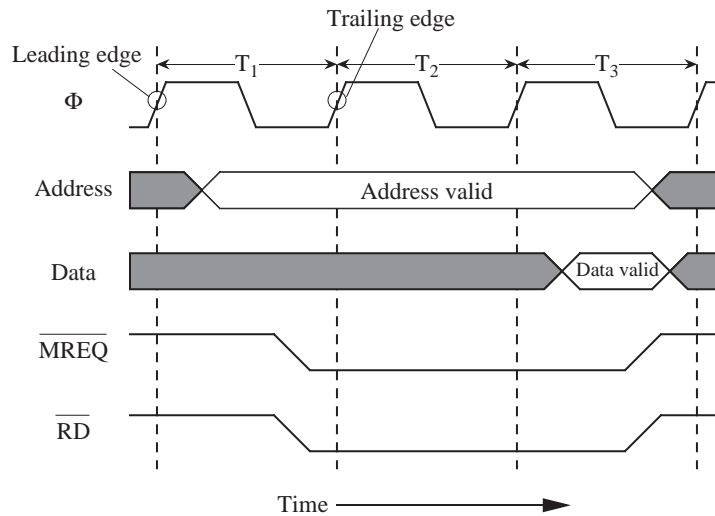


Figure 8-4 Timing diagram for a synchronous memory read (adapted from [Tanenbaum, 1999]).

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

8.1.4 THE ASYNCHRONOUS BUS

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

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

ter/slave combination responds more quickly than a slow master/slave combination.

As an example of how communication takes place over an asynchronous bus, consider the timing diagram shown in Figure 8-5. In order for a CPU to read a

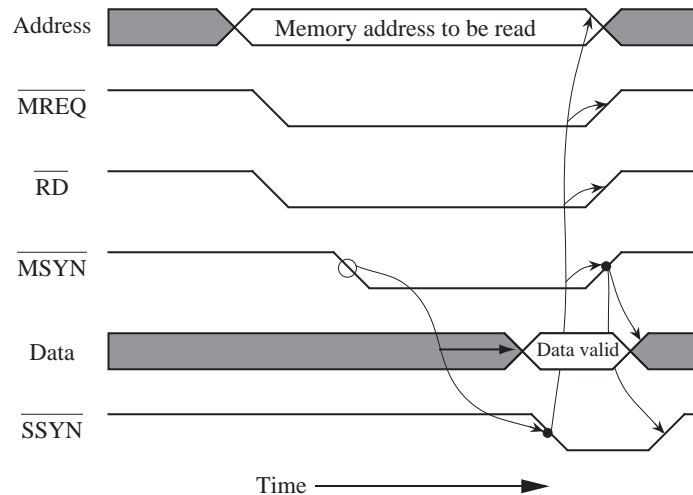


Figure 8-5 Timing diagram for asynchronous memory read (adapted from [Tanenbaum, 1999]).

word from memory, it places an address on the bus, followed by asserting \overline{MREQ} and \overline{RD} . After these lines settle, the CPU asserts \overline{MSYN} . This event triggers the memory to perform a read operation, which results in \overline{SSYN} eventually being asserted by the memory. This is indicated by the **cause-and-effect** arrow between \overline{MSYN} and \overline{SSYN} shown in Figure 8-5. This method of synchronization is referred to as a “full handshake.” In this particular implementation of a full handshake, asserting \overline{MSYN} initiates the transfer, followed by the slave asserting \overline{SSYN} , followed by the CPU de-asserting \overline{MSYN} , followed by the memory de-asserting \overline{SSYN} . Notice the absence of a bus clock signal.

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

8.1.5 BUS ARBITRATION—MASTERS AND SLAVES

Suppose now that more than one device wants to be a bus master at the same

time. How is a decision made as to who will be the bus master? This is the **bus arbitration** problem, and there are two basic schemes: **centralized** and **decentralized** (distributed). Figure 8-6 illustrates four organizations for these two

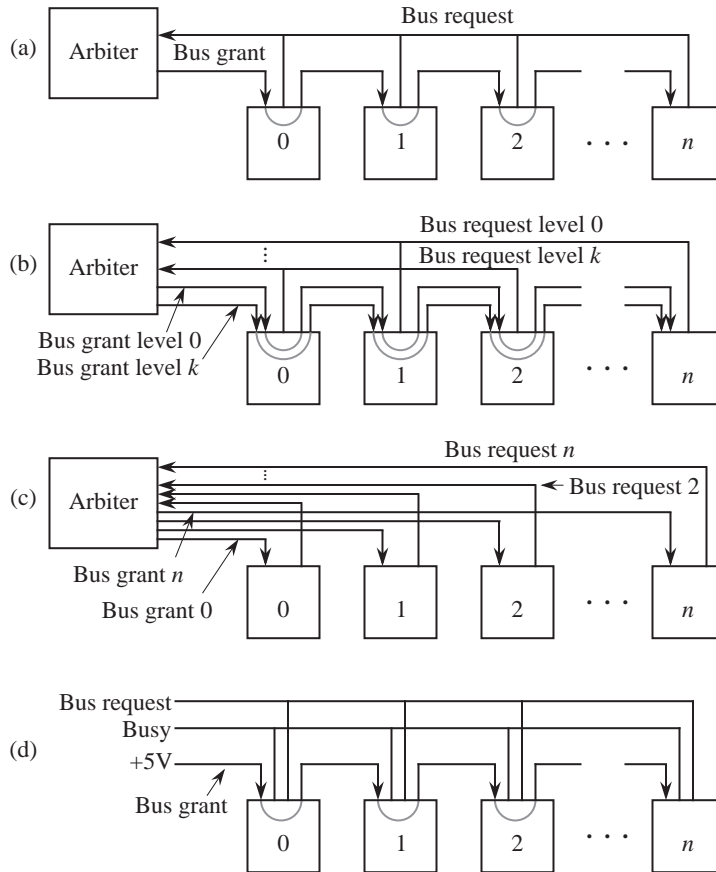


Figure 8-6 (a) Simple centralized bus arbitration; (b) centralized arbitration with priority levels; (c) fully centralized bus arbitration; (d) decentralized bus arbitration. (Adapted from [Tanenbaum, 1999]).

schemes. In Figure 8-6a, a centralized arbitration scheme is used. Devices 0 through n are all attached to the same bus (not shown), and they also share a **bus request** line that goes into an **arbiter**. When a device wants to be a bus master, it asserts the bus request line. When the arbiter sees the bus request, it determines if a **bus grant** can be issued (it may be the case that the current bus master will not allow itself to be interrupted). If a bus grant can be issued, then the arbiter asserts the bus grant line. The bus grant line is **daisy chained** from one device to the next. The first device that sees the asserted bus grant and also wants to be the bus master takes control of the bus and does not propagate the bus grant to higher

numbered devices. If a device does not want the bus, then it simply passes the bus grant to the next device. In this way, devices that are electrically closer to the arbiter have higher priorities than devices that are farther away.

Sometimes an absolute priority ordering is not appropriate, and a number of bus request/bus grant lines are used as shown in Figure 8-6(b). Lower numbered bus request lines have higher priorities than higher numbered bus request lines. In order to raise the priority of a device that is far from the arbiter, it can be assigned to a lower numbered bus request line. Priorities are assigned within a group on the same bus request level by electrical proximity to the arbiter.

Taking this to an extreme, each device can have its own bus request/bus grant line as shown in Figure 8-6(c). This fully centralized approach is the most powerful from a logical standpoint, but from a practical standpoint, it is the least scalable of all of the approaches. A significant cost is the need for additional lines (a precious commodity) on the bus.

In a fourth approach, a decentralized bus arbitration scheme is used as illustrated in Figure 8-6(d). Notice the lack of a central arbiter. A device that wants to become a bus master first asserts the bus request line, and then it checks if the bus is busy. If the busy line is not asserted, then the device sends a 0 to the next higher numbered device on the daisy chain, asserts the busy line, and de-asserts the bus request line. If the bus is busy, or if a device does not want the bus, then it simply propagates the bus grant to the next device.

Arbitration needs to be a fast operation, and for that reason, a centralized scheme will only work well for a small number of devices (up to about eight). For a large number of devices, a decentralized scheme is more appropriate.

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

When a card is removed, higher cards should be repositioned to fill in the missing slot, or a dummy card that continues the bus grant line should be inserted in

place of the removed card. Fast devices (like disk controllers) should be given higher priority than slow devices (like terminals), and should thus be placed close to the arbiter in a centralized scheme, or close to the beginning of the Bus grant line in a decentralized scheme. This is an imperfect solution given the opportunities for leaving gaps in the bus and getting the device ordering wrong. These days, it is more common for each device to have a separate path to the arbiter.

8.2 Bridge-Based Bus Architectures

From a logical viewpoint, all of the system components are connected directly to the system bus in the previous section. From an operational viewpoint, this approach is overly burdensome on the system bus because simultaneous transfers cannot take place among the various components. While every device at the same time, several independent transfers may need to take place at any time. For example, a graphics component may be repainting a video screen at the same time that a cache line is being retrieved from main memory, while an I/O transfer is taking place over a network.

