

[Figures are not included in this sample chapter]

Upgrading and Repairing PCs, Tenth Anniversary Edition

- 5 -

Memory

This chapter looks at memory from both a physical and logical point of view. We will first look at what memory is, where it fits into the PC architecture, and how it works. We will discuss the different types of memory, speeds, and packaging of the chips and memory modules that you can purchase and install.

The chapter also looks at the logical layout of memory, defining the different areas and uses of these areas from the system's point of view. Because the logical layout and uses are within the "mind" of the processor, memory mapping and logical layout remain as perhaps the most difficult subjects to grasp in the PC universe. This chapter contains useful information that removes the mysteries associated with memory and enables you to get the most out of your system.

Memory Basics

Memory is the workspace for the computer's processor. It is a temporary storage area where the programs and data being operated on by the processor must reside. Memory storage is considered temporary because the data and programs will remain there only as long as the computer has electrical power or is not reset. Before being shut down or reset, any data that has been changed should be saved to a more permanent storage device of some type (usually a hard disk) so it can be reloaded into memory again in the future.

We often call memory RAM, for Random Access Memory. Main memory is called RAM because you can randomly (and quickly) access any location in memory. When we talk about a computer's *memory*, we usually mean the RAM in the system, meaning primarily the memory chips or modules that make up the primary active program and data storage used by the processor. This is often confused with the term "storage," which should be used when referring to things such as disk and tape drives (although some people do consider them a form of memory).

RAM can refer to both the physical chips that make up the memory in the system, and the logical mapping and layout of that memory. Logical mapping and layout refer to how the memory addresses are mapped to actual chips, and what address locations contain what types of system information.

People new to computers often confuse main memory with disk storage, as both have capacities that are expressed in similar megabyte or gigabyte terms. The best analogy to explain the relationship between memory and disk storage I've found is to think of a small office with a desk and a file cabinet.

In this popular analogy, the file cabinet represents the system's hard disk, where both programs and

data are stored for long-term safekeeping. The desktop represents the system's main memory, which allows the person working at the desk (acting as the processor) direct access to any files placed on it. To work on a particular file, it first must be retrieved from the cabinet and placed on the desktop. If the desktop is large enough, you may be able to have several files open on it at one time; likewise, if your system has more memory, you can run more or larger programs.

Adding hard disk space to a system is like putting a bigger file cabinet in the office; more files can be permanently stored. Adding more memory to a system is like getting a bigger desk; you can work on more programs and data at the same time.

One difference between this analogy and the way things really work in a computer is that when a file is loaded into memory, it is a copy of the file that is actually loaded; the original still resides on the hard disk. Note that because of the temporary nature of memory, any files that have been changed after being loaded into memory must then be saved back to the hard disk before the system is powered off and the memory subsequently cleared. If the changed file is not saved, then the original copy of the file on the hard disk will remain unaltered. This is like saying that any changes made to any files left on the desktop will be discarded when the office is closed, although the original files themselves will still be present in the cabinet.

Memory temporarily stores programs when they are running, along with the data being used by those programs. RAM chips are sometimes termed *volatile storage* because when you turn off your computer or an electrical outage occurs, whatever is stored in RAM is lost unless you saved it to your hard drive. Because of the volatile nature of RAM, many computer users make it a habit to save their work frequently. (Some software applications can do timed backups automatically.)

Launching a computer program brings files into RAM, and as long as they are running, computer programs reside in RAM. The CPU executes programmed instructions in RAM, and also stores results in RAM. RAM stores your keystrokes when you use a word processor, and also stores numbers used in calculations. Telling a program to save your data instructs the program to store RAM contents on your hard drive as a file.

Physically, the main memory in a system is a collection of chips or modules containing chips that are normally plugged into the motherboard. These chips or modules vary in both their electrical and physical design and must be compatible with the system into which they are being installed in order to function properly. In this chapter, we will discuss the different types of chips and modules that may be installed in different systems.

Next to the processor, memory can be one of the more expensive components in a modern PC, although the total amount spent on memory for a typical system has declined over the last few years. Even after the price drops, you should still be spending more on the memory for your system than the cost of your motherboard; in fact, up to twice as much. Prior to the memory price crash in mid-'96, memory had maintained a fairly consistent price for many years of about \$40 per megabyte. 16M (a typical configuration back then) cost more than \$600. In fact, memory was so expensive at that time, it was worth more than its weight in gold. These high prices caught the attention of criminals as memory module manufacturers were robbed at gunpoint in several large heists. These robberies were partially induced by the fact that memory was so valuable, the demand was high, and stolen chips or modules were virtually impossible to trace. After the rash of armed robberies and other thefts, memory module manufacturers began posting armed guards and implementing beefed-up security procedures.

By the end of '96, memory prices had cooled considerably to about \$4 a meg, a tenfold price drop in less than a year. Prices continued to fall after the major crash and recently have reached about a dollar and a half per meg, or about \$100 for 64M, a typical configuration today. This means that out of all the money spent on a PC system today, we are now installing about four times more memory in a system compared to a few years ago and, at the same time, are spending about 1/6 of the total amount for it.

Memory may cost us less now than a few years ago, but its useful life has also become much shorter. New types of memory are being adopted more quickly than before, and it is now more likely that any new systems you purchase will not accept the same memory as your existing ones. This means that in an upgrade or repair situation, you will often have to change the memory if you change the motherboard. The chance that you can re-use the memory in an existing motherboard when upgrading to a new one is slim. This is likely to continue in the future, meaning that the system or upgrade motherboard you purchase next year will most likely not be able to use the memory in your current systems.

Because of this, it is important to understand all the different types of memory on the market today, so you can best determine which types are required by which systems, and thus more easily plan for future upgrades and repairs.

Types of Memory

To better understand physical memory in a system, it is necessary to see where and how it fits into the system. Three main types of physical memory used in modern PCs are

- *ROM*. Read Only Memory
- *DRAM*. Dynamic Random Access Memory
- *SRAM*. Static RAM

ROM

Read Only Memory, or ROM, is a type of memory that can permanently or semi- permanently hold data. It is called read-only because it is either impossible or difficult to write to. ROM is also often referred to as non-volatile memory because any data stored in ROM will remain, even if the power is turned off. As such, ROM is an ideal place to put the PC's startup instructions--that is, the software that boots the system.

Note that ROM and RAM are not opposites, as some people seem to believe. In fact, ROM is technically a subset of the system's RAM. In other words, a portion of the system's Random Access Memory address space is mapped into one or more ROM chips. This is necessary to contain the software that enables the PC to boot up; otherwise, the processor would have no program in memory to execute when it was powered on.

See "The Boot Process," p. 1042

For example, when a PC is turned on, the processor automatically jumps to address FFFF0h, expecting to find instructions to tell the processor what to do. This location is exactly 16 bytes from the end of the first megabyte of RAM space, and the end of the ROM. If this location was mapped into regular memory chips, any data stored there would have disappeared when the power was turned off previously, and the processor would subsequently find no instructions to run the next time power was turned on. By placing a ROM chip at this address, a system startup program can be permanently loaded into the ROM and will be available every time the system is turned on.

For more information about Dynamic RAM, see "DRAM," p. 311

Normally, the system ROM will start at address F0000h, which is 64K prior to the end of the first megabyte. Because the ROM chip is normally 64K in size, the ROM programs occupy the entire last 64K of the first megabyte, including the critical FFFF0h startup instruction address.

Some think it strange that the PC would start executing instructions 16 bytes from the end of the ROM, but this design was intentional. All the ROM programmer has to do is place a JMP (jump) instruction at that address, which instructs the processor to jump to the actual beginning of the ROM--in most cases close to F0000h, which is about 64K earlier in the memory map. It's kind of like deciding to read every book starting 16 pages from the end, and then having all book publishers agree to place an instruction there to jump back the necessary number of pages to get to page 1. By setting the startup location in this way, Intel allowed the ROM to grow to be any size, all the while keeping it at the upper end of addresses in the first megabyte of the memory address space.

The main ROM BIOS is contained in a ROM chip on the motherboard, but there are also adapter cards with ROMs on them as well. ROMs on adapter cards contain auxiliary BIOS routines and drivers needed by the particular card, especially for those cards that must be active early in the boot process, such as video cards. Cards that don't need drivers active at boot time will normally not have a ROM because those drivers can be loaded from the hard disk later in the boot process.

The motherboard ROM normally contains four main programs, including the following in most systems:

- *POST*. Power-On Self Test. A series of test routines that ensure the system components are operating properly.
- *CMOS Setup*. A menu-driven application that allows the user to set system configuration parameters, options, security settings, and preferences.
- *Bootstrap Loader*. The routine that first scans the floppy drive and then the hard disk, looking for an operating system to load.
- *BIOS*. Basic Input/Output System. A series of device driver programs designed to present a standard interface to the basic system hardware, especially hardware that must be active during the boot process.

Because the BIOS is the main portion of the code stored in ROM, we often call the ROM the ROM BIOS. In older PCs, the motherboard ROM BIOS could consist of up to five or six total chips, but most PCs have required only a single chip for many years now. For more information on the

motherboard ROM, see Chapter 4, "Motherboards and Buses."

Adapter cards that require startup drivers also have ROMs on them. This includes cards such as video cards, most SCSI (Small Computer Systems Interface) cards, Enhanced IDE controller cards, and some Network cards. The ROM chip on these cards contains drivers and startup programs that will be executed by the motherboard ROM at boot time. This, for example, is how a video card can be recognized and initialized, even though your motherboard ROM does not contain specific drivers for it. You wouldn't want to load the initial VGA mode drivers from disk because the screen would remain dark until those drivers were loaded. What happens is the motherboard ROM scans a special adapter ROM area of RAM (addresses C0000-DFFFFh), looking for a 55AAh signature byte pair that indicates the start of a ROM. The motherboard BIOS automatically runs the programs in any adapter ROMs it finds during the scan. You see this in most systems when you turn your system on, and during the POST you see the video card BIOS initialize and announce its presence.

