# Get on the Bus

The processor is certainly the most important component of a computer, but it's not the only component. A computer also requires random access memory (RAM) that contains machine-code instructions for the processor to execute. The computer must also include some way for those instructions to get into RAM (an input device) and some way for the results of the program to be observed (an output device). As you'll also recall, RAM is volatile—it loses its contents when the power is turned off. So another useful component of a computer is a long-term storage device that can retain code and data when the computer is turned off.

All the integrated circuits that make up a complete computer must be mounted on circuit boards. In some smaller machines, all the ICs can fit on a single board. But it's more usual for the various components of the computer to be divided among two or more boards. These boards communicate with each other by means of a *bus*. A bus is simply a collection of digital signals that are provided to every board in a computer. These signals fall into four categories:

- Address signals. These are signals generated by the microprocessor and used mostly to address random access memory. But they're also used to address other devices attached to the computer.
- Data Output signals. These also are signals provided by the microprocessor. They're used to write data to RAM or to other devices. Be careful with the terms *input* and *output*. A data output signal from the microprocessor becomes a data input signal to RAM and other devices.
- Data Input signals. These are signals that are provided by other parts of the computer and are read by the microprocessor. The

data input signals most often originate in RAM output; this is how the microprocessor reads the contents of memory. But other components also provide data input signals to the microprocessor.

- Control signals. These are miscellaneous signals that usually correspond to the control signals of the particular microprocessor around which the computer is built. Control signals may originate in the microprocessor or from other devices to signal the microprocessor. An example of a control signal is the signal used by the microprocessor to indicate that it needs to write some data output into a particular memory address.

In addition, the bus supplies power to the various boards that the computer comprises.

One of the earliest popular busses for home computers was the S-100 bus, which was introduced in 1975 in the first home computer, the MITS Altair. Although this bus was based on the 8080 microprocessor, it was later adapted to other processors such as the 6800. An S-100 circuit board is 5.3 inches by 10 inches. One edge of the circuit board fits into a socket that has 100 connectors (hence the name S-100).

An S-100 computer contains a larger board called a *motherboard* (or *main board*) that contains a number of S-100 sockets (perhaps 12 of them) wired to one another. These sockets are sometimes called *expansion slots*. The S-100 circuit boards (also called *expansion boards*) fit into these sockets. The 8080 microprocessor and support chips (some of which I mentioned in Chapter 19) occupy one S-100 board. Random access memory occupies one or more other boards.

Because the S-100 bus was designed for the 8080 chip, it has 16 address signals, 8 data input signals, and 8 data output signals. (As you'll recall, the 8080 itself combines the data input and data output signals. These signals are divided into separate input and output signals by other chips on the circuit board that contains the 8080.) The bus also includes 8 *interrupt* signals. These are signals generated by other devices when they need immediate attention from the CPU. For example (as we'll see later in this chapter), a keyboard might generate an interrupt signal when a key is pressed. A short program run by the 8080 can then determine what that key was and take some action. The board containing the 8080 also generally includes a chip called the Intel 8214 Priority Interrupt Control Unit to handle these interrupts. When an interrupt occurs, this chip generates an interrupt signal to the 8080. When the 8080 acknowledges the interrupt, the chip provides a *RST* (*Restart*) instruction that causes the microprocessor to save the current program counter and branch to address 0000h, 0008h, 0010h, 0018h, 0020h, 0028h, 0030h, or 0038h depending on the interrupt.

If you were designing a new computer system that included a new type of bus, you could choose whether to publish (or otherwise make available) the specifications of the bus or to keep them secret.

If the specifications of a particular bus are made public, other manufacturers—so-called *third-party* manufacturers—can design and sell expansion

boards that work with that bus. The availability of these additional expansion boards makes the computer more useful and hence more desirable. More sales of the computer create more of a market for more expansion boards. This phenomenon is the incentive for designers of most small computer systems that adhere to the principle of *open architecture,* which allows other manufacturers to create peripherals for the computer. Eventually, a bus might be considered an industry-wide *standard*. Standards have been an important part of the personal computer industry.

The most famous open architecture personal computer was the original IBM PC introduced in the fall of 1981. IBM published a *Technical Reference* manual for the PC that contained complete circuit diagrams of the entire computer, including all the expansion boards that IBM manufactured for it. This manual was an essential tool that enabled many manufacturers to make their own expansion boards for the PC and, in fact, to create entire *clones* of the PC—computers that were nearly identical to IBM's and ran all the same software.

The descendants of that original IBM PC now account for about 90 percent of the market in the desktop computers. Although IBM itself has only a small share of this market, it could very well be that IBM's share is larger than if the original PC had a *closed architecture* with a *proprietary* design. The Apple Macintosh was originally designed with a closed architecture, and despite occasional flirtations with open architecture, that original decision possibly explains why the Macintosh currently accounts for less than 10 percent of the desktop market. (Keep in mind that whether a computer system is designed under the principle of open architecture or closed architecture doesn't affect the ability of other companies to write *software* that runs on the computer. Only the manufacturers of certain video games have restricted other companies from writing software for their systems.)

The original IBM PC used the Intel 8088 microprocessor, which can address 1 megabyte of memory. Although internally the 8088 is a 16-bit microprocessor, externally it addresses memory in 8-bit chunks. The bus that IBM designed for the original PC is now called the ISA (Industry Standard Architecture) bus. The expansion boards have 62 connectors. The signals include 20 address signals, 8 combined data input and output signals, 6 interrupt requests, and 3 *direct memory access* (DMA) requests. DMA allows storage devices (which I'll describe toward the end of this chapter) to perform more quickly than would otherwise be possible. Normally, the microprocessor handles all reading from and writing to memory. But using DMA, another device can bypass the microprocessor by taking over the bus and reading from or writing to memory directly.

In an S-100 system, all components are mounted on expansion boards. In the IBM PC, the microprocessor, some support chips, and some RAM are located on what IBM called the *system board* but which is also often called a motherboard or a main board.