These different transfers are typically segregated onto separate busses through the use of **bridges**. Figure 8-7 illustrates bridging with Intel's Pentium II Xeon processors. At the top of the diagram are two Pentium II processors, arranged in a **symmetric multiprocessor** (SMP) configuration. The operating system performs load balancing by selecting one processor over another when assigning tasks (this is different from parallel processing, discussed in Chapter 10, in which multiple processors work on a common problem.) Each Pentium II processor has a "backside bus" to its own cache of 3200 MB/sec (8 bytes wide \times 400 MHz), thus segregating cache traffic from other bus traffic.

Working down from the top of the diagram, the two Pentium II processors converge on the System Bus (sometimes called the "frontside bus." The System Bus is 32 bits wide and makes transfers on both the leading and falling edges of the 100 MHz bus clock, giving a total available bandwidth of 4 bytes \times 2 edges \times 100 MHz = 800 MB/sec that is shared between the processors.

At the center of the diagram is the Intel 440GX AGPset "Host Bridge" which connects the System Bus to the remaining busses. The Host Bridge acts as a go-between among the System Bus, the main memory, the graphics processor, and a hierarchy of other busses. To the right of the Host Bridge is the main memory (synchronous DRAM), connected to the Host Bridge by an 800 MB/sec bus.

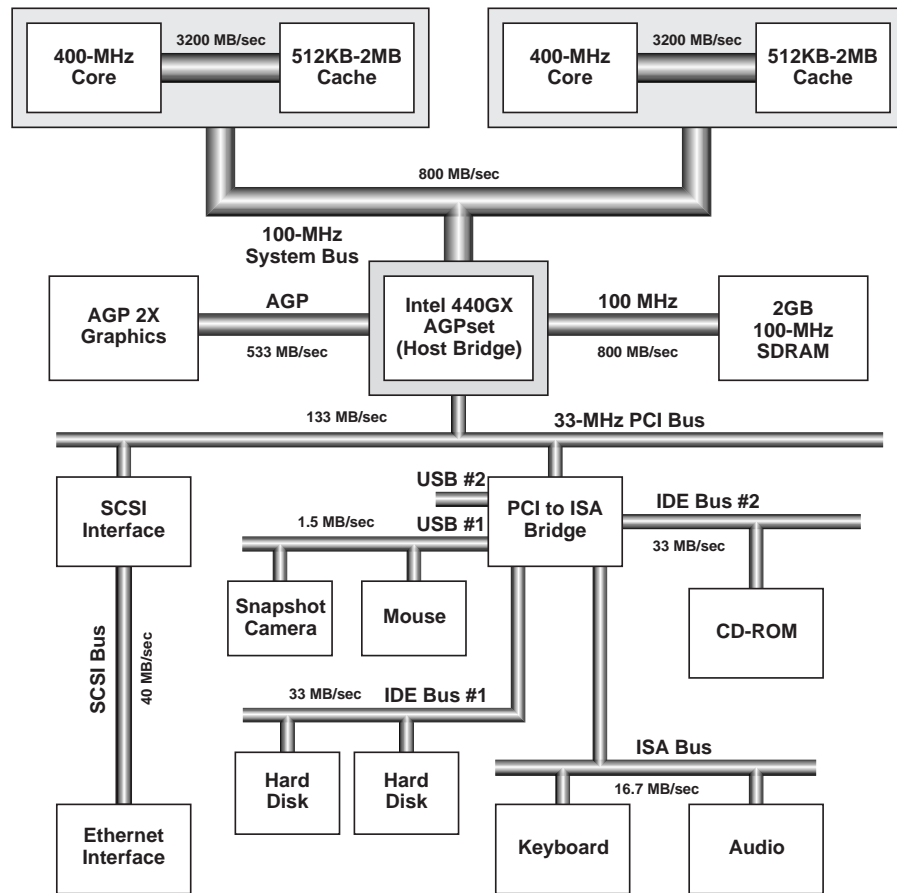


Figure 8-7 Bridging with dual Pentium II Xeon processors on Slot 2.

[Source: <http://www.intel.com>]

In this particular example, a separate bus known as the Advanced Graphics Port (AGP) is provided from the Host Bridge to the graphics processor over a 533 MB/sec bus. Graphics rendering (that is, filling an object with color) commonly needs texture information that is too large to economically place on a graphics card. The AGP allows for a high speed path between the graphics processor and the main memory, where texture maps can now be effectively stored.

Below the Host Bridge is the 33 MHz Peripheral Component Interconnect (PCI) bus, which connects the remaining busses to the Host Bridge. The PCI bus has a number of components connected to it, such as the Small Computer System Interface (SCSI) controller which is yet another bus, that in this illustration accepts an Ethernet network interface card. Prior to the introduction of the

AGP, graphics cards were placed on the PCI bus, which created a bottleneck for all of the other bus traffic.

Attached to the PCI bus is a PCI-to-ISA bridge, which actually provides bridging for two 1.5 MB/sec Universal Serial Bus (USB) busses, two 33 MB/sec integrated Drive Electronics (IDE) busses, and a 16.7 MB/sec Industry Standard Architecture (ISA) bus. The IDE busses are generally used for disk drives, the ISA bus is generally used for moderate rate devices like printers and voice-band modems, and the USB busses are used for low bit rate devices like mice and snapshot digital cameras.

8.3 Communication Methodologies

Computer systems have a wide range of communication tasks. The CPU must communicate with memory and with a wide range of I/O devices, from extremely slow devices such as keyboards, to high-speed devices like disk drives and network interfaces. There may be multiple CPUs that communicate with one another either directly or through a shared memory, as described in the previous section for the dual Pentium II Xeon configuration.

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

8.3.1 PROGRAMMED I/O

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

The operations that take place for programmed I/O are shown in the flowchart in Figure 8-8. The CPU first checks the status of the disk by reading a special register that can be accessed in the memory space, or by issuing a special I/O instruction if this is how the architecture implements I/O. If the disk is not ready to be read or written, then the process loops back and checks the status continuously until the disk is ready. This is referred to as a **busy-wait**. When the disk is finally ready, then a transfer of data is made between the disk and the CPU.

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

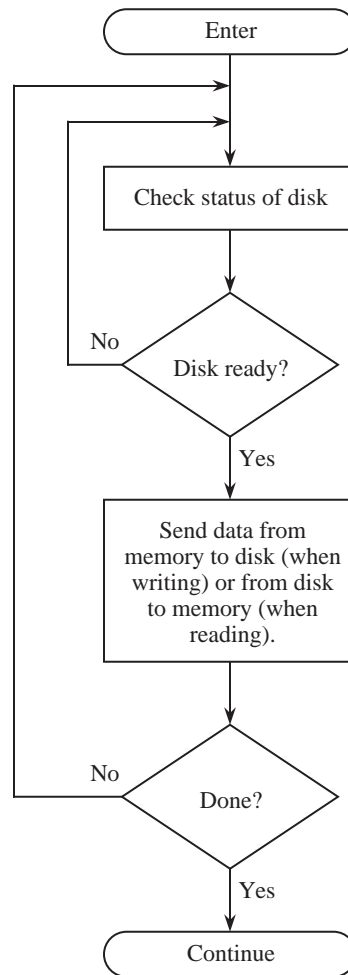


Figure 8-8 Programmed I/O flowchart for a disk transfer.

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

8.3.2 INTERRUPT-DRIVEN I/O

With interrupt driven I/O, the CPU does not access a device until it needs servicing, and so it does not get caught up in busy-waits. In interrupt-driven I/O, the device requests service through a special interrupt request line that goes

directly to the CPU. The restaurant analogy would have the patron politely tapping silverware on a water glass, thus interrupting the waiter when service is required.

A flowchart for interrupt driven I/O is shown in Figure 8-9. The CPU issues a

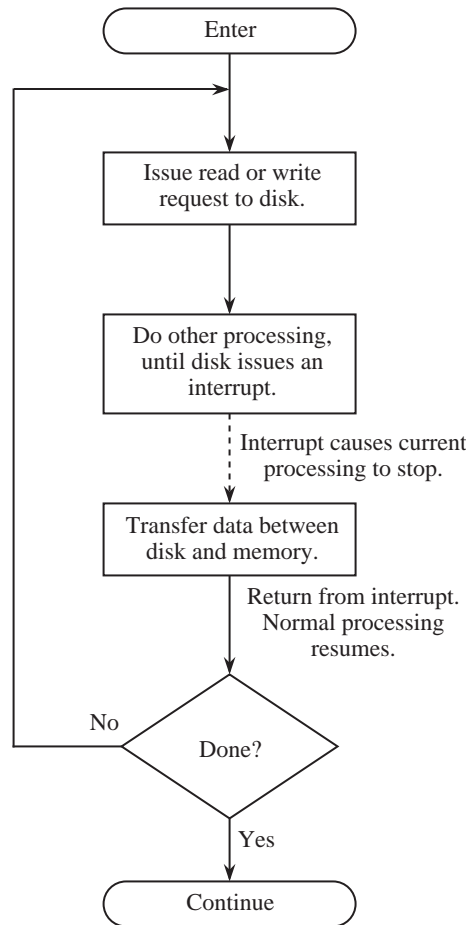


Figure 8-9 Interrupt driven I/O flowchart for a disk transfer.

request to the disk for reading or for writing, and then immediately resumes execution of another process. At some later time, when the disk is ready, it interrupts the CPU. The CPU then invokes an **interrupt service routine (ISR)** for the disk, and returns to normal execution when the interrupt service routine completes its task. The ISR is similar in structure to the procedure presented in Chapter 4, except that interrupts occur asynchronously with respect to the pro-

cess being executed by the CPU: an interrupt can occur at any time during program execution.