ROM chips are very slow by nature, with normal access times of 150ns (nanoseconds, or billionths of a second; for more information, see the Memory Speed section later in this chapter), compared to DRAM access times of 60ns or less. Due to this fact, in many systems the ROMs are *shadowed*, which means they are copied into DRAM chips at startup to allow faster access during normal operation. The shadowing procedure copies the ROM into RAM, and then assigns that RAM the same address as the ROM originally used, disabling the actual ROM in the process. This makes the system seem like it has 60ns (or whatever the RAM speed is) ROM. The performance gain from shadowing is often very slight, and it can cause problems if not set up properly, so in most cases it is wise to shadow only the motherboard and maybe video card BIOS, and leave the others alone. Shadowing is mostly useful only if you are running 16-bit operating systems such as DOS or Windows 3.x. If you are running a 32-bit operating system such as Windows 95, Windows 98, or Windows NT, then shadowing is virtually useless because those operating systems do not use the 16-bit ROM code while running. Instead, those operating systems load 32-bit drivers into RAM, which replace the 16-bit BIOS code used only during system startup. Shadowing controls are found in the CMOS Setup program in the motherboard ROM, which is covered in more detail in Chapter 4.

Four different types of ROM chips are

- *ROM*. Read Only Memory
- *PROM*. Programmable ROM
- *EPROM*. Erasable PROM
- *EEPROM*. Electrically Erasable PROM, also called a flash ROM

No matter which type of ROM you use, the data stored in a ROM chip is non-volatile and will remain indefinitely unless intentionally erased or overwritten.

Table 5.1 lists the identifying part numbers typically used for each type of ROM chip along with any other identifying information.

Table 5.1 ROM Chip Part Numbers

ROM Type	Part Number	Other
ROM	No longer in use	
PROM	27nnnn	
EPROM	27nnnn	Quartz window
EEPROM	28xxxx or	29xxxx

Mask ROM

Originally, most ROMs were manufactured with the 0s and 1s already "cast in" or integrated into the die. The die represents the actual silicon chip. These are called mask ROMs, because the data is formed into the mask from which the ROM die is photolithographically produced. This type of manufacturing method is economical if you are making hundreds of thousands of ROMs with the exact same information. If you have to change a single bit, however, you must remake the mask, which is an expensive proposition. Due to costs and inflexibility, nobody uses mask ROMs anymore.

PROM

PROMs are a type of ROM that is blank when new and must be programmed with whatever data you want. The PROM was invented in the late '70s by Texas Instruments, and has been available in sizes from 1K (8K bits) to 2M (16M bits) or more. They can be identified by their part numbers, which are usually 27nnnn, where the 27 indicates the TI type PROM and the nnnn indicates the size of the chip in Kbits. For example, most PCs that used PROMs came with 27512 or 271000 chips, which indicate 512K bits (64K bytes) or 1M bits (128K bytes).

NOTE

My '89 Pontiac Turbo Trans Am came with an onboard computer containing a 2732 PROM, which was a 32Kbit (4K byte) chip in the ECM (Electronic Control Module or vehicle computer) under the dash. This chip contained a portion of the vehicle operating software and all the data tables describing spark advance, fuel delivery, and other engine and vehicle operating parameters. Many devices with integrated computers use PROMs to store their operating programs.

Although we say these chips are blank when new, they are technically preloaded with binary 1s. In other words, a 1Mbit ROM chip used in a modern PC would come with 1 million (actually 1,048,576) bit locations, each containing a binary 1. A blank PROM can then be programmed, which is the act of writing to it. This normally requires a special machine called a Device Programmer, ROM Programmer, or ROM burner (see Figure 5.1). We sometimes refer to programming the ROM as "burning" it because that is technically an apt description of the process. Each binary 1 bit can be thought of as a fuse that is intact. Most chips run on 5 volts, but when we program a PROM, we place a higher voltage (normally 12 volts) at the various addresses within the chip. This higher voltage actually blows or burns the fuses at the locations we desire, thus turning any given 1 into a 0. Although we can turn a 1 into a 0, you should note that the process is irreversible; that is, we cannot turn a 0 back into a 1. The programmer device examines the program you want to write into the chip, and then selectively changes only the 1s to 0s where necessary in the chip. PROM chips are often referred to as OTP (One Time Programmable) chips for this reason. They can be programmed once, and never erased. Most PROMs are very inexpensive, on the order of \$3 for a typical PC motherboard

PROM, so if you wanted to change the program in a PROM, you discard it and program a fresh one with the new data.

FIG. 5.1 *Typical Gang (multi-socket) Device Programmer (PROM Burner).*

The act of programming a PROM takes anywhere from a few seconds to a few minutes, depending on the size of the chip and the algorithm used by the programming device. Figure 5.1 shows a typical PROM programmer that has multiple sockets. This is called a gang-programmer and can program several chips at once, saving time if you have several chips to write with the same data. Less expensive programmers are available with only one socket, which is fine for most individual use.

I use and recommend a very inexpensive programmer from a company called Andromeda Research (see Appendix A, "Vendor List"). Besides being economical, their unit has the advantage of connecting to a PC via the parallel port for fast and easy data transfer of files between the PC and the programming unit. Their unit is also portable and comes built in to a convenient carrying case. It is operated by an included menu-driven program that you install on the connected PC. The program contains several features, including a function that allows you to read the data from a chip and save it in a file on your system; you can also write a chip from a data file, verify a chip matches a file, and verify a chip is blank before programming begins.

Chapter 4 shows how I used my PROM programmer to modify the BIOS in some of my older PCs. With today's systems primarily using flash ROMs, this capability is somewhat limited, but I include the information because I think it is very interesting.

NOTE

I even used my PROM programmer to reprogram the PROM in my '89 Turbo Trans Am, changing the factory preset speed and rpm limiters, turbocharger boost, torque converter lockup points, spark advance, fuel delivery, idle speed, and much more! I also incorporated a switch box under the dash that allows me to switch among four different chips even while the vehicle is running. One chip I created I call the "valet chip," which when engaged will cut off the fuel injectors at a preset speed of 36 miles per hour, and restart them when the vehicle coasts down to 35 mph. I imagine this type of modification would be useful for those with teenagers driving, as you could set the mph or engine rpm limit to whatever you want! Another chip I created cuts off all fuel to the engine altogether, which I engage when the vehicle is parked, for security purposes. No matter how clever, a thief will not be able to steal this car unless they tow it away. If you are interested in such a chip-switching device or custom chips for your Turbo Trans Am or Buick Grand National, contact Casper's Electronics (see the vendor list in Appendix A). For other vehicles with replaceable PROMs, companies such as Superchips, Hypertech, and Evergreen offer custom PROMs for improved performance (see Appendix A). I installed a Superchips chip in a Ford Explorer I had and it made a dramatic improvement in engine operation and vehicle performance.

EPROM

One variation of the PROM that has been very popular is the EPROM. An EPROM is a PROM that is erasable. An EPROM chip can be easily recognized by the clear quartz crystal window set in the chip package directly over the die. You can actually see the die through the window! EPROMs have the same 27xxxx part numbering scheme as the standard PROM, and are functionally and physically

identical except for the clear quartz window above the die.

The purpose of the window is to allow ultraviolet light to reach the chip die, because the EPROM is erased by exposure to intense UV light. The window is quartz crystal because regular glass will block UV light. You can't get a suntan through a window! The quartz window makes the EPROMS more expensive than the OTP PROMs. This extra expense is needless if erasability is not important.

The UV light erases the chip by causing a chemical reaction that essentially melts the fuses back together! Thus, any binary 0s in the chip become 1s, and the chip is restored to new condition with binary 1s in all locations. To work, the UV exposure must be at a specific wavelength (2,537 angstroms), at a fairly high intensity (12,000 uw/cm²), in close proximity (2-3cm or about 1 inch), and last for between 5 and 15 minutes duration. An EPROM eraser (see Figure 5.2) is a device that contains a UV light source (usually a sunlamp-type bulb) above a sealed compartment drawer where you place the chip or chips.

FIG. 5.2 *A professional EPROM eraser.*

Figure 5.2 shows a professional-type EPROM eraser that can handle up to 50 chips at a time. I use a much smaller and less expensive one called the DataRase by Walling Co. (see the vendor list). This device erases up to four chips at a time and is both economical and portable.

The quartz crystal window on an EPROM is normally covered by tape, which prevents accidental exposure to UV light. There is UV light in sunlight, of course, and even in standard room lighting, such that over time a chip exposed to the light may begin to degrade. For this reason, after a chip is programmed, it is a good idea to put a sticker over the window to protect it.

EEPROM/Flash ROM

A newer type of ROM is the EEPROM, which stands for Electrically Erasable PROM. These chips are also called flash ROMs, and are characterized by their capability to be erased and reprogrammed directly in the circuit board in which they are installed, with no special equipment required. By using an EEPROM or flash ROM, it is possible to erase and reprogram the motherboard ROM in a PC without removing the chip from the system, or even opening up the system chassis! With a flash ROM or EEPROM, you don't need a UV eraser or device programmer to program or erase chips. Not only do virtually all PC motherboards built since 1994 use flash ROMs or EEPROMs, most automobiles built since then also use them as well. For example, my '94 Chevy Impala SS has a PCM (Powertrain Control Module) with an integral flash ROM.

The EEPROM or flash ROM can be identified by a 28xxxx or 29xxxx part number, and the lack of a window on the chip. Having an EEPROM or flash ROM in your PC motherboard means you can now easily upgrade the motherboard ROM without having to swap chips. In most cases, you will download the updated ROM from the motherboard manufacturer's Web site, and then run a special program they provide to update the ROM. This procedure is described in more detail in Chapter 4.

See "Upgrading the ROM BIOS," p. 215

It is recommended that you periodically check with your motherboard manufacturer to see whether an updated BIOS is available for your system. An updated BIOS may contain bug fixes or enable new

features not originally found in your system. For example, you might find fixes for a potential Year 2000 date problem or new drivers to support booting from LS-120 (120M floppy) drives.