In 1984, IBM introduced the Personal Computer AT, which used the 16-bit Intel 80286 microprocessor that can address 16 megabytes of memory. IBM retained the existing bus but added another 36-connector socket that included

7 more address signals (although only 4 more were needed), 8 more data input and output signals, 5 more interrupt requests, and 4 more DMA requests.

Busses need to be upgraded or replaced when microprocessors outgrow them, either in data width (from 8 to 16 to 32 bits) or in the number of address signals they output. But microprocessors also outgrow busses when they achieve faster speeds. Early busses were designed for microprocessors operating at a clock speed of several megahertz rather than several hundred megahertz. When a bus isn't properly designed for high speeds, it can give off radio frequency interference (RFI) that causes static or other noise on nearby radios and television sets.

In 1987, IBM introduced the Micro Channel Architecture (MCA) bus. Some aspects of this bus had been patented by IBM, so IBM was able to collect licensing fees from other companies that used the bus. Perhaps for this reason, the MCA bus did *not* become an industry standard. Instead, in 1988 a consortium of nine companies (not including IBM) countered with the 32-bit EISA (Extended Industry Standard Architecture) bus. More recently, the Intel-designed Peripheral Component Interconnect (PCI) bus has become common in PC-compatibles.

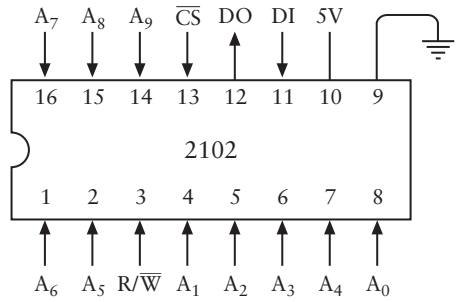
To understand how the various components of the computer work, it's again helpful to return to that earlier and simpler era of the mid-1970s. We might imagine that we're designing boards for the Altair, or perhaps for an 8080 or 6800 computer of our own design. We probably want to design some memory for the computer and to have a keyboard for input, a TV set for output, and perhaps some way to save the contents of memory when we turn off the computer. Let's look at the various *interfaces* we can design to add these components to our computer.

You'll recall from Chapter 16 that RAM arrays have address inputs, data inputs, data outputs, and a signal used to write data into memory. The number of address inputs indicates the number of separate values that can be stored in the RAM array:

$$\text{Number of values in RAM array} = 2^{\text{Number of address inputs}}$$

The number of data input and output signals indicates the size of the stored values.

One popular memory chip for home computers in the mid-1970s was the 2102:

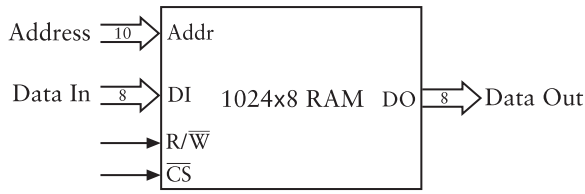


The 2102 is a member of the MOS (metal-oxide semiconductor) family of semiconductors, which is the same technology used for the 8080 and 6800 microprocessors themselves. MOS semiconductors can be easily connected to TTL chips; they generally have a higher density of transistors than TTL but aren't as fast.

As you can probably figure out by counting the address signals ($A_0$ through $A_9$) and noting the single data output (DO) and data input (DI) signals, this chip stores 1024 bits. Depending on the type of 2102 chip you're using, the *read access time*—the time it takes for the data output to be valid after a particular address has been applied to the chip—ranges from 350 to 1000 nanoseconds. The R/$\overline{W}$ (*read/write*) signal is normally 1 when you're reading memory. When you want to write data into the chip, this signal must be 0 for a period of at least 170 to 550 nsec, again depending on the type of 2102 chip you're using.

Of particular interest is the $\overline{CS}$ signal, which stands for *chip select*. When this signal is 1, the chip is *deselected,* which means that it doesn't respond to the R/$\overline{W}$ signal. The $\overline{CS}$ signal has another profound effect on the chip, however, that I'll describe shortly.

Of course, if you're putting together memory for an 8-bit microprocessor, you want to organize this memory so that it stores 8-bit values rather than 1-bit values. At the very least, you'll need to wire 8 of these 2102 chips together to store entire bytes. You can do this by connecting all the corresponding address signals, the R/$\overline{W}$ signals, and the $\overline{CS}$ signals of eight 2102 chips. The result can be drawn like this:




This is a 1024 × 8 RAM array, or 1 KB of RAM.

From a practical perspective, you need to put the memory chips on a circuit board. How many can you fit on one board? Well, if you really cram them close together, you can fit 64 of these chips on a single S-100 board. That will give you 8 KB of memory. But let's go for a more modest 4 KB using just 32 chips. Each set of chips that are wired together to store a whole byte (as illustrated above) is known as a *bank*. A 4-KB memory board contains four banks of 8 chips each.

Eight-bit microprocessors such as the 8080 and 6800 have 16-bit addresses that can address a total of 64 KB of memory. When you wire a 4-KB memory

board containing four banks of chips, the memory board's 16 address signals perform the following functions:

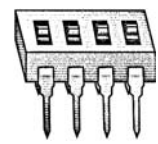
$$\underbrace{A_{15}\ A_{14}\ A_{13}\ A_{12}}_{\text{Select the board}}\ \underbrace{A_{11}\ A_{10}}_{\text{Select the bank}}\ \underbrace{A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0}_{\text{Address the RAM}}$$

The 10 address signals $A_0$ through $A_9$ are directly wired to the RAM chips. The address signals $A_{10}$ and $A_{11}$ select which of the four banks is being addressed. The address signals $A_{12}$ through $A_{15}$ determine which addresses apply to this particular board—in other words, the addresses that the board responds to. The 4-KB memory board we're designing can occupy one of 16 different 4-KB ranges in the entire 64-KB memory space of the microprocessor:

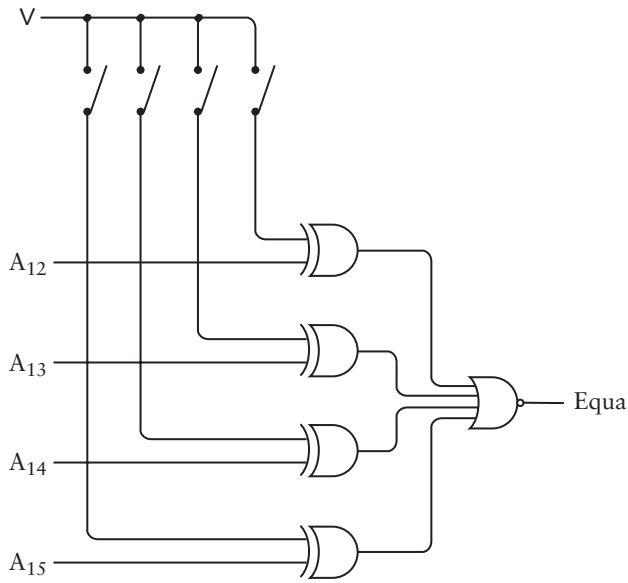
0000h through 0FFFh, or
1000h through 1FFFh, or
2000h through 2FFFh, or
⋮
F000h through FFFFh.

For example, suppose we decide that this 4-KB memory board will apply to addresses A000h through AFFFh. This means that addresses A000h through A3FFh will apply to the first bank of 1-KB chips, addresses A400h through A7FFh to the second bank, addresses A800h through ABFFh to the third bank, and addresses AC00h through AFFFh to the fourth bank.

It's common to wire a 4-KB memory board so that you can flexibly specify at a later time what range of addresses it responds to. To achieve this flexibility, you use something called a *DIP switch*. This is a series of tiny switches (anywhere from 2 through 12) in a dual inline package (DIP) that can be inserted in a normal IC socket:

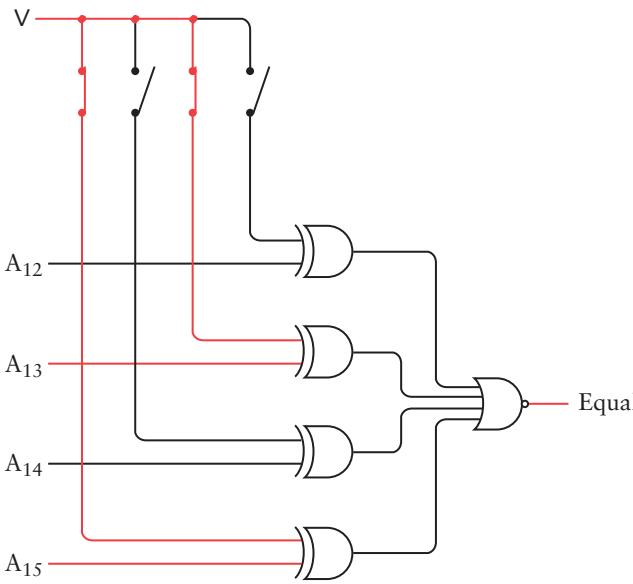


You can wire this switch with the high 4 address bits from the bus in a circuit called a *comparator*.

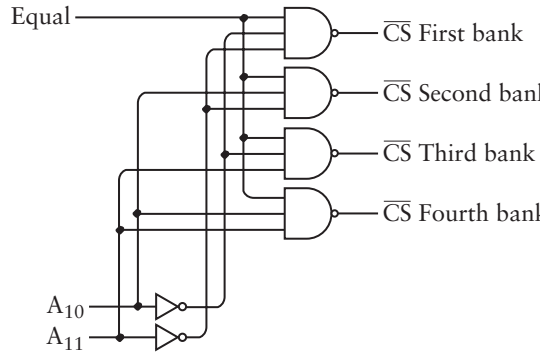



As you'll recall, the output of an XOR gate is 1 if either of the two inputs is 1 but not both. Another way to think of this is that the output of an XOR gate is 0 if the two inputs are the same—either both 0 or both 1.

For example, if we close the switches corresponding to $A_{13}$ and $A_{15}$, that means we want the memory board to respond to memory addresses A000h through AFFFh. When the address signals $A_{12}$, $A_{13}$, $A_{14}$, and $A_{15}$ from the bus are equal to the values set on the switches, the outputs of all four XOR gates are 0, which means the output from the NOR gate is 1:



You can then combine that Equal signal with a 2-Line-to-4-Line Decoder to generate $\overline{CS}$ signals for each of the four banks of memory:




For example, when $A_{10}$ is 0 and $A_{11}$ is 1, that's the third bank.

If you recall the messy details of combining RAM arrays in Chapter 16, you might assume that we also need eight 4-to-1 Selectors to select the correct data output signals from the four banks of memory. But we don't, and here's why.

Normally, the output signals of TTL-compatible integrated circuits are either greater than 2.2 volts (for a logical 1) or less than 0.4 volts (for a logical 0). But what happens if you try connecting outputs? If one integrated circuit has a 1 output and another has a 0 output, and these two outputs are connected, what will result? You can't really tell, and that's why outputs of integrated circuits aren't normally connected together.

The data output signal of the 2102 chip is known as a *3-state,* or *tri-state,* output. Besides a logical 0 and a logical 1, this data output signal can also be a third state. This state is—lo and behold—nothing at all! It's as if nothing is connected to the pin of the chip. The data output signal of the 2102 chip goes into this third state when the $\overline{CS}$ input is 1. This means that we *can* connect the corresponding data output signals of all four banks and use those eight combined outputs as the eight data input signals of the bus.

I'm emphasizing the concept of the tri-state output because it's essential to the operation of a bus. Just about everything that's connected to the bus uses the data input signals of the bus. At any time, only one board connected to the bus should be determining what those data input signals are. The other boards must be connected to the bus with deselected tri-state outputs.