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

At the time when an interrupt occurs (which is sometimes loosely referred to as a **trap**, even though traps usually have a different meaning, as explained in Chapter 6), the Processor Status Register and the Program Counter (`%psr` and `%pc` for the ARC) are automatically pushed onto the stack, and the Program Counter is loaded with the address of the appropriate interrupt service routine. The processor status register is pushed onto the stack because it contains the interrupt flag (IF), and the processor must disable interrupts for at least the duration of the first instruction of the ISR. (See problem 8.2.) Execution of the interrupt routine then begins. When the interrupt service routine finishes, execution of the interrupted program then resumes.

The ARC `jmp1` instruction (see Chapter 4) will not work properly for resuming execution of the interrupted routine, because in addition to restoring the program counter contents, the processor status register must be restored. Instead, the `rett` (return from trap) instruction is invoked, which reverses the interrupt process and restores the `%psr` and `%pc` registers to their values prior to the interrupt. In the ARC architecture, `rett` is an arithmetic format instruction with `op3 = 111001`, and an unused `rd` field (all zeros).

8.3.3 DIRECT MEMORY ACCESS (DMA)

Although interrupt driven I/O frees the CPU until the device requires service, the CPU is still responsible for making the actual data transfer. Figure 8-10 high-

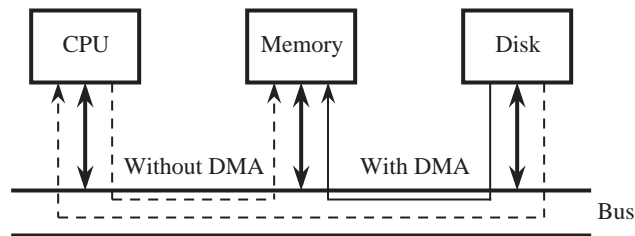


Figure 8-10 DMA transfer from disk to memory bypasses the CPU.

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

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

Figure 8-11 illustrates the DMA process for a disk transfer. The CPU sets up the DMA device and then signals the device to start the transfer. While the transfer is taking place, the CPU continues execution of another process. When the DMA transfer is completed, the device informs the CPU through an interrupt. A system that implements DMA thus also implements interrupts as well.

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

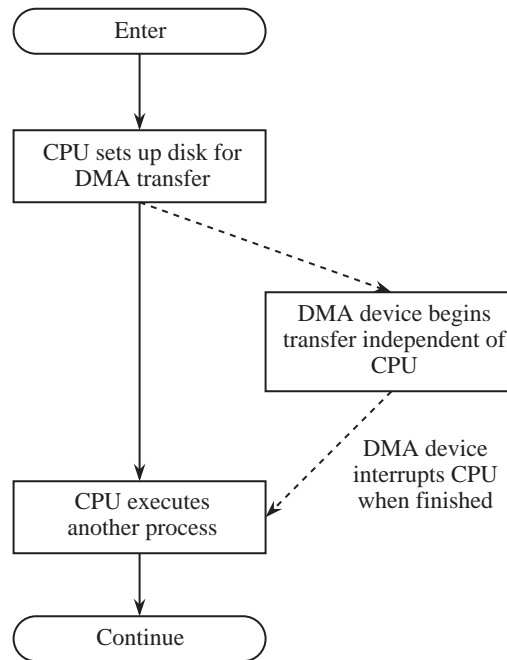


Figure 8-11 DMA flowchart for a disk transfer.

8.4 Case Study: Communication on the Intel Pentium Architecture

The Intel Pentium processor family is Intel's current state-of-the-art implementation of their venerable x86 family, which began with the Intel 8086, released in 1978. The Pentium is itself a processor family, with versions that emphasize high speed, multiprocessor environments, graphics, low power, *etc.* In this section we examine the common features that underlie the Pentium System Bus, which connects the Pentium to the Host Bridge (see Section 8.2).

8.4.1 SYSTEM CLOCK, BUS CLOCK, AND BUS SPEEDS

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

tions. A common bus clock frequency in Pentium systems is 66 MHz.

8.4.2 ADDRESS, DATA, MEMORY, AND I/O CAPABILITIES

The system bus effectively has 32 address lines, and can thus address up to 4 GB of main memory. Its data bus is 64 bits wide; thus the processor is capable of transferring an 8-byte quadword in one bus cycle. (Intel x86 words are 16-bits long.) We say “effectively” because in fact the Pentium processor decodes the least significant three address lines, A_2 - A_0 , into eight “byte enable” lines, $BE_0\#$ - $BE_7\#$, prior to placing them on the system bus.¹ The values on these eight lines specify the byte, word, double word, or quad word that is to be transferred from the base address specified by A_{31} - A_3 .

8.4.3 DATA WORDS HAVE SOFT-ALIGNMENT

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

As a bow to the small address spaces of early family members, all Intel processors have separate address spaces for memory and I/O accesses. The address space to be selected is specified by the $M/I/O\#$ bus line. A high value on this line selects the 4 GB memory address space, and low specifies the I/O address space. Separate opcodes, IN and OUT, are used to access this space. It is the responsibility of all devices on the bus to sample the $M/I/O\#$ line at the beginning of each bus cycle to determine the address space to which the bus cycle is referring—memory or I/O. Figure 8-12 shows these address spaces graphically. I/O addresses in the x86 family are limited to 16 bits, allowing up to 64K I/O locations.

8.4.4 BUS CYCLES IN THE PENTIUM FAMILY

The Pentium processor has a total of 18 different bus cycles, to serve different

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

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

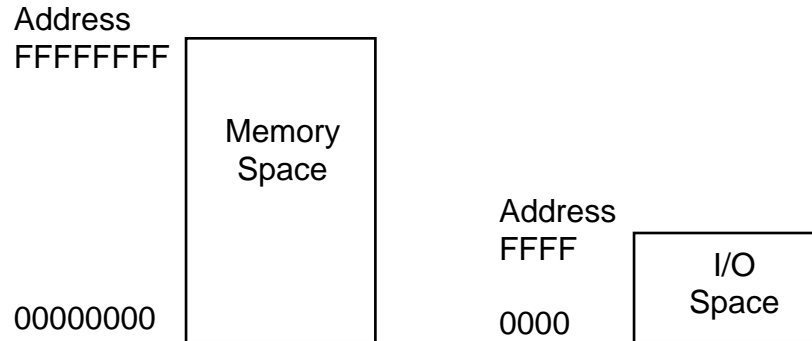


Figure 8-12 Intel memory and I/O address spaces.

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

8.4.5 MEMORY READ AND WRITE BUS CYCLES

The “standard” read and write cycles are shown in Figure 8-13. By convention,

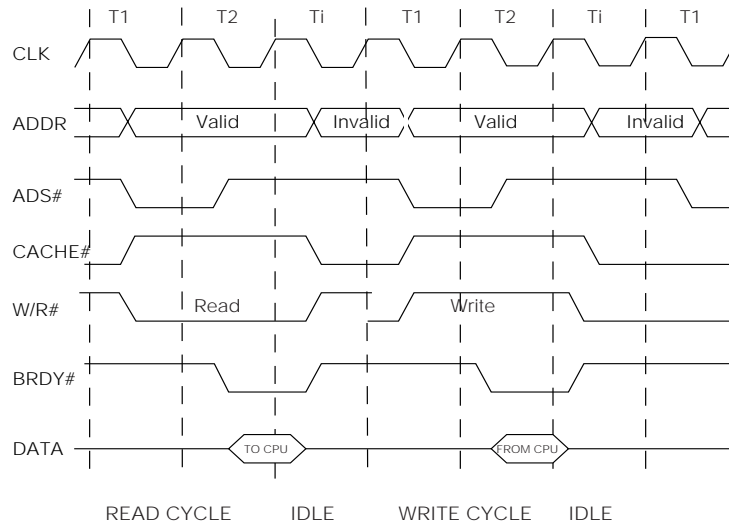


Figure 8-13 The standard Intel Pentium read and write bus cycles.