See "LS-120 (120M) Floptical Drives," p. 798

NOTE

For the auto enthusiasts out there, you might want to do the same for your car; that is, check to see if ROM upgrades are available for your vehicle's computer. Now that updates are so easy and inexpensive, vehicle manufacturers are releasing bug-fix ROM upgrades for cars that correct operational problems or improve vehicle performance. In most cases, you will have to check with your dealer to see whether any new vehicle ROMs are available. If you have a GM car, GM has a site on the Web where you can get information about the BIOS revisions available for your car, which they call Vehicle Calibrations. The GM Vehicle Calibration Information site address is as follows:

<http://207.74.147.14/vci/>

When you enter your VIN (Vehicle Identification Number), this page displays the calibration history for the vehicle, which is a list of all the different flash ROM upgrades (calibrations) developed since the vehicle was new. For example, when I entered the VIN on my '94 Impala, I found that there have been five different flash ROM calibrations over the years, and my car had only the second revision installed originally, meaning there had been three newer ROMs than the one I had! The fixes in the various calibration updates are listed also. A trip to the dealer with this information allowed them to use their diagnostic computer to connect to my car and reflash the PCM with the latest software, which, in my case, fixed several problems including engine surging under specific conditions, shift clunks, erroneous "check engine" light warnings, and several other minor problems.

With the flash ROM capability, I began experimenting with running calibrations originally intended for other vehicles and now, in fact, run a modified Camaro calibration loaded into the flash ROM in my Impala. The spark advance curve and fuel delivery parameters are much more aggressive in the Camaro calibration, as are the transmission shift points and other features. If you are interested in having a custom program installed in your flash ROM-equipped vehicle, I recommend you contact Wright Automotive or Evergreen Performance (see the vendor list).

These days, many objects with embedded computers controlling them are using flash ROMs. Pretty soon you'll be taking your toaster in for flash ROM upgrades! The method for locating and updating your PC motherboard flash ROMs is shown in Chapter 4. Other devices may have flash ROMs, as well--for example, I have updated the flash ROMs in my Motorola ISDN modem and in my Kodak digital camera. Both of these items had minor quirks that were fixed by updating their internal ROM code, which was as easy as downloading a file from their respective Web sites and running the update program included in the file. Flash ROMs are also frequently used to add new capabilities to peripherals or update peripherals such as modems to the latest standards, such as updating a modem from X2 or Kflex to v.90.

DRAM

Dynamic RAM is the type of memory chip used for most of the main memory in a modern PC. The main advantages of DRAM is that it is very dense, meaning you can pack a lot of bits into a very small chip, and it is very inexpensive, which makes it affordable for large amounts of memory.

The memory cells in a DRAM chip are tiny capacitors that retain a charge to indicate a bit. The problem with DRAM is that it is dynamic, and because of the design must be constantly *refreshed* or the electrical charges in the individual memory capacitors will drain and the data will be lost. Refresh occurs when the system memory controller takes a tiny break and accesses all the rows of data in the memory chips. Most systems have a memory controller (normally built in to the motherboard chipset), which is set for an industry-standard refresh rate of 15 μ s (microseconds). The memory is accessed in such a way that all rows of data will be accessed after exactly 128 of these special refresh cycles. This means that every 1.92ms (milliseconds or 128x15 μ sec), all the rows in the memory are read to refresh the data.

See "Chipsets," p. 183

Refreshing the memory unfortunately takes processor time away from other tasks because each refresh cycle takes several CPU cycles to complete. In older systems, the refresh cycling could take up to 10% or more of the total CPU time, but with modern systems running in the hundreds of megahertz, refresh overhead is now on the order of 1% or less of the total CPU time. Some systems allow you to alter the refresh timing parameters via the CMOS Setup, but be aware that increasing the time between refresh cycles to speed up your system can allow some of the memory cells to begin draining, which can cause random soft memory errors to appear. A soft error is a data error that is not caused by a defective chip. It is safer to stick with the recommended or default refresh timing in most cases. Because refresh consumes less than 1% of modern system overall bandwidth, altering the refresh rate has little effect on performance.

DRAMs use only one transistor and capacitor pair per bit, which makes them very dense, offering a lot of memory capacity per chip than other types of memory. There are currently DRAM chips available with densities of up to 256Mbits or more. This means that there are DRAM chips with 256 million or more transistors! Compare this to a Pentium II, which has 7.5 million transistors, and it makes the processor look wimpy by comparison. The difference is that in a memory chip, the transistors and capacitors are all consistently arranged in a (normally square) grid of very simple repetitive structures, unlike the processor, which is a much more complex circuit of different structures and elements interconnected in a highly irregular fashion. At least one manufacturer is working on chips, intended for the 2001-2002 timeframe, with 256-gigabit densities using circuit line widths of 0.05 micron, indicating that memory densities will continue to rise as they have been for years.

The transistor for each DRAM bit cell is used to read the charge state of the adjacent capacitor. If the capacitor is charged, the cell is read to contain a 1; no charge indicates a 0. The charge in the tiny capacitors is constantly draining, which is why the memory must be refreshed constantly. Even a momentary power interruption, or anything that interferes with the refresh cycles, will cause a DRAM memory cell to lose the charge and therefore the data.

DRAM is used in PC systems because it is inexpensive and the chips can be densely packed, meaning that a lot of memory capacity can fit in a small space. Unfortunately, DRAM is also slow, normally much slower than the processor. For this reason, there have been many different types of DRAM

architectures developed to improve performance.

Memory Speeds

The speed and performance issue with memory is confusing to some because memory speed is usually expressed in ns (nanoseconds), while processor speed is expressed in MHz (megahertz).

A nanosecond is defined as one billionth of a second--a very short time indeed. To put some perspective on that, note that the speed of light is 186,282 miles (299,792 kilometers) per second in a vacuum. In one billionth of a second, a beam of light would travel a mere 11.80 inches or 29.98 centimeters, less than the length of a typical ruler!

Chip and system speed has been expressed in megahertz (MHz), which is millions of cycles per second. There are systems today with processors running 500MHz or faster, and we should see gigahertz (GHz or billions of cycles per second) speeds within a few years.

Because it is confusing to speak in these different terms for speeds, I thought it would be interesting to see how they compare. Table 5.2 shows the relationship between nanoseconds (ns) and megahertz (MHz) for the speeds associated with PCs today.

Table 5.2 The Relationship Between Clock Speeds in Megahertz (MHz) to Cycle Times in Nanoseconds (ns)

Clock Speed	Cycle Time	Clock Speed	Cycle Time
6MHz	166ns	120MHz	8.3ns
8MHz	125ns	133MHz	7.5ns
10MHz	100ns	150MHz	6.6ns
12MHz	83ns	166MHz	6.0ns
16MHz	62ns	180MHz	5.5ns
20MHz	50ns	200MHz	5.0ns
25MHz	40ns	233MHz	4.2ns
33MHz	30ns	250MHz	4.0ns
40MHz	25ns	300MHz	3.3ns
50MHz	20ns	333MHz	3.0ns
60MHz	16ns	350MHz	2.8ns
66MHz	15ns	400MHz	2.5ns
75MHz	13ns	450MHz	2.2ns
80MHz	12ns	500MHz	2.0ns
100MHz	10ns	550MHz	1.8ns

As you can see from this table, as clock speeds increase, cycle time decreases. If you examine this table, you can clearly see that the DRAM memory used in the typical PC for many years is totally inadequate when compared to processor speeds of 400MHz and higher. Note that until recently, most DRAM memory used in PCs has been rated at an access time of 60ns, which works out to be only about 16.7MHz! Consider that this super slow memory has been installed in systems running up to

300MHz or faster; you can see what a mismatch there is between processor and main memory performance.

System memory timing is a little more involved than converting nanoseconds to megahertz. Because the transistors for each bit in a memory chip are most efficiently arranged in a grid, each transistor is accessed by using a row and column scheme. All memory accesses involve first selecting a row address, then a column address, and then transferring the data. The initial setup for a memory transfer where the row and column addresses are selected is a necessary overhead normally referred to as *latency*. The access time for memory is the cycle time plus latency for selecting the row and column addresses. For example, memory rated at 60ns normally has a latency of about 25ns (to select the row and column address) and a cycle time of about 35ns to actually transfer the data. Thus, the true memory clock rate in a system with 60ns memory would be on the order of 28.5MHz (35ns = 28.5MHz). Even so, a single memory transfer will still require a full 60ns, so consecutive transfers will happen at a rate of only 16.7MHz (60ns) because of the added latency.

What happens when a 300MHz processor is trying to read multiple bytes of data from 16MHz memory? The answer is a lot of wait states! A *wait state* is an additional "do-nothing" cycle that the processor must execute while waiting for the data to become ready. In the case with memory cycling every 60ns (16MHz) and a processor cycling every 3ns (300MHz), the processor would have to execute approximately 19 wait states before the data would be ready on the 20th cycle. Adding wait states in this fashion effectively slows the processing speed to that of the memory, or 16MHz in this example. To reduce the number of wait states required, there are several types of faster memory and cache available, all of which are discussed in this chapter.

Fast Page Mode (FPM) DRAM

Standard DRAM is accessed via a technique called *paging*. Normal memory access requires that a row and column address be selected, which takes time. Paging allows for faster access to all the data within a given row of memory by keeping the row address the same and changing only the column. Memory that uses this technique is called *Page Mode* or *Fast Page Mode* memory. Other variations on Page Mode were called Static Column or Nibble Mode memory.

Paged memory is a simple scheme for improving memory performance that divides memory into pages ranging from 512 bytes to a few kilobytes long. The paging circuitry then enables memory locations within a page to be accessed with fewer wait states. If the desired memory location is outside the current page, one or more wait states are added while the system selects the new page.

To improve further on memory access speeds, systems have evolved to allow for faster access to DRAM. One of the more significant changes was the implementation of burst mode access in the 486 and later processors. Burst mode cycling takes advantage of the fact that most memory accesses are consecutive in nature. After setting up the row and column addresses for a given access, using burst mode, one can then access the next three adjacent addresses with no additional latency or wait states. A burst access is normally limited to four total accesses. To describe this, we often refer to the timing in the number of cycles for each access. A typical burst mode access of standard DRAM would be expressed as x-y-y-y, where x is the time for the first access (latency plus cycle time) and y represents the number of cycles required for each consecutive access.