The 2102 chip is known as *static* random access memory, or SRAM (pronounced *ess ram*), to differentiate it from *dynamic* random access memory, or DRAM (pronounced *dee ram*). SRAM generally requires 4 transistors per bit of memory (not quite as many transistors as the flip-flops I used for memory in Chapter 16). DRAM, however, requires only 1 transistor per bit. The drawback of DRAM is that it requires more complex support circuitry.

An SRAM chip such as the 2102 will retain its contents as long as the chip has power. If the power goes off, the chip loses its contents. The DRAM is

also similar in that respect, but a DRAM chip requires also that the contents of the memory be periodically accessed, even if the contents aren't needed. This is called a *refresh* cycle, and it must occur several hundred times per second. It's like periodically nudging someone so that the person doesn't fall asleep.

Despite the hassle of using DRAM, the ever-increasing capacity of DRAM chips over the years has made DRAM the standard. In 1975, Intel introduced a DRAM chip that stored 16,384 bits. In accordance with Moore's Law, DRAM chips have quadrupled in capacity roughly every three years. Today's computers usually have sockets for memory right on the system board. The sockets take small boards called *single inline memory modules* (SIMMs) or *dual inline memory modules* (DIMMs) that contain several DRAM chips. Today you can buy a DIMM containing 128 megabytes of memory for under $300.

Now that you know how to make memory boards, you don't want to fill up the entire memory space of your microprocessor with memory. You want to leave some memory space for your output device.

The *cathode-ray tube* (CRT)—a familiar sight in homes for the last half century in its guise as the television set—has become the most common output device for computers. A CRT attached to a computer is usually known as the *video display,* or *monitor*. The electronic components that provide the signal to the video display are usually known as the *video display adapter*. Often the video display adapter occupies its own board in the computer, which is known as the *video board*.

While the two-dimensional image of a video display or a television might seem complex, the image is actually composed of a single continuous beam of light that sweeps across the screen very rapidly. The beam begins in the upper left corner and moves across the screen to the right, whereupon it zips back to the left to begin the second line. Each horizontal line is known as a *scan line*. The movement back to the beginning of each of these lines is known as the *horizontal retrace*. When the beam finishes the bottom line, it zips from the lower right corner of the screen to the upper left corner (the *vertical retrace*) and the process begins again. For American television signals, this happens 60 times a second, which is known as the *field rate*. It's fast enough so that the image doesn't appear to be flickering.

Television is complicated somewhat by the use of an *interlaced* display. Two fields are required to make up a single *frame,* which is a complete still video image. Each field contributes half the scan lines of the entire frame—the first field has the even scan lines, and the second field has the odd scan lines. The *horizontal scan rate,* which is the rate at which each horizontal scan line is drawn, is 15,750 Hertz. If you divide that number by 60 Hertz, you get 262.5 lines. That's the number of scan lines in one field. An entire frame is double that, or 525 scan lines.

Regardless of the mechanics of interlaced displays, the continuous beam of light that makes up the video image is controlled by a single continuous signal. Although the audio and video components of a television program

are combined when they're broadcast or transmitted through a cable television system, they're eventually separated. The *video signal* that I'll describe here is identical to the signal that's input to or output from those jacks labeled *Video* found on VCRs, camcorders, and some television sets.

For black and white television, this video signal is quite straightforward and easy to comprehend. (Color gets a bit messier.) Sixty times per second, the signal contains a *vertical sync pulse* that indicates the beginning of a field. This pulse is 0 volts (ground) for about 400 microseconds. A *horizontal sync pulse* indicates the beginning of each scan line: The video signal is 0 volts for 5 microseconds 15,750 times per second. Between the horizontal sync pulses, the signal varies from 0.5 volt for black to 2 volts for white, with voltages between 0.5 volt and 2 volts to indicate shades of gray.

The image of a television is thus partially digital and partially analog. The image is divided into 525 lines vertically, but each scan line is a continuous variation of voltages—an analog of the visual intensity of the image. But the voltage can't vary indiscriminately. There's an upper limit to how quickly the television set can respond to the varying signal. This is known as the television's *bandwidth*.

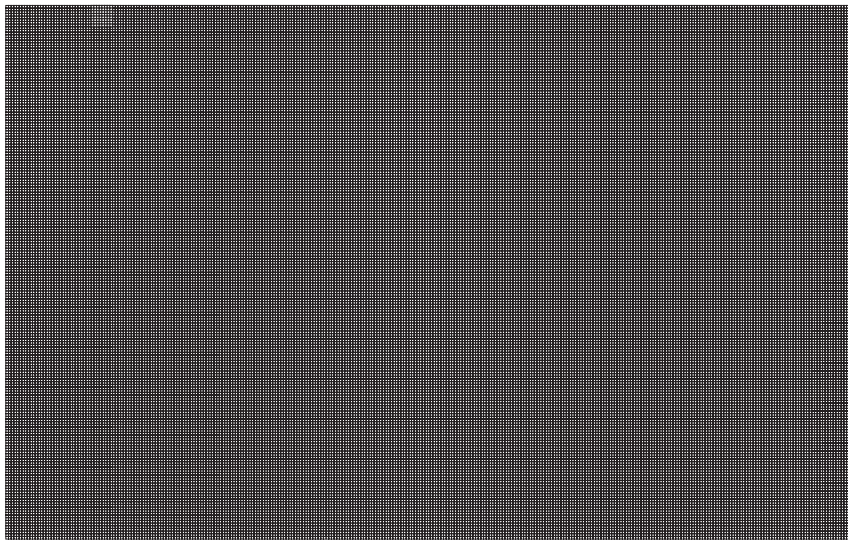
Bandwidth is an extremely important concept in communication, and it relates to the amount of information that can be transferred over a particular communication medium. In the case of television, bandwidth is the limit to the speed with which the video signal can change from black to white and back to black again. For American broadcast television, this is about 4.2 MHz.

If we want to connect a video display to a computer, it's awkward to think of the display as a hybrid analog and digital device. It's easier to treat it as a completely digital device. From the perspective of a computer, it's most convenient to conceive of the video image as being divided into a rectangular grid of discrete dots known as *pixels*. (The term comes from the phrase *picture element*.)