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

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

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

8.4.6 THE BURST READ BUS CYCLE

Because of the critical need to supply the CPU with instructions and data from memory that is inherently slower than the CPU, Intel designed the burst read and write cycles. These cycles read and write four eight-byte quad words in a burst, from consecutive addresses. Figure 8-14 shows the Pentium burst read

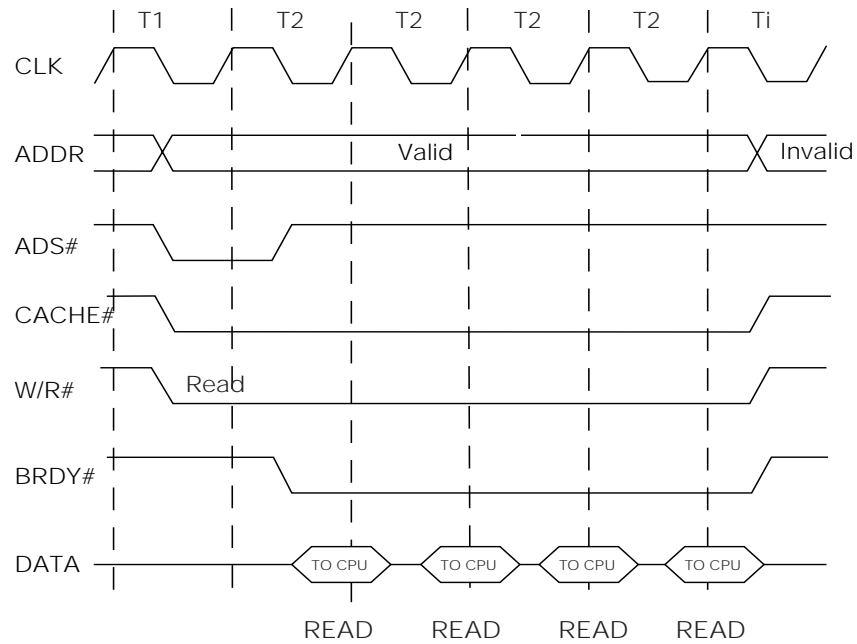


Figure 8-14 The Intel Pentium burst read bus cycle.

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

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

8.4.7 BUS HOLD FOR REQUEST BY BUS MASTER

There are two bus signals for use by devices requesting to become bus master: hold (HOLD) and hold acknowledge (HLDA). Figure 8-15 shows how the transactions work. The figure assumes that the processor is in the midst of a read cycle when the HOLD request signal arrives. The processor completes the current (read) cycle, and inserts two idle cycles, Ti. During the falling edge of the

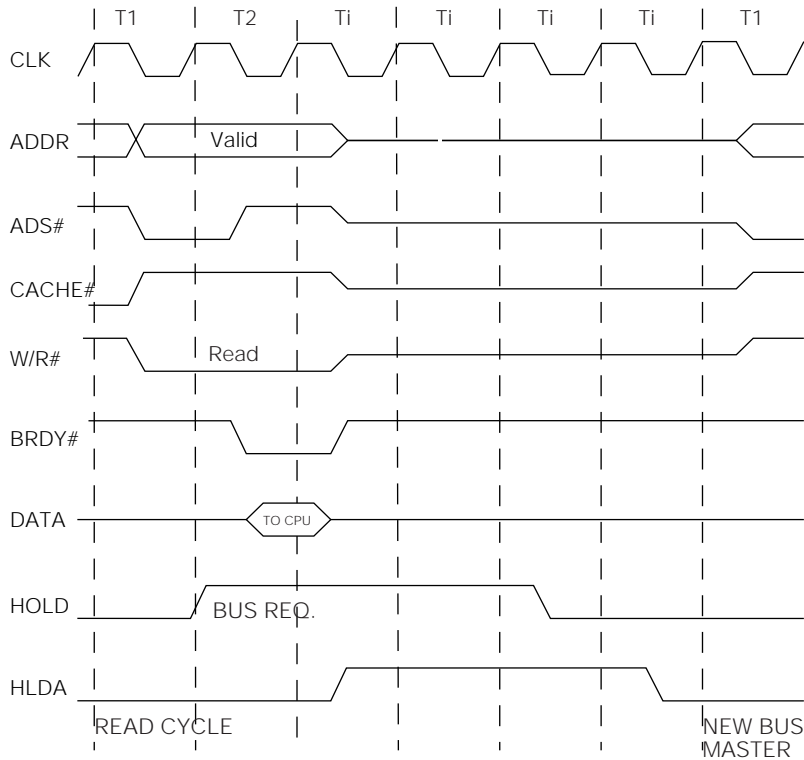


Figure 8-15 The Intel Pentium Hold-Hold Acknowledge bus cycle.

second T_i cycle the processor floats all of its lines and asserts HLDA. It keeps HLDA asserted for two clocks. At the end of the second clock cycle the device asserting HLDA “owns” the bus, and it may begin a new bus operation at the following cycle, as shown at the far right end of the figure. In systems of any complexity there will be a separate bus controller chip to mediate among the several devices that may wish to become the bus master.

8.4.8 DATA TRANSFER RATES

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

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

or 264 million bytes per second. In burst mode this rate increases to four 8-byte

bursts in five clock cycles, for a transfer rate of

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

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

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

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

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

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

8.5 Mass Storage

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

8.5.1 MAGNETIC DISKS

A magnetic disk is a device for storing information that supports a large storage

density and a relatively fast access time. A **moving head** magnetic disk is composed of a stack of one or more **platters** that are spaced several millimeters apart and are connected to a **spindle**, as shown in Figure 8-16. Each platter has two

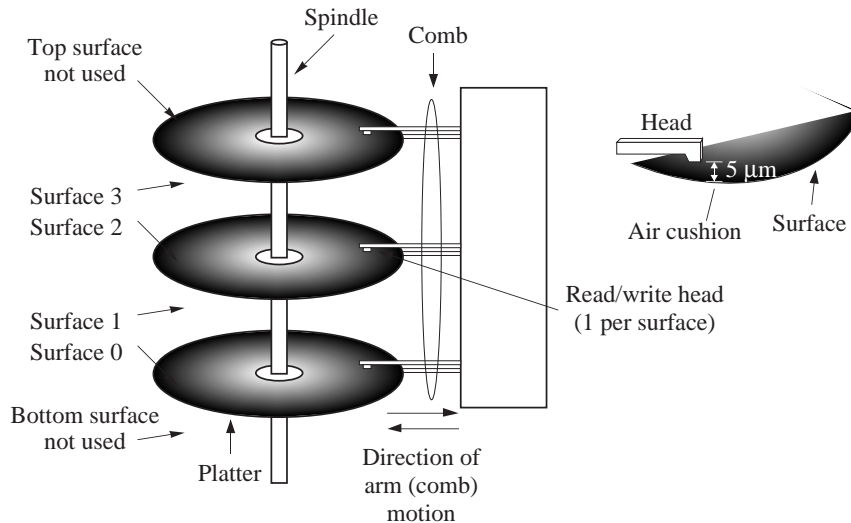


Figure 8-16 A magnetic disk with three platters.

surfaces made of aluminum or glass (which expands less than aluminum as it heats up), which are coated with small particles of a magnetic material such as iron oxide, which is the essence of rust. This is why disk platters, floppy diskettes, audio tapes, and other magnetic media are brown. Binary 1's and 0's are stored by magnetizing small areas of the material.

A single **head** is dedicated to each surface. Six heads are used in the example shown in Figure 8-16, for the six surfaces. The top surface of the top platter and the bottom surface of the bottom platter are sometimes not used on multi-platter disks because they are more susceptible to contamination than the inner surfaces. The heads are attached to a common **arm** (also known as a **comb**) which moves in and out to reach different portions of the surfaces.

In a hard disk drive, the platters rotate at a constant speed of typically 3600 to 10,000 revolutions per minute (RPM). The heads read or write data by magnetizing the magnetic material as it passes under the heads when writing, or by sensing the magnetic fields when reading. Only a single head is used for reading or writing at any time, so data is stored in serial fashion even though the heads

can in principle be used to read or write several bits in parallel. One reason that the parallel mode of operation is not normally used is that heads can become misaligned, which corrupts the way that data is read or written. A single surface is relatively insensitive to the alignment of the corresponding head because the head position is always accurately known with respect to reference markings on the disk.

Data encoding

Only the transitions between magnetized areas are sensed when reading a disk, and so runs of 1's or 0's may not be accurately detected unless a method of encoding is used that embeds timing information into the data to identify the breaks between bits. **Manchester encoding** is one method that addresses this problem, and another method is **modified frequency modulation (MFM)**. For comparison, Figure 8-17a shows an ASCII 'F' character encoded in the

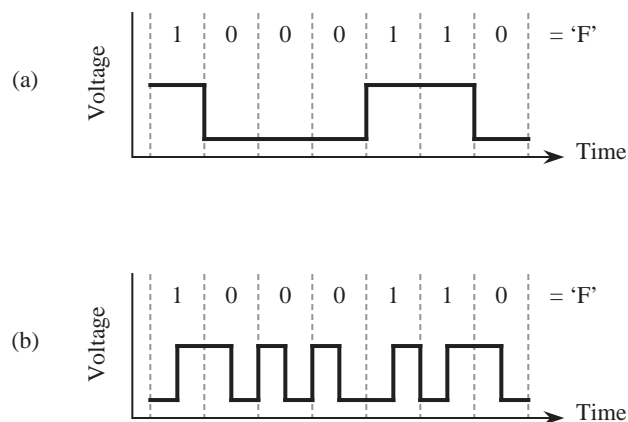


Figure 8-17 (a) Straight amplitude (NRZ) encoding of ASCII 'F'; (b) Manchester encoding of ASCII 'F'.

non-return-to-zero (NRZ) format, which is the way information is carried inside of a CPU. Figure 8-17b shows the same character encoded in the Manchester code. In Manchester encoding there is a transition between high and low signals on every bit, resulting in a transition at every bit time. A transition from low to high indicates a 1, whereas a transition from high to low indicates a 0. These transitions are used to recover the timing information.