Standard 60ns DRAM normally runs 5-3-3-3 burst mode timing. This means the first access takes a total of five cycles (on a 66MHz system bus, this would be about 75ns total or 5x15ns cycles), while

the consecutive cycles take three cycles each ($3 \times 15\text{ns} = 45\text{ns}$). As you can see, the actual system timing is somewhat less than the memory is technically rated for. Note that without the bursting technique, memory access would be 5-5-5-5 because the full latency would be needed for each memory transfer.

DRAM memory that supports paging and this bursting technique is called *Fast Page Mode (FPM) memory*. The term comes from the fact that memory accesses to data on the same page can be done with less latency. Most 486 and newer systems use FPM memory, while older systems used conventional DRAM.

Another technique for speeding up FPM memory was a technique called *interleaving*. This is a design where two separate banks of memory are used together, alternating access from one to the other as even and odd bytes. While one is being accessed, the other is being precharged, where the row and column addresses are being selected. Then, by the time the first bank in the pair is finished returning data, the second bank in the pair is finished with the latency part of the cycle and is now ready to return data. While the second bank is returning data, the first bank is being precharged, selecting the row and column address of the next access. This overlapping of accesses in two banks reduces the effect of the latency or precharge cycles and allows for faster overall data retrieval. The only problem is that to utilize interleaving, you must install identical pairs of banks together, doubling the amount of SIMMs or DIMMs required. This was popular on 32-bit wide memory systems on 486 processors, but fell out of favor on Pentiums due to the 64-bit wide memory width on those systems. To do interleaving on a Pentium machine, you would need to install memory 128 bits at a time, meaning four 72-pin SIMMs or two DIMMs at a time.

EDO RAM

Starting in 1995, a new type of memory became available for Pentium systems called *EDO (Extended Data Out) RAM*. EDO is a modified form of FPM memory and is also referred to as *Hyper Page Mode*. EDO was invented and patented by Micron Technology, although they have licensed production to numerous other memory manufacturers. EDO memory consists of specially manufactured chips that allow for a timing overlap between successive accesses. The name Extended Data Out refers specifically to the fact that unlike FPM, the data output drivers on the chip are not turned off when the memory controller removes the column address to begin the next cycle. This allows the next cycle to overlap the previous one, saving approximately 10ns per cycle.

The effect of EDO is that cycle times are improved by allowing the memory controller to begin a new column address instruction while it is reading data at the current address. This is almost identical to what was achieved in older systems by interleaving banks of memory, but unlike interleaving, you don't need to install two identical banks of memory in the system at a time.

EDO RAM allows for burst mode cycling of 5-2-2-2, as compared to the 5-3-3-3 of standard Fast Page Mode memory. This means that to do four memory transfers, EDO would require 11 total system cycles as compared to 14 total cycles for FPM. This is a 22% improvement in overall cycling time, but in actual testing EDO normally increases overall system benchmark speed by about 5%. Even though the overall system improvement may seem small, the important thing about EDO is that it uses the same basic DRAM chip design as FPM, meaning that there is virtually no additional cost over FPM. As such, EDO costs about the same as FPM and offers higher performance.

To actually use EDO memory, your motherboard chipset must support it. Most mother-board chipsets

since the Intel 430FX (Triton), which debuted in 1995, have offered support for EDO. Because EDO memory chips cost the same to manufacture as standard chips, combined with the fact that Intel began to support EDO in all their chipsets, the PC market jumped on the EDO bandwagon full force.

See "Fifth-Generation (P5 Pentium Class) Chipsets," p. 187, and "Sixth-Generation (P6 Pentium Pro / Pentium II Class) Chipsets," p. 199

EDO RAM is ideal for systems with bus speeds of up to 66MHz, which fit perfectly with the PC market up through 1997; however, for 1998 and beyond, the market for EDO will rapidly decline as the newer and faster SDRAM (Synchronous DRAM) architecture becomes the standard for new PC system memory.

Burst EDO

A variation of EDO is *Burst Extended-Data-Out Dynamic Random Access Memory (BEDO DRAM)*. BEDO is basically EDO memory with special burst features for even speedier data transfers than standard EDO. Unfortunately, only one chipset (Intel 440FX Natoma) supported it, and it was quickly overshadowed by SDRAM, which seemed to be favored among PC system chipset and system designers. As such, you won't see BEDO being used in systems today, and it is no longer in production.

SDRAM

SDRAM stands for Synchronous DRAM, a type of DRAM that runs in synchronization with the memory bus. SDRAM delivers information in very high-speed bursts using a high-speed, clocked interface. SDRAM removes most of the latency involved in asynchronous DRAM because the signals are already in synchronization with the motherboard clock.

Like EDO RAM, your chipset must support this type of memory for it to be usable in your system. Starting in 1997 with the 430VX and 430TX, all of Intel's subsequent chipsets fully support SDRAM, making it the most popular type of memory for new systems. SDRAM is especially suited to the Pentium II architecture, and the new high-performance motherboards that run it.

Performance of SDRAM is dramatically improved over FPM or EDO RAM. Because SDRAM is still a type of DRAM, the initial latency is the same, but overall cycle times are much faster than with FPM or EDO. SDRAM timing for a burst access would be 5-1-1-1, meaning that four memory reads would complete in only eight system bus cycles, as compared to 11 cycles for EDO and 14 cycles for FPM.

Besides the fact that SDRAM can work in fewer cycles, it is also capable of supporting 100MHz (10ns) or faster system bus cycling, which has become the new standard for system speed starting in 1998. As such, virtually all new PC systems sold in 1998 and for probably two years thereafter will include SDRAM memory.

It is anticipated that in the near future, this figure will be pushed to 200MHz in order to keep up with faster systems of the future. SDRAM is sold in DIMM form, and is often rated by megahertz speed rather than nanosecond cycling time. As such, you will see SDRAM sold as 66MHz or 15ns, 83MHz

or 12ns, or 100MHz or 10ns. Due to the extreme timing at the 100MHz level, Intel has created a PC/100 standard that defines the performance criteria a 100MHz memory module must meet to be reliable in a 100MHz system. To meet the criteria with enough overhead to spare, most PC/100-certified memory will be 8ns, or 125MHz. Even though they are technically capable of 125MHz, they will be PC/100 or 100MHz "certified." Although 10ns memory might work, Intel feels that it will not be reliable enough because the margins are too close.

Although SDRAM is significantly faster than previous types of memory, prices are not appreciably higher, which has made the acceptance of SDRAM even more rapid.

See "SIMMs and DIMMs," p. 324

Future DRAM Memory Technologies

RDRAM

The RDRAM, or Rambus DRAM, is a radical new memory design that will be used in high-end PC systems starting in the 1999-2000 timeframe. It is endorsed by Intel and will be directly supported in the future chipsets released during that timeframe.

RDRAMs boost raw memory bandwidth by doubling the on-chip data bus to 16 bits and increasing the clock to 800MHz, for a peak bandwidth of 1.6 Gbytes/second. Rambus made improvements to the bus protocol so that it no longer multiplexes or shares control information on the data bus. Instead, it creates an independent control and address bus split into two groups of pins for row and column commands, while data is transferred across the 2-byte-wide data bus. Synchronization is achieved by sending packets on the falling edge of the clock.

The architecture supports multiple, simultaneous interleaved transactions. Internally, the 64-Mbit device uses a total 128-bit-wide data path operating at 100MHz, allowing 16-byte transfers to or from the core every 10ns. Because RDRAMs have four power-down modes and automatically transition into standby at the end of a transaction, total module power is slightly lower than an SDRAM's at the desktop and similar to EDO for mobile systems.

RDRAM chips will be installed in modules called RIMMs (Rambus Inline Memory Modules). A RIMM (shown in Figure 5.3) is similar in size and form to current DIMMs, but they are not interchangeable.

FIG. 5.3 *168-pin RIMM module.*

An RDRAM memory controller with a single Rambus channel supports up to three RIMM modules, which at the 64Mbit chip density level supports 256M per RIMM. Future RIMM versions will be available in higher capacities, up to 1G or more, and it will be possible to develop chipsets (with integral Rambus memory controllers), which can support more Rambus channels, allowing for more RIMM sockets per board.

Interestingly Rambus does not manufacture either the RDRAM chips or the RIMMs; that is left to other companies. Rambus is merely a design company, and they have no chip fabs or manufacturing facilities of their own. They license their technology to other companies who then manufacture the

devices and modules. There are at least 13 RDRAM licensees, such as Fujitsu Ltd., Hitachi Ltd., Hyundai Electronics Industry Co. Ltd., IBM Microelectronics, LG Semiconductor Co. Ltd., Micron Technology Inc., Mitsubishi Electric Corp., NEC Corp., Oki Electric Industry Co. Ltd., Samsung Electronics Corp., Siemens AG, and Toshiba Corp. All these companies will be manufacturing both the RDRAM devices and the RIMMs that contain them.

DDR SDRAM

Double Data Rate (DDR) SDRAM memory is an evolutionary design of standard SDRAM where data is transferred twice as fast. Instead of doubling the actual clock rate, DDR memory achieves the doubling in performance by transferring two times per transfer cycle, once at the leading and once at the trailing edge of the cycle. This effectively doubles the transfer rate, even though the same overall clock and timing signals are used.

DDR is being proposed by processor companies such as AMD and Cyrix, and chipset manufacturers such as VIA Technologies, ALi (Acer Labs, Inc.), and SiS (Silicon integrated Systems) as a low-cost license-free alternative to RDRAMs. Intel is officially supporting only RDRAMs for new high-end systems in 1999 and beyond, while DDR seems destined for the lower-end commodity PCs as an alternative. Official standardization of DDR was undertaken by the DDR Consortium, an industry panel consisting of Fujitsu, Ltd., Hitachi, Ltd., Hyundai Electronics Industries Co., Mitsubishi Electric Corp., NEC Corp., Samsung Electronics Co., Texas Instruments, Inc., and Toshiba Corp.

Expect DDR SDRAM to make an appearance during 1999, primarily in non-Intel processor-based systems.

Cache Memory--SRAM

There is another distinctly different type of memory that is significantly faster than most types of DRAM. SRAM stands for Static RAM, which is so named because it does not need the periodic refresh rates like DRAM (Dynamic RAM). Due to the design of SRAM, not only are refresh rates unnecessary, but SRAM is much faster than DRAM and is fully able to keep pace with modern processors.