The video bandwidth enforces a limit to the number of pixels that can fit in a horizontal scan line. I defined the bandwidth as the speed with which the video signal can change from black to white and back to black again. A bandwidth of 4.2 MHz for television sets allows two pixels 4.2 million times a second, or—dividing 2 × 4,200,000 by the horizontal scan rate of 15,750—533 pixels in each horizontal scan line. But about a third of these pixels aren't available because they're hidden from view—either at the far ends of the image or while the light beam is in the horizontal retrace. That leaves about 320 useful pixels horizontally.

Likewise, we don't get 525 pixels vertically. Instead, some are lost at the top and bottom of the screen and during the vertical retrace. Also, it's most convenient to *not* rely upon interlace when computers use television sets. A reasonable number of pixels in the vertical dimension is 200.

We can thus say that the *resolution* of a primitive video display adapter attached to a conventional television set is 320 pixels across by 200 pixels down, or 320 pixels horizontally by 200 pixels vertically, commonly referred to as 320 by 200 or 320 × 200:



To determine the total number of pixels in this grid, you can count them or simply multiply 320 by 200 to get 64,000 pixels. Depending on how you've configured your video adapter (as I'll explain shortly), each pixel can be either black or white, or each pixel can be a particular color.

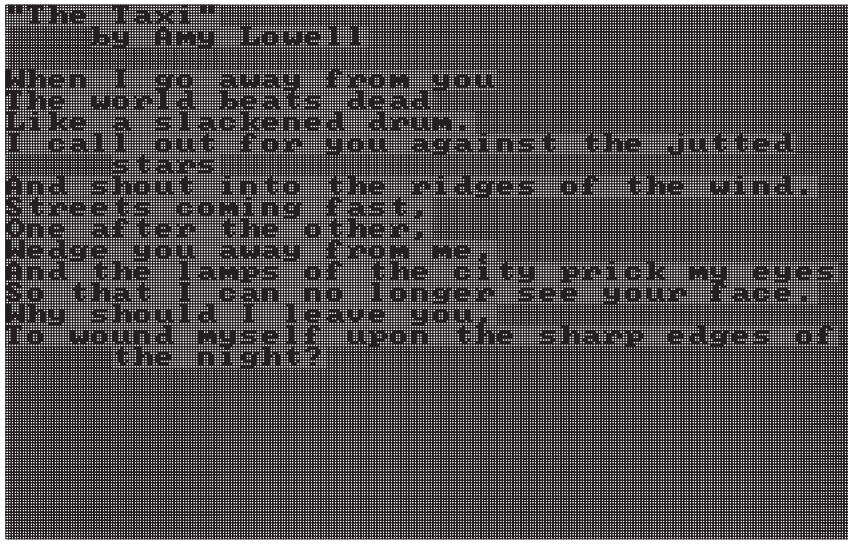
Suppose we wanted to display some text on this display. How much can we fit?

Well, that obviously depends on how many pixels are used for each text character. Here's one possible approach that uses an 8 × 8 grid (64 pixels) for each character:

These are the characters corresponding to ASCII codes 20h through 7Fh. (No visible characters are associated with ASCII codes 00h through 1Fh.)

Each character is identified by a 7-bit ASCII code, but each character is also associated with 64 bits that determine the visual appearance of the character. You can also think of these 64 bits of information as codes.

Using these character definitions, you can fit 25 lines of 40 characters each on the 320 × 200 video display, which (for example) is enough to fit an entire short poem by Amy Lowell:




A video display adapter must contain some RAM to store the contents of the display, and the microprocessor must be able to write data into this RAM to change the display's appearance. Most conveniently, this RAM is part of the microprocessor's normal memory space. How much RAM is required for a display adapter like the one I'm describing?

This isn't a simple question! The possible answers can range from 1 kilobyte to 192 kilobytes!

Let's start with the low estimate. One way to reduce the memory requirements of a video display adapter is to restrict the adapter to text only. We've already established that we can display 25 rows of 40 characters each, or a total of 1000 characters. The RAM on the video board need only store the 7-bit ASCII codes of those 1000 characters. That's 1000 7-bit values, which is approximately 1024 bytes, or 1 kilobyte.

Such a video adapter board must also include a *character generator* that contains the pixel patterns of all the ASCII characters, such as I illustrated earlier. This character generator is generally *read-only memory,* or ROM (pronounced *rahm*). A ROM is an integrated circuit manufactured so that a particular address always results in a particular data output. Unlike RAM, a ROM doesn't have any data input signals.

You can think of ROM as a circuit that converts one code to another. A ROM that stores 8 × 8 pixel patterns of 128 ASCII characters could have 7 address signals (for the ASCII codes) and 64 data output signals. The ROM thus converts a 7-bit ASCII code to a 64-bit code that defines the character's

appearance. But 64 data output signals would make the chip quite large! It's more convenient to have 10 address signals and 8 output signals. Seven of the address signals specify the particular ASCII character. (These 7 address bits come from the data output of the RAM on the video board.) The other 3 address signals indicate the row. For example, address bits 000 indicate the top row and 111 indicate the bottom row. The 8 output bits are the eight pixels of each row.

For example, suppose the ASCII code is 41h. That's a capital A. There are eight rows of 8 bits each. This table shows the 10-bit address (a space separates the ASCII code from the row code) and the data output signals for a capital A:

| Address | Data Output |
|---|---|
| 1000001 000 | 00110000 |
| 1000001 001 | 01111000 |
| 1000001 010 | 11001100 |
| 1000001 011 | 11001100 |
| 1000001 100 | 11111100 |
| 1000001 101 | 11001100 |
| 1000001 110 | 11001100 |
| 1000001 111 | 00000000 |

Do you see the A drawn with 1s against a background of 0s?

A video display adapter that displays text only must also have logic for a *cursor*. The cursor is the little underline that indicates where the next character you type on the keyboard will appear on the display. The character row and column position of the cursor is usually stored in two 8-bit registers on the video board that the microprocessor can write values into.