A single surface contains several hundred concentric **tracks**, which in turn are composed of **sectors** of typically 512 bytes in size, stored serially, as shown in

Figure 8-18. The sectors are spaced apart by **inter-sector gaps**, and the tracks are

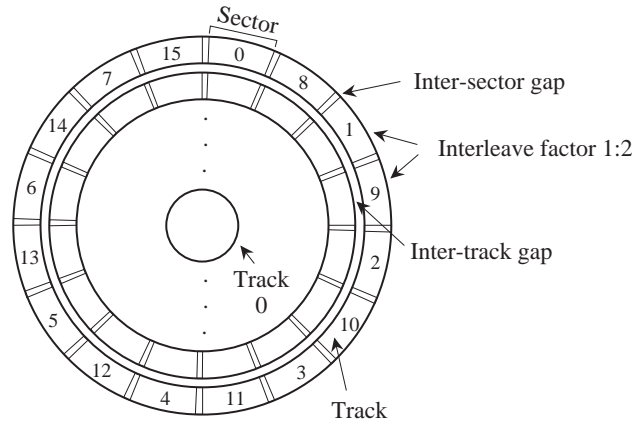


Figure 8-18 Organization of a disk platter with a 1:2 interleave factor.

spaced apart by **inter-track gaps**, which simplify positioning of the head. A set of corresponding tracks on all of the surfaces forms a **cylinder**. For instance, track 0 on each of surfaces 0, 1, 2, 3, 4, and 5 in Figure 8-16 collectively form cylinder 0. The number of bytes per sector is generally invariant across the entire platter.

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

Disk drive capacities and speeds

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

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

A high-capacity disk drive may have $N = 512$ bytes, $S = 1,000$ sectors per track, $T = 5,000$ tracks per surface, and $P = 8$ platters. The total capacity of this drive is

$C = 512 \text{ bytes/sector} \times 1000 \text{ sectors/track} \times 5000 \text{ tracks/surface} \times 8 \text{ platters} \times 2 \text{ surfaces/platter}$ or 38 GB.

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

Head seek time is the largest contributor to overall access time of a disk. Manufacturers usually specify an average seek time, which is roughly the time required for the head to travel half the distance across the disk. The rationale for this definition is that it is difficult to know, *a priori*, which track the data is on, or where the head is positioned when the disk access request is made. Thus it is assumed that the head will, on average, be required to travel over half the surface before arriving at the correct track. On modern disk drives average seek time is approximately 10 ms.

Once the head is positioned at the correct track, it is again difficult to know ahead of time how long it will take for the desired sector to appear under the head. Therefore the average rotational latency is taken to be 1/2 the time of one complete revolution, which is on the order of 4-8 ms. The sector transfer time is just the time for one complete revolution divided by the number of sectors per track. If large amounts of data are to be transferred, then after a complete track is transferred, the head must move to the next track. The parameter of interest here is the track-to-track access time, which is approximately 2 ms (notice that the time for the head to travel past multiple tracks is much less than 2 ms per track). An important parameter related to the sector transfer time is the **burst rate**, the rate at which data streams on or off the disk once the read/write operation has started. The burst rate equals the disk speed in revolutions per second times the capacity per track. This is not necessarily the same as the transfer rate, because there is a setup time needed to position the head and synchronize timing for each sector.

The maximum transfer rate computed from the factors above may not be realized in practice. The limiting factor may be the speed of the bus interconnecting the disk drive and its interface, or it may be the time required by the CPU to transfer the data between the disk and main memory. For example, disks that

operate with the **Small Computer Systems Interface** (SCSI) standards have a transfer rate between the disk and a host computer of from 5 to 40 MB/second, which may be slower than the transfer rate between the head and the internal buffer on the disk. Disk drives contain internal buffers that help match the speed of the disk with the speed of transfer from the disk unit to the host computer.

Disk drives are delicate mechanisms.

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

Smoke particles from cigarette ash are 10 μm or larger, and so smoking should not take place when disks are exposed to the environment. Disks are usually assembled into sealed units in **clean rooms**, so that virtually no large particles are introduced during assembly. Unfortunately, materials used in manufacturing (such as glue) that are internal to the unit can deteriorate over time and can generate particles large enough to cause a head crash. For this reason, sealed disks (formerly called **Winchester** disks) contain filters that remove particles generated within the unit and that prevent particulate matter from entering the drive from the external environment.

Floppy disks

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

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

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

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

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

Disk file systems

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

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

A related facet in disk organization is **interleaving**. If the CPU and interface circuitry between the disk unit and the CPU all keep pace with the internal rate of

the disk, then there may still be a hidden performance problem. After a sector is read and buffered, it is transferred to the CPU. If the CPU then requests the next contiguous sector, then it may be too late to read the sector without waiting for another revolution. If the sectors are interleaved, for example if a file is stored on alternate sectors, say 2, 4, 6, *etc.*, then the time required for the intermediate sectors to pass under the head may be enough time to set up the next transfer. In this scenario, two or more revolutions of the disk are required to read an entire track, but this is less than the revolution per sector that would otherwise be needed. If a single sector time is not long enough to set up the next read, than a greater interleave factor can be used, such as 1:3 or 1:4. In Figure 8-18, an interleave factor of 1:2 is used.

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

Figure 8-19 shows one version of an MCB. Not all systems keep all of this information in the MCB, but it has to be kept *somewhere*, and some of it may even be kept in part of the file itself. There are four major components to the MCB. The Preamble section specifies information relating to the physical layout of the disk, such as the number of surfaces, number of sectors per surface, *etc.* The Files section cross references file names with the list of sectors of which they are composed, and file attributes such as the file creation date, last modification date, the identification of the owner, and protections. Only the starting sector is needed for a fixed file size disk, otherwise, a list of all of the sectors that make up a file is maintained.

The Free blocks section lists the positions of blocks that are free to be used for new or growing files. The Bad blocks section lists positions of blocks that are free but produce **checksums** (see Section 9.4.3) that indicate errors. The bad blocks are thus unused.

As a file grows in size, the operating system reads the MCB to find a free block, and then updates the MCB accordingly. Unfortunately, this generates a great deal of head movement since the MCB and free blocks are rarely (if ever) on the same

Preamble	{	No. surfaces on disk = 4																																																																																							
		No. tracks/surface = 814																																																																																							
		No. sectors/track = 32																																																																																							
		No. bytes/sector = 512																																																																																							
		Interleave factor = 1:3																																																																																							
Starting sector, or sector list																																																																																									
Files	{	<table border="0" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; border-right: 1px solid black; padding: 2px 10px;">Filename</th> <th style="text-align: left; border-right: 1px solid black; padding: 2px 10px;">Surface</th> <th style="text-align: left; border-right: 1px solid black; padding: 2px 10px;">Track</th> <th style="text-align: left; padding: 2px 10px;">Sector</th> <th style="text-align: left; padding: 2px 10px;">Creation Date</th> <th style="text-align: left; padding: 2px 10px;">Last Modified</th> <th style="text-align: left; padding: 2px 10px;">Owner</th> <th style="text-align: left; padding: 2px 10px;">Protec- tions</th> </tr> </thead> <tbody> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;">xyz.p</td> <td style="border-right: 1px solid black; padding: 2px 10px;">1</td> <td style="border-right: 1px solid black; padding: 2px 10px;">10</td> <td style="padding: 2px 10px;">5</td> <td style="padding: 2px 10px;">11/14/93 10:30:57</td> <td style="padding: 2px 10px;">11/14/93 19:30:57</td> <td style="padding: 2px 10px;">16</td> <td style="padding: 2px 10px;">RWX by Owner</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="padding: 2px 10px;">7</td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> </tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="padding: 2px 10px;">4</td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> </tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;">ab.c</td> <td style="border-right: 1px solid black; padding: 2px 10px;">1</td> <td style="border-right: 1px solid black; padding: 2px 10px;">10</td> <td style="padding: 2px 10px;">8</td> <td style="padding: 2px 10px;">8/18/93 16:03:12</td> <td style="padding: 2px 10px;">1/21/94 14:45:03</td> <td style="padding: 2px 10px;">20</td> <td style="padding: 2px 10px;">RX - All W-Owner</td> </tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="padding: 2px 10px;">2</td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> </tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="padding: 2px 10px;">0</td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> </tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="padding: 2px 10px;">⋮</td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> </tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="padding: 2px 10px;">⋮</td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> </tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="border-right: 1px solid black; padding: 2px 10px;"></td> <td style="padding: 2px 10px;">⋮</td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> <td style="padding: 2px 10px;"></td> </tr> </tbody> </table>	Filename	Surface	Track	Sector	Creation Date	Last Modified	Owner	Protec- tions	xyz.p	1	10	5	11/14/93 10:30:57	11/14/93 19:30:57	16	RWX by Owner				7								4					ab.c	1	10	8	8/18/93 16:03:12	1/21/94 14:45:03	20	RX - All W-Owner				2								0								⋮								⋮								⋮											
		Filename	Surface	Track	Sector	Creation Date	Last Modified	Owner	Protec- tions																																																																																
		xyz.p	1	10	5	11/14/93 10:30:57	11/14/93 19:30:57	16	RWX by Owner																																																																																
					7																																																																																				
					4																																																																																				
		ab.c	1	10	8	8/18/93 16:03:12	1/21/94 14:45:03	20	RX - All W-Owner																																																																																
					2																																																																																				
					0																																																																																				
					⋮																																																																																				
					⋮																																																																																				
			⋮																																																																																						
Free blocks	{			1	1	0																																																																																			
				1	1	1																																																																																			
				1	2	5																																																																																			
				⋮	⋮	⋮																																																																																			
Bad blocks	{			1	1	3																																																																																			
				2	5	7																																																																																			
				⋮	⋮	⋮																																																																																			

Figure 8-19 Simplified example of an MCB.

track. A solution that is used in practice is to copy the MCB to main memory and make updates there, and then periodically update the MCB on the disk, which is known as **syncing** the disk.