SRAM memory is available in access times of 2ns or less, which means it can keep pace with processors running 500MHz or faster! This is due to the SRAM design, which calls for a cluster of six transistors for each bit of storage. The use of transistors but no capacitors means that refresh rates are not necessary because there are no capacitors to lose their charges over time. As long as there is power, SRAM will remember what is stored. With these attributes, then why don't we use SRAM for all system memory? The answers are simple:

Type	Speed	Density	Cost
DRAM	Slow	High	Low
SRAM	Fast	Low	High

Compared to DRAM, SRAM is much faster, but also much lower in density and much more expensive. The lower density means that SRAM chips are physically larger and store many less bits overall. The high number of transistors and the clustered design means that SRAM chips are both physically larger and much more expensive to produce than DRAM chips. For example, a DRAM

module might contain 64M of RAM or more, while SRAM modules of the same approximate physical size would have room for only 2M or so of data and would cost the same as the 64M DRAM module. Basically, SRAM is up to 30 times larger physically and up to 30 times more expensive than DRAM. The high cost and physical constraints have prevented SRAM from being used as the main memory for PC systems.

Even though SRAM is too expensive for PC use as main memory, PC designers have found a way to use SRAM to dramatically improve PC performance. Rather than spend the money for all RAM to be SRAM memory, which can run fast enough to match the CPU, it is much more cost-effective to design in a small amount of high-speed SRAM memory, called *cache memory*. The cache runs at speeds close to or even equal to the processor, and is the memory from which the processor normally directly reads from and writes to. During read operations, the data in the high-speed cache memory is resupplied from the lower-speed main memory or DRAM in advance.

Up until recently, DRAM was limited to about 60ns (16MHz) in speed. When PC systems were running 16MHz and less, it was possible for the DRAM to fully keep pace with the motherboard and system processor, and there was no need for cache. However, as soon as processors crossed the 16MHz barrier, it was no longer possible for DRAM to keep pace, and that is exactly when SRAM began to enter PC system designs. This occurred back in 1986-1987 with the debut of systems with the 386 processor running at 16- and 20MHz. These were among the first PC systems to employ what we call *cache memory*, a high-speed buffer made up of SRAM that directly feeds the processor. Because the cache can run at the speed of the processor, the system is designed so the cache controller will anticipate the processor's memory needs, and preload the high-speed cache memory with that data. Then, as the processor calls for a memory address, the data can be retrieved from the high-speed cache rather than the much lower speed main memory.

Cache effectiveness is expressed as a *hit ratio*. This is the ratio of cache hits to total memory accesses. A *hit* is when the data the processor needs has been preloaded into the cache from the main memory, meaning that the processor can read it from the cache. A *cache miss* is when the cache controller did not anticipate the need for a specific address; the desired data was not preloaded into the cache; therefore, the processor must retrieve the data from the slower main memory, instead of the faster cache. Anytime the processor reads data from main memory, the processor will have to wait because the main memory cycles at a much slower rate than the processor. If the processor is running at 233MHz, then it is cycling at nearly 4ns, while the main memory might be 60ns, which means it is running at only 16MHz. Thus, every time the processor reads from main memory, it would effectively slow down to 16MHz! The slowdown is accomplished by having the processor execute what are called *wait states*, which are cycles where nothing is done; the processor essentially cools its heels while waiting for the slower main memory to return the desired data. Obviously, we don't want our processors slowing down, so cache function and design become very important as system speeds increase.

To minimize the situation where the processor is forced to read data from the slow main memory, there are normally two stages of cache in a modern system, called Level 1 (L1) and Level 2 (L2). The Level 1 cache is also called integral or internal cache because it is directly built in to the processor, and is actually a part of the processor die (raw chip). Because of this, L1 cache always runs at the full speed of the processor core, and is the fastest cache in any system. All 486 and higher processors incorporate integral L1 cache, making them significantly faster than their predecessors. Level 2 cache is also called external cache because it is external to the processor chip. Originally, this meant that it was installed on the motherboard, as was the case with all 386, 486, and Pentium systems. In those

systems, the L2 cache runs at motherboard speed because it is installed on the motherboard. You can normally find the L2 cache directly adjacent to the processor socket in Pentium and earlier systems.

In the interest of improved performance, later processor designs from Intel, including the Pentium Pro and Pentium II, have included the L2 cache as a part of the processor. It is still external to the actual CPU die, and is instead a separate die or chip mounted inside the processor package or module. As such, Pentium Pro or Pentium II systems will not have any cache on the motherboard; it is all contained within the processor module, instead.

The key to understanding both cache and main memory is to see where they fit in the overall system architecture.

Figure 5.4 shows a typical Pentium MMX system with the Intel 430TX chipset.

FIG. 5.4 *System Architecture, Pentium MMX system with an Intel 430TX chipset.*

Table 5.3 illustrates the need for and function of Level 1 (internal) and Level 2 (external) cache in modern systems.

Table 5.3 The Relationship Between Level 1 (Internal) and Level 2 (External) Cache in Modern Systems

CPU Type	486 DX4	Pentium	Pentium Pro	Pentium II (1997)	Pentium II (1998)
Typical CPU Speed:	100MHz	233MHz	200MHz	300MHz	400MHz
L1 Cache Speed:	10ns (100MHz)	4ns (233MHz)	5ns (200MHz)	3ns (300MHz)	2ns (400MHz)
L2 Cache Speed:	30ns (33MHz)	15ns (66MHz)	5ns (200MHz)	6ns (150MHz)	5ns (200MHz)
Motherboard Speed:	33MHz	66MHz	66MHz	66MHz	100MHz
SIMM/DIMM Speed:	60ns (16MHz)	60ns (16MHz)	60ns (16MHz)	15ns (66MHz)	10ns (100MHz)

Cache designs were originally asynchronous, meaning they ran at a clock speed that was not identical or in sync with the processor bus. Starting with the 430FX chipset released in early '95, a new type of synchronous cache design was supported. This required that the chips now run in sync or at the same identical clock timing as the processor bus, further improving speed and performance. Also added at that time was a feature called Pipeline Burst mode, which reduced overall cache latency (wait states) by allowing for single-cycle accesses for multiple transfers after the first one. Because both synchronous and pipeline burst capability came at the same time in new modules, usually specifying one implies the other. Overall synchronous pipeline burst cache allowed for about a 20% improvement in overall system performance, which was a very significant jump.

The cache controller for a modern system is contained within either the North Bridge of the chipset as with Pentium and lesser systems, or within the processor as with the Pentium Pro/II and newer

systems. The capabilities of the cache controller dictate the performance and capabilities of the cache. One important thing to note is that most cache controllers have a limitation of the amount of memory that may be cached. Often, this limit can be quite low, as with the 430TX chipset-based Pentium systems. The 430TX chipset can cache data only within the first 64M of system RAM. If you have more memory than that, you will experience a noticeable slowdown in system performance because all data outside the first 64M will never be cached, and would always be accessed with all the wait states required by the slower DRAM. Depending on what software you use and where data is stored in memory, this can be very significant. For example, 32-bit operating systems such as Win95/98 and NT load from the top down, so if you had 96M of RAM, the operating system and applications would be loading directly into the upper 32M (past 64M), which is not cached. This would result in a dramatic slowdown in overall system use. Removing the additional memory to bring the system total down to the cacheable limit of 64M would be the solution. In short, it is unwise to install more main RAM memory than your system (chipset) can cache. Consult your system documentation or the chipset section in Chapter 4, "Motherboards and Buses," for more information.

Physical Memory

The CPU and motherboard architecture (chipset) dictates a particular computer's physical memory capacity, and the types and forms of memory that can be installed. Over the years, there have been two main changes occurring with computer memory--it has gradually become faster and wider! The speed and width requirements are indicated by the CPU and the memory controller circuitry. The memory controller in a modern PC resides in the motherboard chipset. Even though a system may physically support a given amount of memory, the type of software you run may dictate whether or not all the memory can be used.

The 8088 and 8086, with 20 address lines, can use as much as 1M (1024K) of RAM. The 286 and 386SX CPUs have 24 address lines; they can keep track of as much as 16M of memory. The 386DX, 486, Pentium, Pentium-MMX, and Pentium Pro CPUs have a full set of 32 address lines; they can keep track of 4G of memory, while the Pentium II with 36 address lines can manage an impressive 64G!

See "Processor Specifications," p. 31

When the 286 and higher chips emulate the 8088 chip (as they do when running 16-bit software such as DOS or Windows 3.x), they implement a hardware operating mode called *real mode*. Real mode is the only mode available on the 8086 and 8088 chips used in PC and XT systems. In real mode, all Intel processors--even the mighty Pentium--are restricted to using only 1M of memory, just as their 8086 and 8088 ancestors were, and the system design reserves 384K of that amount. Only in protected mode can the 286 or better chips use their maximum potential for memory addressing.

See "Processor Modes," p. 43

Pentium-based systems can address as much as 4G of memory, and Pentium Pro/II systems can address 64G. To put these memory-addressing capabilities into perspective, 64G (65,536M) of memory would cost about \$100,000! Even if you could afford all this memory, some of the largest memory modules available today are 168-pin DIMMs with 256M capacity. To install 64G of RAM would require 256 of the largest 256M DIMMs available, and most systems today can support up to only four or maybe eight DIMM sockets.

Most Pentium II motherboards have a maximum of only three to six DIMM sockets, which allows a maximum of .75-1.5G if all the sockets are filled. These limitations are from the chipset, not the processor. Technically, a Pentium can address 4G of memory and a Pentium II can address 64G, but there isn't a chipset on the market that will allow that! Most of the PII chipsets today are limited to 1G of memory.

Pentium systems have even further limitations. Pentium systems have been available since '93, but only those built in '97 or later use a motherboard chipset that will support SDRAM DIMMs. Even those using the newest 430TX chipset from Intel will not support more than 256M of total memory, and should not have more than 64M actually installed due to memory-caching limitations. Don't install more than 64M of RAM in your Pentium system unless you are sure the motherboard and chipset will allow the L2 cache to function with the additional memory. See the chipset section in Chapter 4 for the maximum cacheable limits on all the Intel and other motherboard chipsets.