If the video adapter board is *not* restricted to text only, it's referred to as a *graphics* board. By writing into the RAM on a graphics video board, a microprocessor can draw pictures, including text in a multitude of sizes and styles. Graphics video boards require more memory than text-only boards. A graphics video board that displays 320 pixels across by 200 pixels down has 64,000 pixels. If each pixel corresponds to one bit of RAM, such a board requires 64,000 bits of RAM, or 8000 bytes. This, however, is the rock-bottom minimum. A correspondence of 1 bit to 1 pixel allows the use of only two colors—for instance, black and white. A 0 bit might correspond to a black pixel, and a 1 bit might correspond to a white pixel.

Black-and-white televisions display more than just black and white, of course. They're also capable of displaying many shades of gray. To display shades of gray from a graphics board, it's common for each pixel to correspond to an entire *byte* of RAM, where 00h is black and FFh is white, and all the values in between correspond to shades of gray. A 320-by-200 video board that displays 256 gray shades requires 64,000 *bytes* of RAM. That's very nearly the entire address space of one of the 8-bit microprocessors I've been talking about!

Moving up to full gorgeous color requires 3 bytes per pixel. If you use a magnifying glass to examine a color television or a computer video display,

you'll discover that each color is represented by various combinations of the primary colors red, green, and blue. To get the full range of color, a byte is required to indicate the intensity of each of the three primaries. That means 192,000 bytes of RAM. (I'll have more to say about color graphics in the last chapter of this book.)

The number of different colors that a video adapter is capable of is related to the number of bits used for each pixel. The relationship might look familiar because like many codes in this book, it once again involves a power of 2:

$$\text{Number of Colors} = 2^{\text{Number of bits per pixel}}$$

The 320-by-200 resolution is just about the best you can do on a standard television set. That's why monitors made specifically for computers have a much higher bandwidth than television sets. The first monitors sold with the IBM Personal Computer in 1981 could display 25 lines of 80 characters each. This is the number of characters found on the CRT displays used with IBM's large and expensive mainframe computers. To IBM, 80 characters is a very special number. And why? *Because that's the number of characters on an IBM punch card!* Indeed, in the early days the CRT displays attached to mainframes were often used for viewing the contents of punch cards. Occasionally, you'll hear an old-timer refer to the lines of a text-only video display as *cards*.

Over the years, video display adapters have been characterized by increasing resolution and color capability. An important milestone was reached in 1987 when IBM's Personal System/2 series of personal computers and Apple's Macintosh II both introduced video adapters that did 640 pixels horizontally by 480 pixels vertically. This has remained the minimum-standard video resolution ever since.

The 640-by-480 resolution was a significant milestone, but you might not believe that the reason for its importance goes back to Thomas Edison! Around 1889, when Edison and his engineer William Kennedy Laurie Dickson were working on the Kinetograph motion picture camera and the Kinetoscope projector, they decided to make the motion picture image one-third wider than it was high. The ratio of the width of the image to its height is called the *aspect ratio*. The ratio that Edison and Dickson established is commonly expressed as 1.33 to 1, or 1.33:1, or, to avoid fractions, 4:3. This aspect ratio was used for most movies for over 60 years, and it was also used for television. Only in the early 1950s did the Hollywood studios introduce some *widescreen* techniques that competed against television by going beyond the 4:3 aspect ratio.

The aspect ratio of most computer monitors is (like television) also 4:3, which you can easily prove to yourself using a ruler. The resolution 640 by 480 is also in the ratio 4:3. This means that (for example) a 100-pixel horizontal line is the same physical length as a 100-pixel vertical line. This is considered a desirable feature for computer graphics and is known as *square pixels*.

Today's video adapters and monitors almost always do 640 by 480 but are also capable of various additional video *modes,* often including resolutions of 800 by 600, 1024 by 768, 1280 by 960, and 1600 by 1200.

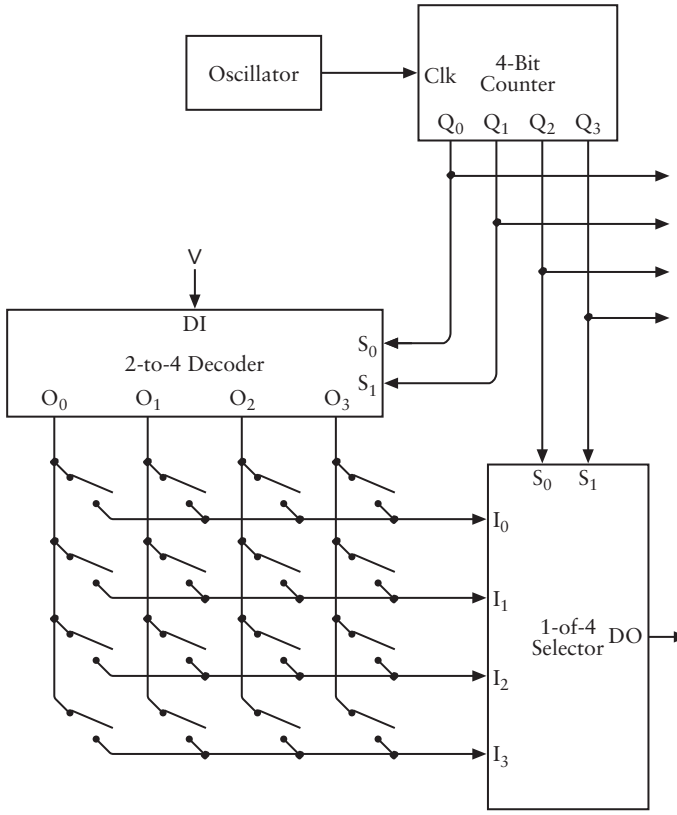
Although we normally think of the computer display and the keyboard as connected in some way—what you type on the keyboard is displayed on the screen—they're usually physically distinct.