A problem with having two copies of the MCB, one on the disk and one in main memory, is that if a computer is shut down before the main memory version of the MCB is synced to the disk, then the integrity of the disk is destroyed. The normal shutdown procedure for personal computers and other machines syncs the disks, so it is important to shut down a computer this way rather than by simply shutting off the power. In the event that a disk is not properly synced, there is usually enough circumstantial information for a disk recovery program to restore the integrity of the disk, often with the help of a user. (Note: See problem 8.12 at the end of the chapter for an alternative MCB organization that makes recovery easier.)

8.5.2 MAGNETIC TAPE

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

Information is stored on a tape in two-dimensional fashion, as shown in Figure 8-20. Bits are stored across the width of the tape in **frames** and along the length

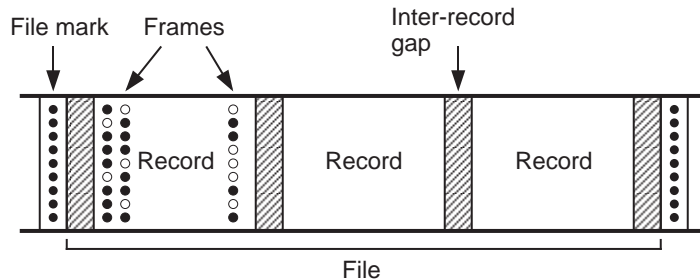


Figure 8-20 A portion of a magnetic tape (adapted from [Hamacher, 1990]).

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

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

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

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

8.5.3 MAGNETIC DRUMS

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

The configuration of a drum is shown in Figure 8-21. The outer surface of the

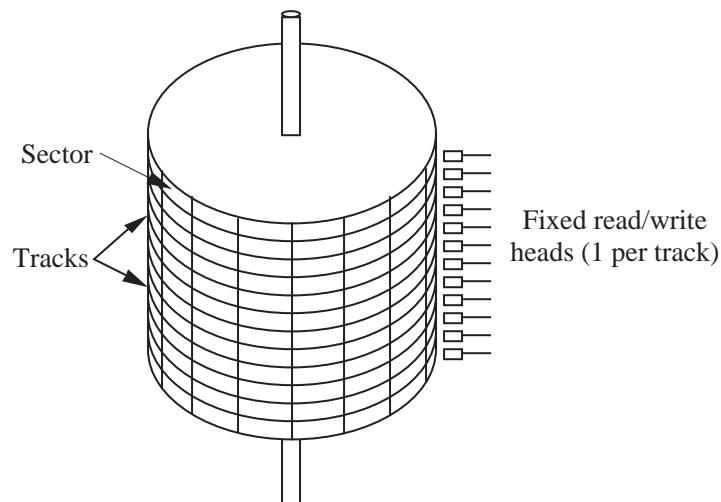


Figure 8-21 A magnetic drum unit.

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

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

8.5.4 OPTICAL DISKS

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

The Compact Disc

The CD was introduced in 1983 as a medium for playback of music. CDs have the capacity to store 74 minutes of audio, in digital stereo (2-channel) format. The audio is sampled at $2 \times 44,000$ 16-bit samples per second, or nearly 700 MB capacity. Since the introduction of the CD in 1983, CD technology has improved in terms of price, density, and reliability, which led to the development of **CD ROMs** (CD read only memories) for computers, which also have the same 700 MB capacity. Their low cost, only a few cents each when produced in volume, coupled with good reliability and high capacity, have made CD ROMs the medium of choice for distributing commercial software, replacing floppy disks.

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

Unlike a magnetic disk in which all of the sectors on concentric tracks are lined up like a sliced pie (where the disk rotation uses **constant angular velocity**), a CD is arranged in a spiral format (using **constant linear velocity**) as shown in Figure 8-22. The pits are laid down on this spiral with equal spacing from one

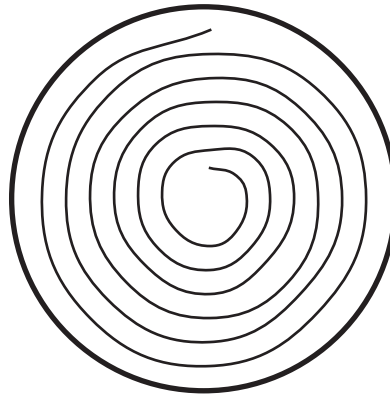


Figure 8-22 Spiral storage format for a CD.

end of the disk to the other. The speed of rotation, originally the same 30 RPM as the floppy disk, is adjusted so that the disk moves more slowly when the head is at the edge than when it is at the center. Thus CD ROMs suffer from the same long access time as floppy disks because of the high rotational latency. CD ROM drives are available with rotational speeds up to 24 \times , or 24 times the rotational speed of an audio CD, with a resulting decrease in average access time.

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

The Digital Versatile Disc

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

EXAMPLE: TRANSFER TIME FOR A HARD DISK

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

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

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

Solution:

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

The time to read or write a sector is simply the time it takes for the sector to pass under the head, which is $(16 \text{ ms/track}) \times (1/32 \text{ tracks/sector}) = .5 \text{ ms/sector}$. For this example, the head movement time is only 2 ms because the head moves between adjacent tracks. After reading sector 1 on track 12, which takes .5 ms,

an additional 15.5 ms of rotational delay is needed for the head to line up with sector 1 again. The head movement time of 2 ms overlaps the 15.5 ms of rotational delay, and so only the greater of the two times (15.5 ms) is used.

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

The time to transfer all of track 12 to track 13 is computed in a similar manner. The memory buffer can hold an entire track, and so the time to read or write an entire track is simply the rotational delay for a track, which is 16 ms. The head movement time is 2 ms, which is also the time for four sectors to pass under the head (at .5 ms per sector). Thus, after reading a track and repositioning the head, the head is now on track 13, at four sectors past the initial sector that was read on track 12.

Sectors can be written in a different order than they are read. Track 13 can thus be written starting with sector 5. The time to write track 13 is 16 ms, and the time for the entire transfer then is: $16 \text{ ms} + 2 \text{ ms} + 16 \text{ ms} = 34 \text{ ms}$. Notice that the rotational delay is zero for this example because the head lands at the beginning of the first sector to be written. ■

8.6 Input Devices

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

8.6.1 KEYBOARDS

Keyboards are used for manual input to a computer. A keyboard layout using the ECMA-23 Standard (2nd ed.) is shown in Figure 8-23. The “QWERTY” layout (for the upper left row of letters D01 – D06) conforms to the traditional layout used in typewriters. Frequently used letters are placed far apart so that the typist is slowed and jams in mechanical typewriters are reduced. Although jams are not a problem with electronic keyboards, the traditional layout prevails.

When a character is typed, a bit pattern is created that is transmitted to a host computer. For 7-bit ASCII characters, only 128 bit patterns can be used, but many keyboards that expand on the basic ECMA-23 standard use additional

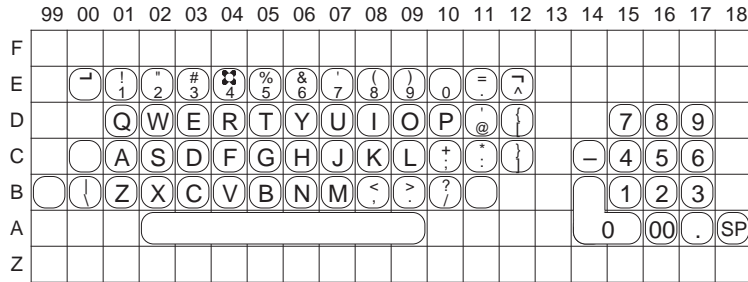


Figure 8-23 Keyboard layout for the ECMA-23 Standard (2nd ed.). Shift keys are frequently placed in the B row.

modifier keys (shift, escape, and control) and so a seven-bit pattern is no longer large enough. A number of alternatives are possible, but one method that has gained acceptance is to provide one bit pattern for each modifier key and other bit patterns for the remaining keys.