Older 386 and 486 motherboards may have problems addressing memory past 16M due to DMA (Direct Memory Access) controller problems. If you install an ISA adapter that uses a DMA channel (such as a busmaster SCSI adapter) and you have more than 16M of memory, you have the potential for problems because the ISA bus allows only DMA access to 16M. Attempted transfers beyond 16M cause the system to crash. This should not be an issue with newer 32-bit operating systems, and 32-bit slots such as PCI. The 32-bit operating systems will automatically remap ISA bus DMA transfers beyond 16M properly, and PCI simply doesn't have any such limitations.

SIMMs and DIMMs

Originally, systems had memory installed via individual chips. These are often referred to as DIP (Dual Inline Package) chips because of their design. The original IBM XT and AT had 36 sockets on the motherboard for these individual chips, and then more of them would be installed on various memory cards plugged into the bus slots. I remember spending hours populating boards with these chips, which was a tedious job.

Besides being a time-consuming and labor-intensive way to deal with memory, DIP chips had one notorious problem--they would creep out of their sockets over time as the system would go through thermal cycles. Every day when you powered the system on and off, it would heat and cool, and the chips would gradually walk their way out of the sockets. Eventually, good contact would be lost and memory errors would result. Fortunately, reseating all the chips back in their sockets would usually rectify the problem, but that was labor-intensive if you had a lot of systems to support.

The alternative to this at the time was to have the memory soldered into either the motherboard or an expansion card. This prevented the chips from creeping and made the connections more permanent, but that caused another problem. If a chip did go bad, you would either have to attempt desoldering and resoldering a new one, or resort to scrapping the motherboard or memory card in which the chip was installed. This was expensive, and it made memory troubleshooting very difficult.

What was needed was a chip that was both soldered yet removable, and that is exactly what was found in the chip that we call a SIMM. For memory storage, most modern systems have adopted the single inline memory module (SIMM) or dual inline memory module (DIMM) as an alternative to individual memory chips. These small boards plug into special connectors on a motherboard or memory card. The individual memory chips are soldered to the SIMM/DIMM, so removing and

replacing individual memory chips is impossible. Instead, you must replace the entire module if any part of it fails. The SIMM/DIMM is treated as though it were one large memory chip.

PC systems use two main physical types of SIMMs--30-pin (8 bits plus one optional parity bit) and 72-pin (32 bits plus four optional parity bits)--with various capacities and other specifications. The 30-pin SIMMs are smaller than the 72-pin versions, and may have chips on either one or both sides. 30-pin SIMMs are basically obsolete, and they are being followed rapidly by the 72-pin versions. This is true primarily because the 64-bit systems, which are now the industry standard, would require eight 30-pin SIMMs or two 72-pin SIMMs per bank. DIMMs, which have become popular on Pentium-MMX and Pentium Pro-based systems, are 168-pin units with 64-bit (non-parity) or 72-bit (parity or ECC) data paths.

Figures 5.5, 5.6, and 5.7 show typical 30-pin (8-bit) and 72-pin (32-bit) SIMMs, and a 168-pin (64-bit) DIMM, respectively. The pins are numbered from left to right and are connected through to both sides of the module on the SIMMs. The pins on the DIMM are different on each side. Note that all dimensions are in both inches and millimeters (in parentheses).

FIG. 5.5 *A typical 30-pin SIMM. The one shown here is 9-bit, although the dimensions would be the same for 8-bit as well.*

FIG. 5.6 *A typical 72-pin SIMM. The one shown here is 36-bit, although the dimensions would be the same for 32-bit as well.*

FIG. 5.7 *A typical 168-pin DIMM. The one shown here is 72-bit, although the dimensions would be the same for 64-bit as well.*

Single inline pinned packages, sometimes called SIPPs, really are SIMMs with pins, not contacts. The pins are designed to be installed in a long connector socket that is much cheaper than the standard SIMM socket. SIPPs are inferior to SIMMs because they lack the positive latching mechanism that retains the module, and the connector lacks the high-force wiping contacts that resist corrosion. SIPPs are rarely used today.

NOTE

It would be possible to convert a SIPP to a SIMM by cutting off the pins, or to convert a SIMM to a SIPP by soldering pins on. Also, some companies have made SIPP-to-SIMM converters that allow the SIPPs to be plugged into 30-pin SIMM sockets.

These memory modules are extremely compact considering the amount of memory they hold. SIMMs and DIMMs are available in several capacities and speeds. Table 5.4 lists the different capacities available for both the 30-pin and 72-pin SIMMs, and 168-pin DIMMs.

Table 5.4 SIMM and DIMM Capacities

30-Pin SIMM Capacities

Capacity	Parity SIMM	Non-Parity SIMM
256K	256Kx9	256Kx8
1M	1Mx9	1Mx8
4M	4Mx9	4Mx8
16M	16Mx9	16Mx8

72-Pin SIMM Capacities

Capacity	Parity SIMM	Non-Parity SIMM
1M	256Kx36	256Kx32
2M	512Kx36	512Kx32
4M	1Mx36	1Mx32
8M	2Mx36	2Mx32
16M	4Mx36	4Mx32
32M	8Mx36	8Mx32
64M	16Mx36	16Mx32
128M	32Mx36	32Mx32

168-Pin DIMM Capacities

Capacity	Parity DIMM	Non-Parity DIMM
8M	1Mx72	1Mx64
16M	2Mx72	2Mx64
32M	4Mx72	4Mx64
64M	8Mx72	8Mx64
128M	16Mx72	16Mx64
256M	32Mx72	32Mx64

Dynamic RAM (DRAM) SIMMs and DIMMs of each type and capacity are available in different speed ratings. SIMMs have been available in many different speed ratings, ranging from 120ns for some of the slowest, to 50ns for some of the fastest available. DIMMs are available in speeds from 60ns to 10ns or faster. Many of the first systems to use SIMMs used versions rated at 120ns. These were quickly replaced in the market by 100ns and even faster versions. Today, you generally purchase EDO SIMMs rated at 60ns, and SDRAM DIMMs rated at 10ns. Both faster and slower ones are available, but they are not frequently required and are difficult to obtain.

If a system requires a specific speed, you can almost always substitute faster speeds if the one specified is not available. There are no problems in mixing SIMM speeds, as long as you use SIMMs equal to or faster than what the system requires. Because very little price difference exists between the different speed versions, I usually buy faster SIMMs than are needed for a particular application. This may make them more usable in a future system that may require the faster speed.

NOTE

Most DIMMs are Synchronous DRAM (SDRAM) memory, which means they deliver data in very high-speed bursts using a clocked interface. SDRAM supports bus speeds of up to 100MHz, with data transfer rates of up to 200MHz possible in the future.

Several variations on the 30-pin SIMMs can affect how they work (if at all) in a particular system. First, there are actually two variations on the pinout configurations. Most systems use a *generic* type of SIMM, which has an industry-standard pin configuration. Many older IBM systems used a slightly modified 30-pin SIMM, starting with the XT-286 introduced in 1986 through the PS/2 Model 25, 30, 50, and 60. These systems require a SIMM with different signals on five of the pins. These are known as *IBM-style* 30-pin SIMMs. You can modify a generic 30-pin SIMM to work in the IBM systems and vice versa, but purchasing a SIMM with the correct pinouts is much easier. Be sure you tell the SIMM vendor if you need the specific IBM-style versions.

Another issue with respect to the 30-pin SIMMs relates to the chip count. The SIMM acts as if it were a single chip of 8 bits wide (with optional parity), and it really does not matter how this total is derived. Older SIMMs were constructed with eight or nine individual 1-bit-wide chips to make up the module, whereas many newer SIMMs use two 4-bit-wide chips and optionally one 1-bit-wide chip for parity, making a total of two or three chips on the SIMM. Accessing the two- or three-chip SIMMs can require adjustments to the refresh timing circuits on the motherboard, and many older 386 and 486 motherboards could not cope. Most newer motherboards automatically handle the slightly different refresh timing of both the 2/3-chip or 8/9-chip SIMMs, and in this case the 2/3-chip versions are more reliable, use less power, and generally cost less as well. If you have an older system, most likely it will also work with the 2/3-chip SIMMs, but some do not. Unfortunately, the only way to know is to try them. To prevent the additional time required to change them for 8/9-chip versions should the 2/3-chip versions not work in an older system, it seems wise to stick with the 8/9-chip variety in any older system.

The 72-pin SIMMs do not have different pinouts and are differentiated only by capacity and speed. These SIMMs are not affected by the number of chips on them. The 72-pin SIMMs are ideal for 32-bit systems such as 486 machines because they comprise an entire bank of memory (32 data bits plus 4 parity bits). When you configure a 32-bit (486) system that uses 72-pin SIMMs, you can usually add or remove memory as single SIMM modules (except on systems that use interleaved memory schemes to reduce wait states). This is because a 32-bit chip such as a 486 reads and writes banks of memory 32 bits wide, and a 72-pin SIMM is exactly 32 bits wide (36 bits with optional parity).

In 64-bit systems--which includes any Pentium or newer processor--72-pin SIMMs must be used in pairs to fill a single bank. A few motherboard manufacturers offer so-called "SIMM-saver" motherboards that are designed for newer Pentium processors, but have both 72- and 30-pin SIMM sockets. Although this is not the most desirable arrangement, it allowed users on a budget to reuse their old 30-pin SIMMs. In this situation, eight 30-pin SIMMs can be used at a time to fill one bank. Alternatively, you could pair four 30-pin SIMMs with one 72-pin SIMM to create one bank. This really is not a very efficient setup because it consumes large amounts of space on the motherboard.

Other options available are SIMM stackers and converters. These items allow you to use 30-pin SIMMs in 72-pin sockets, thereby saving you the expense of having to scrap all those old 30-pin SIMMs you have lying around. Again, such adapters can cause problems--especially if overhead clearance is tight--so investigate carefully before you buy. With the falling prices of SIMMs today, you are probably better off staying with 72-pin SIMMs and 168-pin DIMMs.