Each key on the keyboard is a simple switch. The switch is closed when the key is pressed. A keyboard that resembles a typewriter might have as few as 48 keys; keyboards for today's personal computers often have over 100 keys.

A keyboard attached to a computer must include some hardware that provides a unique code for each key that's pressed. It's tempting to assume that this code is the ASCII code for the key. But it's not practical nor desirable to design hardware that figures out the ASCII code. For example, the A key on the keyboard could correspond to the ASCII code 41h or 61h depending on whether a user also pressed the Shift key. Also, today's computer keyboards have many keys that don't correspond to ASCII characters. The code provided by the keyboard hardware is instead referred to as a *scan code*. A short computer program can figure out what ASCII code (if any) corresponds to a particular key being pressed on the keyboard.

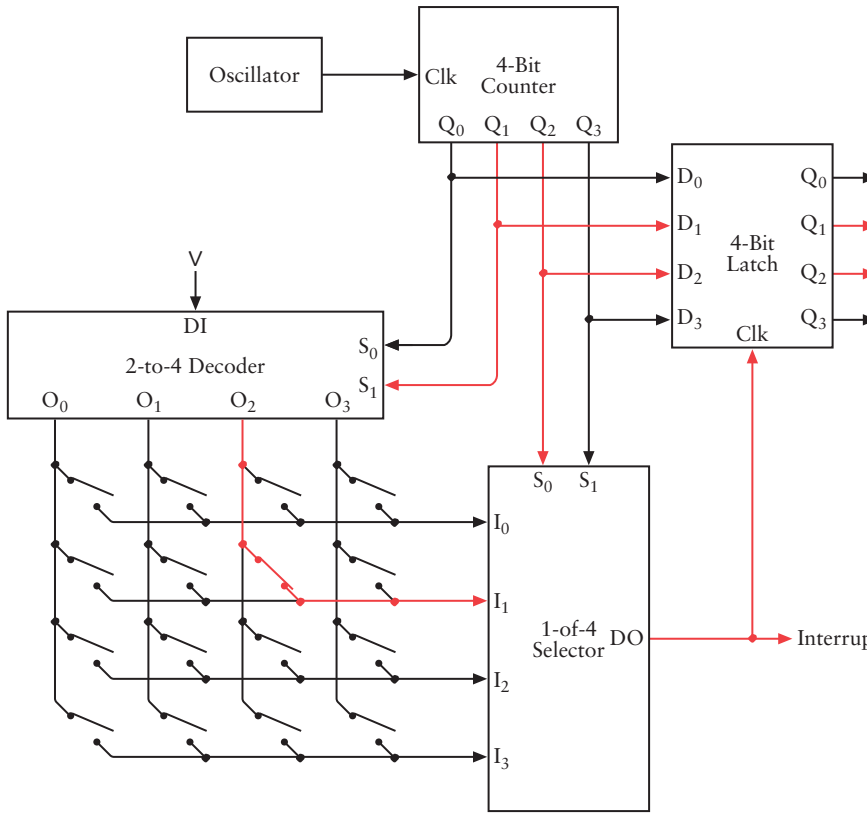
To prevent my diagram of the keyboard hardware from becoming unwieldy, I'm going to assume that our keyboard has a mere 16 keys. Whenever a key is pressed, the keyboard hardware should generate a 4-bit code with binary values ranging from 0000 through 1111.

The keyboard hardware contains components that we've seen before:



The 16 keys of the keyboard are shown as simple switches in the lower left area of this diagram. A 4-bit counter repetitively and very quickly cycles through the 16 codes corresponding to the keys. It must be fast enough to cycle through all the codes faster than a person can press and release a key.

The outputs of the 4-bit counter are the select inputs of both a 2-Line-to-4-Line Decoder and a 4-Line-to-1-Line Data Selector. If no keys are pressed, none of the inputs to the selector can be 1. Therefore the output of the selector isn't 1. But if a particular key is pressed, at a particular 4-bit counter output the output from the selector will be 1. For example, if the switch second from the top and right is pressed, and if the counter output is 0110, the output from the selector becomes 1:




That's the code corresponding to that key. When that key is pressed, no other counter output will cause the output of the selector to be 1. Each key has its own code.

If your keyboard has 64 keys, you need a 6-bit scan code. That would involve a 6-bit counter. You could arrange the keys in an 8×8 array, using a 3-to-8 Decoder and a 1-of-8 Selector. If your keyboard has between 65 and 128 keys, you need a 7-bit code. You could arrange the keys in an 8×16 array and use a 4-to-16 Decoder and an 8-to-1 Selector (or a 3-to-8 Decoder and a 16-to-1 Selector).

What happens next in this circuit depends on the sophistication of the keyboard interface. The keyboard hardware could include 1 bit of RAM for each key. The RAM would be addressed by the counter, and the contents of the RAM could be 0 if the key is up and 1 if the key is down. This RAM could also be read by the microprocessor to determine the status of each key.

One useful part of a keyboard interface is an interrupt signal. As you'll recall, the 8080 microprocessor has an input signal that allows an external device to interrupt what the microprocessor is doing. The microprocessor responds by reading an instruction from memory. This is usually a *RST* instruction and causes the microprocessor to branch to a specific area of memory where a program to handle the interrupt is located.

The final peripheral I'll describe in this chapter is a long-term storage device. As you'll recall, random access memory—whether constructed from relays, tubes, or transistors—loses its contents when the electrical power is shut off. For this reason, a complete computer also needs something for long-term storage. One time-honored approach involves punching holes in paper or cardboard, such as IBM punch cards. In the early days of small computers, rolls of paper tape were punched with holes to save programs and data and to later reload them into memory.

One problem with punch cards and paper tape is that the medium isn't reusable. Once a hole is punched it can't easily be unpunched. Another problem is that it's not particularly efficient. These days, if you can actually *see* a bit, it's probably safe to say that the bit is taking up entirely too much space!