Other modifications to the ECMA-23 keyboard include the addition of function keys (in row F, for example), and the addition of special keys such as tab, delete, and carriage return. A modification that places frequently used keys together was developed for the **Dvorak keyboard** as shown in Figure 8-24. Despite the perfor-

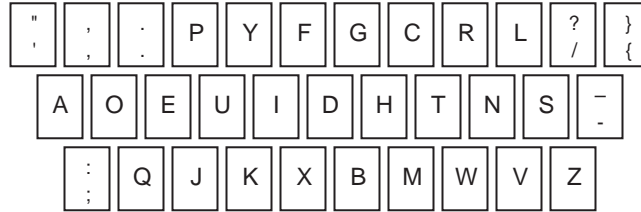


Figure 8-24 The Dvorak keyboard layout.

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

8.6.2 BIT PADS

A **digitizing tablet**, or **bit pad**, is an input device that consists of a flat surface and a **stylus** or **puck** as illustrated in Figure 8-25. The tablet has an embedded two-dimensional mesh of wires that detects an induced current created by the puck as it is moved about the tablet. The bit pad transmits X-Y (horizontal-vertical) positions and the state of the buttons on the puck (or stylus) either continuously, or for an event such as a key click or a movement, depending on the control method. Bit pads are commonly used for entering data from maps, pho-

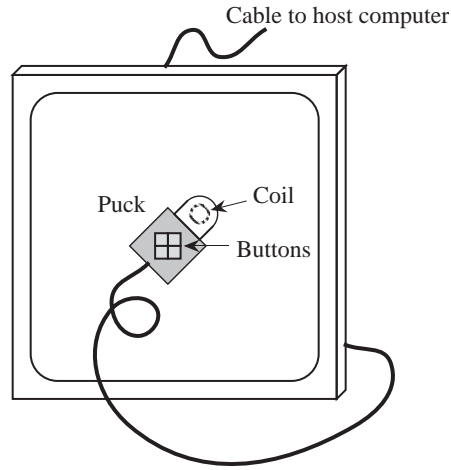


Figure 8-25 A bit pad with a puck.

tographs, charts, or graphs.

8.6.3 MICE AND TRACKBALLS

A **mouse** is a hand-held input device that consists of a rubber ball on the bottom and one or more buttons on the top as illustrated in the left side of Figure 8-26.

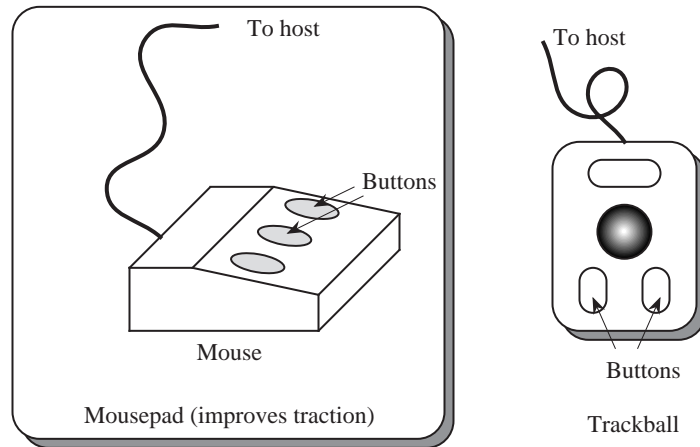


Figure 8-26 A three-button mouse (left) and a three-button trackball (right).

As the mouse is moved, the ball rotates proportional to the distance moved. Potentiometers within the mouse sense the direction of motion and the distance

traveled, which are reported to the host along with the state of the buttons. Two button events are usually distinguished: one for the key-down position and one for the key-up position.

A **trackball** can be thought of as a mouse turned upside down. The trackball unit is held stationary while the ball is manually rotated. The configuration of a trackball is shown in the right side of Figure 8-26.

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

8.6.4 LIGHTPENS AND TOUCHSCREENS

Two devices that are typically used for selecting objects are **lightpens** and **touchscreens**. A lightpen does not actually produce light, but senses light from a video screen as illustrated in Figure 8-27. An electron beam excites a phosphor coating

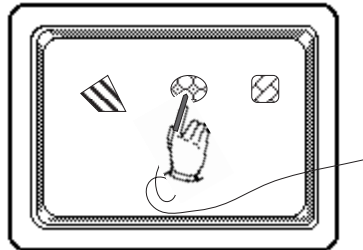


Figure 8-27 A user selecting an object with a lightpen.

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

When a dim spot is refreshed, it becomes brighter, and this change in intensity signals the location of the beam at a particular time. If the lightpen is positioned

at a location where the phosphor is refreshed, then the position of the electron beam locates the position of the pen. Since the lightpen detects intensity, it can only distinguish among illuminated areas. Dark areas of the screen all appear the same since there is no change in intensity over time.

A touchscreen comes in two forms, photonic and electrical. An illustration of the photonic version is shown in Figure 8-28. A matrix of beams covers the screen in

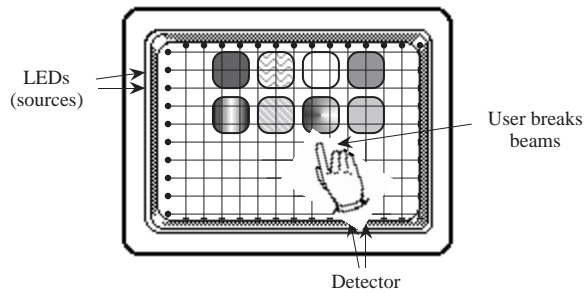


Figure 8-28 A user selecting an object on a touchscreen.

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

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

8.6.5 JOYSTICKS

A **joystick** indicates horizontal and vertical position by the distance a rod that protrudes from the base is moved (see Figure 8-29). Joysticks are commonly used

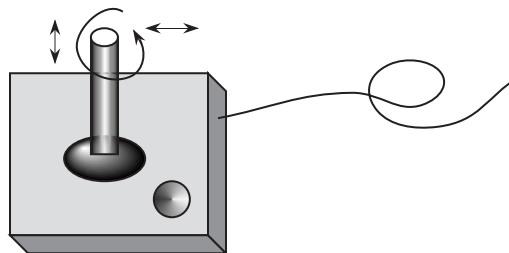


Figure 8-29 A joystick with a selection button and a rotatable rod.

in video games, and for indicating position in graphics systems. Potentiometers within the base of the joystick translate X-Y position information into voltages, which can then be encoded in binary for input to a digital system. In a spring-loaded joystick, the rod returns to the center position when released. If the rod can be rotated, then an additional dimension can be indicated, such as height.

8.7 Output Devices

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

8.7.1 LASER PRINTERS

A laser printer consists of a charged drum in which a laser discharges selected areas according to a bit mapped representation of a page to be printed. As the drum advances for each scan line, the charged areas pick up electrostatically sensitive toner powder. The drum continues to advance, and the toner is transferred to the paper, which is heated to fix the toner on the page. The drum is cleaned of any residual toner and the process repeats for the next page. A schematic diagram of the process is shown in Figure 8-30.

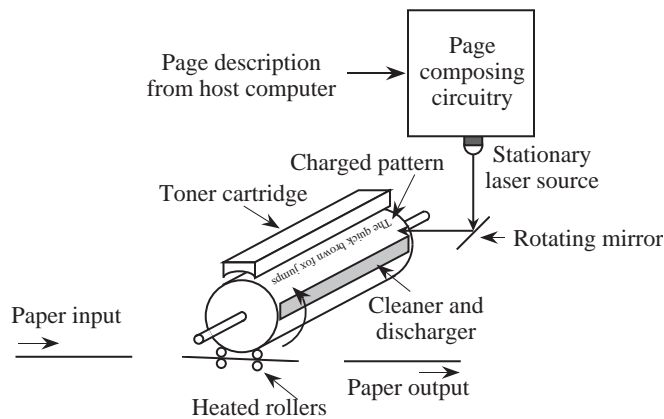


Figure 8-30 Schematic of a laser printer (adapted from [Tanenbaum, 1999]).

Since the toner is a form of plastic, rather than ink, it is not absorbed into the page but is melted onto the surface. For this reason, a folded sheet of laser printed paper will display cracks in the toner along the fold, and the toner is

sometimes unintentionally transferred to other materials if exposed to heat or pressure (as from a stack of books).

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

8.7.2 VIDEO DISPLAYS

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

A configuration for a CRT is shown in Figure 8-31. An electron gun generates a

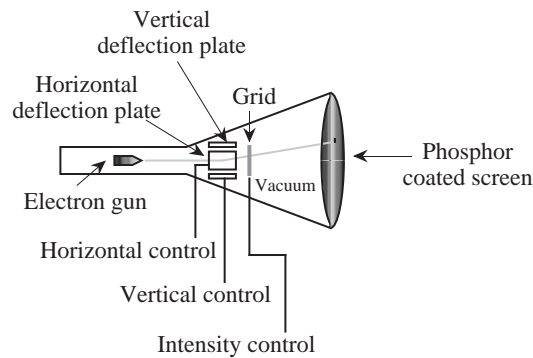


Figure 8-31 A CRT with a single electron gun.

stream of electrons that is imaged onto a phosphor coated screen at positions controlled by voltages on the vertical and horizontal deflection plates. Electrons are negatively charged, and so a positive voltage on the grid accelerates electrons toward the screen and a negative voltage repels electrons away from the screen. The color produced on the screen is determined by the characteristics of the phosphor. For a color CRT, there are usually three different phosphor types (red,

green, and blue) that are interleaved in a regular pattern, and three guns, which produce three beams that are simultaneously deflected on the screen.