Remember that some older high-performance 486 systems use interleaved memory to reduce wait states. This requires a multiple of two 72-pin 36-bit (32-bit) SIMMs because interleaved memory access is alternated between the SIMMs to improve performance. Thus, a 32-bit processor ends up using two 32-bit banks together in an alternating fashion.

NOTE

A bank is the smallest amount of memory that can be addressed by the processor at one time and usually corresponds to the data bus width of the processor. If the memory is interleaved, a virtual bank may be twice the absolute data bus width of the processor.

You cannot always replace a SIMM with a greater-capacity unit and expect it to work. Systems may have specific design limitations as to the maximum capacity of SIMM they will take. A larger-capacity SIMM works only if the motherboard is designed to accept it in the first place. Consult your system documentation to determine the correct capacity and speed to use.

All systems on the market today use SIMMs, and many use DIMMs. The SIMM/DIMM is not a proprietary memory system, but an industry-standard device. As mentioned, some SIMMs have slightly different pinouts and specifications other than speed and capacity, so be sure that you obtain the correct SIMMs for your system.

SIMM Pinouts

Tables 5.5 and 5.6 show the interface connector pinouts for both 30-pin SIMM varieties and the standard 72-pin version. Also included is a special presence detect table that shows the configuration of the presence detect pins on various 72-pin SIMMs. The presence detect pins are used by the motherboard to detect exactly what size and speed SIMM is installed. Industry-standard 30-pin SIMMs do not have a presence detect feature, but IBM did add this capability to their modified 30-pin configuration.

Table 5.5 Industry-Standard and IBM 30-Pin SIMM Pinouts

Pin	Standard SIMM Signal Names	IBM SIMM Signal Names
1	+5 Vdc	+5 Vdc
2	Column Address Strobe	Column Address Strobe
3	Data Bit 0	Data Bit 0
4	Address Bit 0	Address Bit 0
5	Address Bit 1	Address Bit 1
6	Data Bit 1	Data Bit 1
7	Address Bit 2	Address Bit 2
8	Address Bit 3	Address Bit 3
9	Ground	Ground
10	Data Bit 2	Data Bit 2
11	Address Bit 4	Address Bit 4
12	Address Bit 5	Address Bit 5

13	Data Bit 3	Data Bit 3
14	Address Bit 6	Address Bit 6
15	Address Bit 7	Address Bit 7
16	Data Bit 4	Data Bit 4
17	Address Bit 8	Address Bit 8
18	Address Bit 9	Address Bit 9
19	Address Bit 10	Row Address Strobe 1
20	Data Bit 5	Data Bit 5
21	Write Enable	Write Enable
22	Ground	Ground
23	Data Bit 6	Data Bit 6
24	No Connection	Presence Detect (Ground)
25	Data Bit 7	Data Bit 7
26	Data Bit 8 (Parity) Out	Presence Detect (1M = Ground)
27	Row Address Strobe	Row Address Strobe
28	Column Address Strobe Parity	No Connection
29	Data Bit 8 (Parity) In	Data Bit 8 (Parity) I/O
30	+5 Vdc	+5 Vdc

Table 5.6 Standard 72-Pin SIMM Pinout

Pin	SIMM Signal Name	Pin	SIMM Signal Name
1	Ground	22	Data Bit 5
2	Data Bit 0	23	Data Bit 21
3	Data Bit 16	24	Data Bit 6
4	Data Bit 1	25	Data Bit 22
5	Data Bit 17	26	Data Bit 7
6	Data Bit 2	27	Data Bit 23
7	Data Bit 18	28	Address Bit 7
8	Data Bit 3	29	Address Bit 11
9	Data Bit 18	30	+5 Vdc
10	+5 Vdc	31	Address Bit 8
11	Presence Detect 5	32	Address Bit 9
12	Address Bit 0	33	Row Address Strobe 3
13	Address Bit 1	34	Row Address Strobe 2
14	Address Bit 2	35	Parity Data Bit 2
15	Address Bit 3	36	Parity Data Bit 0
16	Address Bit 4	37	Parity Data Bit 1
17	Address Bit 5	38	Parity Data Bit 3

18	Address Bit 6	39	Ground
19	Address Bit 10	40	Column Address Strobe 0
20	Data Bit 4	41	Column Address Strobe 2
21	Data Bit 20	42	Column Address Strobe 3
43	Column Address Strobe 1	58	Data Bit 28
44	Row Address Strobe 0	59	+5 Vdc
45	Row Address Strobe 1	60	Data Bit 29
46	Reserved	61	Data Bit 13
47	Write Enable	62	Data Bit 30
48	ECC Optimized	63	Data Bit 14
49	Data Bit 8	64	Data Bit 31
50	Data Bit 24	65	Data Bit 15
51	Data Bit 9	66	Reserved
52	Data Bit 25	67	Presence Detect 1
53	Data Bit 10	68	Presence Detect 2
54	Data Bit 26	69	Presence Detect 3
55	Data Bit 11	70	Presence Detect 4
56	Data Bit 27	71	Reserved
57	Data Bit 12	72	Ground

Notice that the 72-pin SIMMs employ a set of four or five pins to indicate the type of SIMM to the motherboard. These presence detect pins are either grounded or not connected to indicate the type of SIMM to the motherboard. Presence detect outputs must be tied to the ground through a zero-ohm resistor on the SIMM--to generate a high logic level when the pin is open or low logic level when the pin is grounded by the mother-board. This produces signals that are decodable by the memory interface logic. If presence detect signals are employed by the motherboard, then a Power-On Self Test procedure can determine the size and speed of the installed SIMMs and adjust control and addressing signals automatically. This allows the memory size and speed to be autodetected.

NOTE

In many ways, this is similar to the industry-standard DX code used on modern 35mm film rolls to indicate the ASA (speed) rating of the film to the camera. When you drop the film into the camera, electrical contacts can read the film's speed rating via an industry standard configuration.

Table 5.7 shows the JEDEC industry standard presence detect configuration listing for the 72-pin SIMM family. JEDEC is the Joint Electronic Devices Engineering Council, an organization of the U.S. semiconductor manufacturers and users that sets package outline dimension and other standards for chip and module packages.

Table 5.7 Presence Detect Pin Configurations for 72-Pin SIMMs

Size	Speed	Pin 67	Pin 68	Pin 69	Pin 70	Pin 11
1M	100ns	Gnd	-	Gnd	Gnd	-
1M	80ns	Gnd	-	-	Gnd	-
1M	70ns	Gnd	-	Gnd	-	-
1M	60ns	Gnd	-	-	-	-
2M	100ns	-	Gnd	Gnd	Gnd	-
2M	80ns	-	Gnd	-	Gnd	-
2M	70ns	-	Gnd	Gnd	-	-
2M	60ns	-	Gnd	-	-	-
4M	100ns	Gnd	Gnd	Gnd	Gnd	-
4M	80ns	Gnd	Gnd	-	Gnd	-
4M	70ns	Gnd	Gnd	Gnd	-	-
4M	60ns	Gnd	Gnd	-	-	-
8M	100ns	-	-	Gnd	Gnd	-
8M	80ns	-	-	-	Gnd	-
8M	70ns	-	-	Gnd	-	-
8M	60ns	-	-	-	-	-
16M	80ns	Gnd	-	-	Gnd	Gnd
16M	70ns	Gnd	-	Gnd	-	Gnd
16M	60ns	Gnd	-	-	-	Gnd
16M	50ns	Gnd	-	Gnd	Gnd	Gnd
32M	80ns	-	Gnd	-	Gnd	Gnd
32M	70ns	-	Gnd	Gnd	-	Gnd
32M	60ns	-	Gnd	-	-	Gnd
32M	50ns	-	Gnd	Gnd	Gnd	Gnd

- = No Connection (open)

Gnd = Ground

Pin 67 = Presence detect 1

Pin 68 = Presence detect 2

Pin 69 = Presence detect 3

Pin 70 = Presence detect 4 Pin 11 = Presence detect 5

Unfortunately, unlike the film industry, not everybody in the computer industry follows established standards. As such, presence detect signaling is not a standard throughout the PC industry. Different system manufacturers sometimes use different configurations for what is expected on these four pins. Compaq, IBM (mainly PS/2 systems), and Hewlett-Packard are notorious for this type of behavior;

many of their systems require special SIMMs that are basically the same as standard 72-pin SIMMs, except for special presence detect requirements. Table 5.8 shows how IBM defines these pins.

Table 5.8 72-Pin SIMM Presence Detect Pins

67	68	69	70	SIMM Type	IBM Part Number
-	-	-	-	Not a valid SIMM	N/A
Gnd	-	-	-	1M 120ns	N/A
-	Gnd	-	-	2M 120ns	N/A
Gnd	Gnd	-	-	2M 70ns	92F0102
-	-	Gnd	-	8M 70ns	64F3606
Gnd	-	Gnd	-	Reserved	N/A
-	Gnd	Gnd	-	2M 80ns	92F0103
Gnd	Gnd	Gnd	-	8M 80ns	64F3607
-	-	-	Gnd	Reserved	N/A
Gnd	-	-	Gnd	1M 85ns	90X8624
-	Gnd	-	Gnd	2M 85ns	92F0104
Gnd	Gnd	-	Gnd	4M 70ns	92F0105
-	-	Gnd	Gnd	4M 85ns	79F1003 (square notch) L40-SX
Gnd	-	Gnd	Gnd	1M 100ns	N/A
Gnd	-	Gnd	Gnd	8M 80ns	79F1004 (square notch) L40-SX
-	Gnd	Gnd	Gnd	2M 100ns	N/A
Gnd	Gnd	Gnd	Gnd	4M 80ns	87F9980
Gnd	Gnd	Gnd	Gnd	2M 85ns	79F1003 (square notch) L40SX

- = No Connection (open)

Gnd = Ground

Pin 67 = Presence detect 1

Pin 68 = Presence detect 2

Pin 69 = Presence detect 3

Pin 70 = Presence detect 4

Custom variations on these pins are why you must often specify IBM, Compaq, HP, or generic SIMMs when you order memory.

Table 5.9 shows the pinout configuration of a 168-pin standard unbuffered SDRAM DIMM.