For these reasons, the type of long-term storage that has become much more prevalent is *magnetic storage*. The origins of magnetic storage date back to 1878, when the principles were described by American engineer Oberlin Smith (1840–1926). The first *working* device, however, came 20 years later in 1898 and was built by Danish inventor Valdemar Poulsen (1869–1942). Poulsen's *telegraphone* was originally intended as a device to record telephone messages when the person receiving the call wasn't at home. He employed an electromagnet—that ubiquitous device we've already encountered in the telegraph—to record sound along a moving length of steel wire. The electromagnet magnetizes the wire proportional to the ups and downs of the waveform of the sound. The magnetized wire can then induce a current to the same degree as it's moved along the coils of wire in the electromagnet. The electromagnet used for storing and reading is known as a *head,* regardless of the type of magnetic medium it's used with.

In 1928, Austrian inventor Fritz Pfleumer patented a magnetic recording device based on long lengths of paper tape that had been coated with iron particles using a technology originally designed for creating metallic bands on cigarettes. The paper was soon replaced with a stronger cellulose acetate base, and one of the most enduring and well-known of all recording media was born. Reels of magnetic tape—now conveniently packaged in plastic cassettes—still provide an extremely popular medium for recording and playing back music and video.

The first commercial tape system for recording digital computer data was introduced by Remington Rand in 1950. At the time, a reel of half-inch tape

could store a few megabytes of data. In the early days of home computers, people adapted common cassette tape recorders to save information. Small programs stored the contents of a block of memory to tape and later read it back from tape into memory. The first IBM PCs had a connector for cassette tape storage. Tape remains a popular medium today, particularly for long-term archiving. Tape, however, isn't an ideal medium because moving quickly to an arbitrary spot on the tape isn't possible. It's usually necessary to fast-forward or rewind, and that takes time.

A medium geometrically more conducive to fast access is the disk. The disk itself is spun around its center while one or more heads attached to arms can be moved from the outside of the disk to the inside. Any area on the disk can be accessed very quickly.

For recording sounds, the magnetic disk actually predates the magnetic tape. For storing computer data, however, the first disk drive was invented at IBM in 1956. The Random Access Method of Accounting and Control (RAMAC) contained 50 metal disks 2 feet in diameter and could store 5 megabytes of data.

Since then, disks have become much smaller and of higher capacity. Disks are generally categorized as *floppy disks* (also called *diskettes*) or *hard disks* (also called *fixed disks*). Floppy disks are single sheets of coated plastic inside a protective casing made of cardboard or (more recently) plastic. (A plastic casing prevents the diskette from bending, so the diskette is no longer quite as floppy as the older ones, but it's still referred to as a floppy disk.) Floppy disks must be physically inserted by a person into a floppy disk *drive,* which is the component attached to the computer that writes to and reads from the floppy disk. Early floppy disks were 8 inches in diameter. The first IBM PC used 5¼-inch floppy disks; today the most common format is 3.5 inches in diameter. That floppy disks can be removed from the disk drive allows them to be used for transferring data from one computer to another. Diskettes are also still an important distribution medium of commercial software.

A hard disk usually contains multiple metal disks permanently built into the drive. Hard disks are generally faster than floppy disks and can store more data. But the disks themselves can't be removed.

The surface of a disk is divided into concentric rings called *tracks*. Each track is divided like slices of a pie into *sectors*. Each sector stores a certain number of bytes, usually 512 bytes. The floppy disk drive on the first IBM PC used only one side of the 5¼-inch disk and divided it into 40 tracks with 8 sectors per track and 512 bytes per sector. Each floppy disk thus stored 163,840 bytes, or 160 kilobytes. The 3.5-inch floppy disks used in PC compatibles today have two sides, 80 tracks per side, 18 sectors per track, and 512 bytes per sector for a total of 1,474,560 bytes, or 1440 kilobytes.

The first hard disk drive introduced by IBM for the Personal Computer-XT in 1983 stored ten megabytes. Today, in 1999, a 20-gigabyte hard disk drive (that's 20 *billion* bytes of storage) can be purchased for under $400.

A floppy disk or hard disk usually comes with its own electrical interface and also requires an additional interface between that and the microprocessor. Several standard interfaces are popular for hard drives, including SCSI (Small Computer System Interface, pronounced *scuzzy*), ESDI (Enhanced Small Device Interface, pronounced *ez dee*), and IDE (Integrated Device Electronics). All these interfaces make use of direct memory access (DMA) to take over the bus and transfer data directly between random access memory and the disk, bypassing the microprocessor. These transfers are in increments of the disk sector size, which is usually 512 bytes.

Many newcomers to home computers hear too much technical talk about megabytes of this and gigabytes of that, and they get confused about the difference between semiconductor random access memory and disk storage. In recent years, a rule of sorts has emerged to help alleviate some confusion about terminology. The rule is that the word *memory* is to be used to refer only to semiconductor random access memory, while the word *storage* is to be used for everything else—usually floppy disks, hard disks, and tape. I've tried to follow that rule (even though we've encountered microprocessor machine-code instructions named *Store* that store bytes in RAM).

The most obvious difference between memory and storage is that memory is volatile; it loses its contents when the power is shut off. Storage is nonvolatile; data stays on the floppy disk or hard disk until it's deliberately erased or written over. Yet there's another significant difference that you can appreciate only by understanding what a microprocessor does. When the microprocessor outputs an address signal, it's always addressing memory, not storage.

Getting something from disk storage into memory so that it *can* be accessed by the microprocessor requires extra steps. It requires that the microprocessor run a short program that accesses the disk drive so that the disk drive transfers data from the disk into memory.

The difference between memory and storage can also be understood in a common analogy: Memory is like the top of your desk. Anything that's on your desk you can work with directly. Storage is like a file cabinet. If you need to use something from the file cabinet, you have to get up, walk over to the file cabinet, pull out the file you need, and bring it back to your desk. If your desk gets too crowded, you need to take something from your desk back over to the file cabinet.

This analogy is particularly apt because data stored on disks is actually stored in entities called *files*. Storing files and retrieving them is the province of an extremely important piece of software known as the *operating system*.