A simple display controller for a CRT is shown in Figure 8-32. The writing of

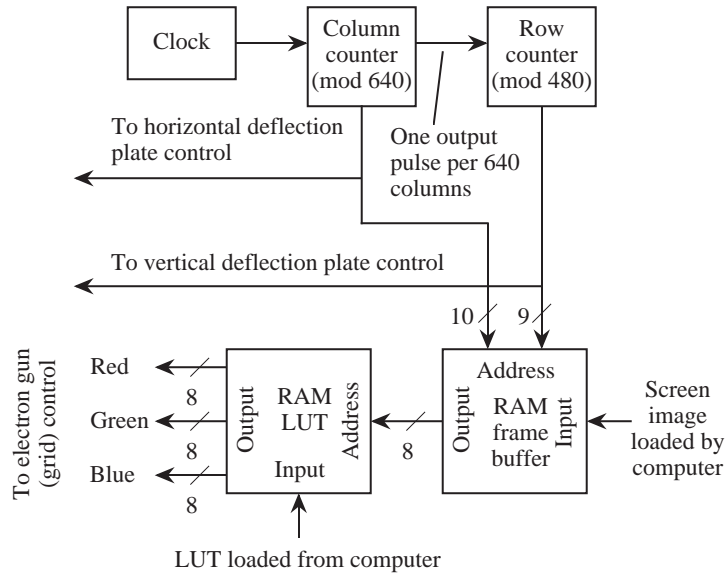


Figure 8-32 Display controller for a 640×480 color monitor (adapted from [Hamacher *et al.*, 1990]).

information on the screen is controlled by the “dot clock,” which generates a continuous stream of alternating 1’s and 0’s at a rate that corresponds to the update time for a single spot on the screen. A single spot is called a **picture element**, or **pixel**. The display controller in Figure 8-32 is for a screen that is 640 pixels wide by 480 pixels high. A column counter is incremented from 0 to 639 for each row, then repeats, and a row counter is incremented from 0 to 479, which then repeats. The row and column addresses index into the **frame buffer**, or “display RAM” that holds the bit patterns corresponding to the image that is to be displayed on the screen. The contents of the frame buffer are transferred to the screen from 30 to 100 times per second. This technique of mapping a RAM area to the screen is referred to as **memory-mapped video**. Each pixel on the screen may be represented by from 1 to 12 or more bits in the frame buffer. When there is only a single bit per pixel, the pixel can only be on or off, black or white; multiple bits per pixel allow a pixel to have varying colors, or shades of gray.

Each pixel in the display controller of Figure 8-32 is represented by eight bits in the frame buffer, which means that one out of $2^8 = 256$ different intensities can be used for each pixel. In a simple configuration the eight bits can be partitioned for the red, green, and blue (R, G, and B) inputs to the CRT as three bits for red, three bits for green, and two bits for blue. An alternative is to pass the eight-bit pixel value to a color lookup table (LUT) in which the eight-bit value is translated into 256 different 24-bit colors. Eight bits of the 24-bit output are then used for each of the red, green, and blue guns. A total of 2^{24} different colors can be displayed, but only 2^8 of the 2^{24} can be displayed at any one time since the LUT has only 2^8 entries. The LUT can be reloaded as necessary to select different subsets of the 2^{24} colors. For example, in order to display a gray scale image (no color), we must have $R=G=B$ and so a ramp from 0 to 255 is stored for each of the red, green, and blue guns.

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

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

The data rates between computer and video monitor can be quite high. Consider that a 24-bit per pixel monitor with 1024×768 pixel resolution and a refresh rate of 60 Hz will require a **bandwidth** (that is, the amount of information that can be carried over a given period of time) of 3 bytes/pixel \times (1024 \times 768) pixels

× 60 Hz, or roughly 140 MB per second. Fortunately, the hardware described above maps the frame buffer onto the screen without CPU intervention, but it is still up to the CPU to output pixels to the frame buffer when the image on the screen changes.

■ SUMMARY

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

A bus provides a fixed bandwidth that is shared among a number of devices. A hierarchy of busses can be interconnected with bridges, so that independent transfers can take place simultaneously. From a programmer's perspective, communication is handled via programmed I/O, interrupt driven I/O, or DMA.

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

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

■ FURTHER READING

Intel data sheets and other literature, including the Pentium, Pentium II, and Pentium Pro hardware and programmer's manuals can be ordered from Intel Literature Sales, PO Box 7641, Mt. Prospect IL 60056-7641, or, in the U. S. and Canada, by calling (800) 548-4725. The source of the material on the dual Pentium II Xeon configuration is <http://www.intel.com/design/chipsets/440zx>.

(Hamacher *et al.*, 1990) provides explanations of communication devices and a number of peripherals such as an alphanumeric CRT controller. (Tanenbaum,

1999) and (Stallings, 1996) also give readable explanations of peripheral devices. The material on synchronous and asynchronous busses, and bus arbitration, is influenced by a presentation in (Tanenbaum, 1999).

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

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

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

■ PROBLEMS

- 8.1** What is the minimum time needed to transfer 100 MB from the Audio device to a Pentium II, using the bridge architecture and parameters shown in Figure 8-7?
- 8.2** Why must the CPU ensure that interrupts are disabled before handing control over to the interrupt service routine?
- 8.3** Show the Manchester encoding for the bit sequence: 10011101.
- 8.4** A disk that has 16 sectors per track uses an interleave factor of 1:4. What is the smallest number of revolutions of the disk required to read all of the sectors of a track in sequence?
- 8.5** A hard magnetic disk has two surfaces. The storage area on each surface has an inner radius of 1 cm and an outer radius of 5 cm. Each track holds the same number of bits, even though each track differs in size from every other. The maximum storage density of the media is 10,000 bits/cm. The spacing between corresponding points on adjacent tracks is .1 mm, which includes the inter-track gap. Assume that the inter-sector gaps are negligible, and that a track exists on each edge of the storage area.
- (a) What is the maximum number of bits that can be stored on the disk?
- (b) What is the data transfer rate from the disk to the head in bits per second

at a rotational speed of 3600 RPM?

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

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

(a) Compute the time to read a track (*not* the time to transmit the track to a host). Assume that interleaving is not used.

(b) What is the minimum time required to read the entire disk pack to a CPU, given the best of all possible circumstances? Assume that the head of the first surface to be read is positioned at the beginning of the first sector of the first track, and that an entire cylinder is read before the arm is moved. Also assume that the disk unit can buffer an entire cylinder, but no more. During operation, the disk unit first fills its buffer, then it empties it to the CPU, and only then does it read more of the disk.

8.8 A fixed head disk has one head per track. The heads do not move, and thus, there is no head movement component in calculating the access time. For this problem, calculate the time that it takes to copy one surface to another surface. This is an internal operation to the disk, and does not involve any communication with the host. There are 1000 cylinders, each track holds 10 sectors, and the rotation rate of the disk is 3000 RPM. The sectors all line up with each other. That is, within a cylinder, sector 0 on each track lines up with sector 0 on every other track, and within a surface, sector 0 on each track begins on the same line drawn from the center of the surface to the edge.

An internal buffer holds a single sector. When a sector is read from one track, it is held in the buffer until it is written onto another track. Only then can

another sector be read. It is not possible to simultaneously read and write the buffer, and the buffer must be entirely loaded or entirely emptied – partial reads or writes are not allowed. Calculate the minimum time required to copy one surface to another, given the best starting conditions. The surfaces must be direct images of each other. That is, sector i in the source surface must be directly above or below sector i in the destination surface.

- 8.9** Compute the storage capacity of a 6250 byte per inch (BPI) tape that is 600 ft long and has a record size of 2048 bytes. The size of an inter-record gap is .5 in.
- 8.10** A bit mapped display is 1024 pixels wide by 1024 pixels high. The refresh rate is 60 Hz, which means that every pixel is rewritten to the screen 60 times a second, but only one pixel is written at any time. What is the maximum time allowed to write a single pixel?
- 8.11** How many bits need to be stored in the LUT in Figure 8-32? If the LUT is removed, and the RAM is changed to provide the 24-bit R, G, and B output directly, how many additional bits need to be stored in the RAM? Assume that the initial size of the RAM is $2^{10} \times 2^9 = 2^{19}$ words \times 8 bits/word.
- 8.12** The MCB as presented in Section 8.2.1 keeps track of every sector on the disk. An alternative organization, which significantly reduces the size of the MCB, is to store blocks in **chains**. The idea is to store only the first block of a file in the MCB, and then store a pointer to the succeeding block at the end of the first block. Each succeeding block is linked in a similar manner.
- (a) How does this approach affect the time to access the middle of a file?
- (b) After a system crash, would a disk recovery be easier if only the first sector of a file is stored in the MCB, and the remaining list of sectors is stored in a header at the beginning of each file? How does this approach affect storage?
- 8.13** You are now the administrator for a computer system that is maintained by Mega Equipment Corporation (MEC). As part of routine maintenance, MEC realigns the heads on one of the disks, and now the disk cannot be read or written without producing errors. What went wrong? Would this happen with or without the use of a timing track?