Table 5.9 168-Pin SDRAM DIMM Pinouts

Pin	x64 Non-Parity	x72 Parity/ECC	Pin	x64 Non-Parity	x72 Parity/ECC
1	Gnd	Gnd	85	Gnd	Gnd
2	Data Bit 0	Data Bit 0	86	Data Bit 32	Data Bit 32
3	Data Bit 1	Data Bit 1	87	Data Bit 33	Data Bit 33
4	Data Bit 2	Data Bit 2	88	Data Bit 34	Data Bit 34
5	Data Bit 3	Data Bit 3	89	Data Bit 35	Data Bit 35
6	+5V	+5V	90	+5V	+5V
7	Data Bit 4	Data Bit 4	91	Data Bit 36	Data Bit 36
8	Data Bit 5	Data Bit 5	92	Data Bit 37	Data Bit 37
9	Data Bit 6	Data Bit 6	93	Data Bit 38	Data Bit 38
10	Data Bit 7	Data Bit 7	94	Data Bit 39	Data Bit 39
11	Data Bit 8	Data Bit 8	95	Data Bit 40	Data Bit 40
12	Gnd	Gnd	96	Gnd	Gnd
13	Data Bit 9	Data Bit 9	97	Data Bit 41	Data Bit 41
14	Data Bit 10	Data Bit 10	98	Data Bit 42	Data Bit 42
15	Data Bit 11	Data Bit 11	99	Data Bit 43	Data Bit 43
16	Data Bit 12	Data Bit 12	100	Data Bit 44	Data Bit 44
17	Data Bit 13	Data Bit 13	101	Data Bit 45	Data Bit 45
18	+5V	+5V	102	+5V	+5V
19	Data Bit 14	Data Bit 14	103	Data Bit 46	Data Bit 46
20	Data Bit 15	Data Bit 15	104	Data Bit 47	Data Bit 47
21	-	Check Bit 0	105	-	Check Bit 4
22	-	Check Bit 1	106	-	Check Bit 5
23	Gnd	Gnd	107	Gnd	Gnd
24	-	-	108	-	-
25	-	-	109	-	-
26	+5V	+5V	110	+5V	+5V
27	Write Enable	Write Enable	111	Column Address Strobe	Column Address Strobe
28	Byte Mask 0	Byte Mask 0	112	Byte Mask 4	Byte Mask 4
29	Byte Mask 1	Byte Mask 1	113	Byte Mask 5	Byte Mask 5
30	S0	S0	114	S1	S1
31	Reserved	Reserved	115	Row Address Strobe	Row Address Strobe
32	Gnd	Gnd	116	Gnd	Gnd
33	Address Bit 0	Address Bit 0	117	Address Bit 1	Address Bit 1
34	Address Bit 2	Address Bit 2	118	Address Bit 3	Address Bit 3

35	Address Bit 4	Address Bit 4	119	Address Bit 5	Address Bit 5
36	Address Bit 6	Address Bit 6	120	Address Bit 7	Address Bit 7
37	Address Bit 8	Address Bit 8	121	Address Bit 9	Address Bit 9
38	Address Bit 10	Address Bit 10	122	Bank Address 0	Bank Address 0
39	Bank Address 1	Bank Address 1	123	Address Bit 11	Address Bit 11
40	+5V	+5V	124	+5V	+5V
41	+5V	+5V	125	Clock 1	Clock 1
42	Clock 0	Clock 0	126	Address Bit 12	Address Bit 12
43	Gnd	Gnd	127	Gnd	Gnd
44	Reserved	Reserved	128	Clock Enable 0	Clock Enable 0
45	S2	S2	129	S3	S3
46	Byte Mask 2	Byte Mask 2	130	Byte Mask 6	Byte Mask 6
47	Byte Mask 3	Byte Mask 3	131	Byte Mask 7	Byte Mask 7
48	Reserved	Reserved	132	Address Bit 13	Address Bit 13
49	+5V	+5V	133	+5V	+5V
50	-	-	134	-	-
51	-	-	135	-	-
52	-	Check Bit 2	136	-	Check Bit 6
53	-	Check Bit 3	137	-	Check Bit 7
54	Gnd	Gnd	138	Gnd	Gnd
55	Data Bit 16	Data Bit 16	139	Data Bit 48	Data Bit 48
56	Data Bit 17	Data Bit 17	140	Data Bit 49	Data Bit 49
57	Data Bit 18	Data Bit 18	141	Data Bit 50	Data Bit 50
58	Data Bit 19	Data Bit 19	142	Data Bit 51	Data Bit 51
59	+5V	+5V	143	+5V	+5V
60	Data Bit 20	Data Bit 20	144	Data Bit 52	Data Bit 52
61	-	-	145	-	-
62	Voltage Reference	Voltage Reference	146	Voltage Reference	Voltage Reference
63	Clock Enable 1	Clock Enable 1	147	-	-
64	Gnd	Gnd	148	Gnd	Gnd
65	Data Bit 21	Data Bit 21	149	Data Bit 53	Data Bit 53
66	Data Bit 22	Data Bit 22	150	Data Bit 54	Data Bit 54
67	Data Bit 23	Data Bit 23	151	Data Bit 55	Data Bit 55
68	Gnd	Gnd	152	Gnd	Gnd
69	Data Bit 24	Data Bit 24	153	Data Bit 56	Data Bit 56
70	Data Bit 25	Data Bit 25	154	Data Bit 57	Data Bit 57
71	Data Bit 26	Data Bit 26	155	Data Bit 58	Data Bit 58

72	Data Bit 27	Data Bit 27	156	Data Bit 59	Data Bit 59
73	+5V	+5V	157	+5V	+5V
74	Data Bit 28	Data Bit 28	158	Data Bit 60	Data Bit 60
75	Data Bit 29	Data Bit 29	159	Data Bit 61	Data Bit 61
76	Data Bit 30	Data Bit 30	160	Data Bit 62	Data Bit 62
77	Data Bit 31	Data Bit 31	161	Data Bit 63	Data Bit 63
78	Gnd	Gnd	162	Gnd	Gnd
79	Clock 2	Clock 2	163	Clock 3	Clock 3
80	-	-	164	-	-
81	-	-	165	Serial PD Address 0	Serial PD Address 0
82	Serial Data I/O	Serial Data I/O	166	Serial PD Address 1	Serial PD Address 1
83	Serial Clock Input	Serial Clock Input	167	Serial PD Address 2	Serial PD Address 2
84	+5V	+5V	168	+5V	+5V

- = No Connection (open)

Gnd = Ground

The DIMM uses a completely different type of Presence Detect called Serial Presence Detect. This consists of a small EEPROM (Electrically Erasable Programmable Read Only Memory) or even a flash memory chip on the DIMM that contains specially formatted data indicating the features of the DIMM. This serial data can be read via the serial data pins on the DIMM, and allows the motherboard to autoconfigure to the exact type of DIMM installed.

Physical RAM Capacity and Organization

Several types of memory chips have been used in PC system motherboards. Most of these chips are single-bit-wide chips, available in several capacities. The following table lists available RAM chips and their capacities:

RAM Chip	Capacity
16K by 1 bit	These devices were used in the original IBM PC with a Type 1 motherboard.
64K by 1 bit	These chips were used in the standard IBM PC Type 2 motherboard and in the XT Type 1 and 2 motherboards. Many memory adapters of the era, such as the popular vintage AST six-pack boards, also used these chips.
128K by 1 bit	These chips, used in the IBM AT Type 1 motherboard, often were a strange physical combination of two 64K chips stacked one on top of the other and soldered together. Single-chip versions were used also for storing the parity bits in the IBM XT 286.
256K by 1 bit (or 64K by 4 bits)	These chips once were very popular in motherboards and memory cards. The IBM XT Type 2 and IBM AT Type 2 motherboards, and most compatible systems of that era used these chips.
1M by 1 bit (or 256K by 4 bits)	1M chips were very popular for a number of years and were most often used in 256K to 8M SIMMs.

4M by 1 bit (or 1M by 4 bits)	4M chips are used primarily in SIMMs from 1M to 16M in capacity. They are used primarily in 4M and 8M SIMMs and generally are not sold as individual chips.
16M by 1 bit (or 4M by 4 bits)	16M chips are often used in 72-pin SIMMs of 16M to 32M capacity.
64M by 1 bit (or 16M by 4 bits)	64M chips are popular in high-capacity 16M or larger memory modules, especially for notebook systems.
256M by 1 bit (or 64M by 4 bits)	256M chips are the most recent on the market. These chips allow enormous SIMM capacities of 128M or larger! Because of the high expense and limited availability of these chips, you see them only in the most expensive and highest capacity modules on the market.

Note that chip capacity normally goes up by factors of 4 because the die that make up the chips are square. When they increase capacity, this normally results in 4 times more transistors and 4 times more capacity. Most modern SIMMs and DIMMs use 16M-bit to 256M-bit chips onboard.

Figure 5.8 shows the markings on a typical older memory chip. Many memory manufacturers use similar schemes for numbering their chips; however, if you are really trying to identify a particular chip, it is best to consult the manufacturer catalog for more information.

FIG. 5.8 *The markings on a typical older memory chip.*

The -10 on the chip corresponds to its speed in nanoseconds (a 100-nanosecond rating). MB81256 is the chip's part number, which usually contains a clue about the chip's capacity. The key digits are 1256, which indicate that this chip is 1 bit wide and has a depth of 256K. The 1 means that to make a full byte with parity, you need nine of these single-bit-wide chips. A chip with a part number KM4164B-10 indicates a 64K-by-1-bit chip at a speed of 100 nanoseconds. The following list matches common chips with their industry-standard part numbers:

Part Number	Chip
4164	64K by 1 bit
4464	64K by 4 bits
41128	128K by 1 bit
44128	128K by 4 bits
41256	256K by 1 bit
44256	256K by 4 bits
41000	1M by 1 bit
44000	1M by 4 bits

Chips wider than 1 bit are used to construct banks of less than 9, 18, or 36 chips (depending on the system architecture). For example, a 32-bit SIMM can be constructed of only eight 4-bit wide chips.

In Figure 5.8, the "F" symbol centered between two lines is the manufacturer's logo for Fujitsu Microelectronics. The 8609 indicates the date of manufacture (ninth week of 1986). Some

manufacturers, however, use a Julian date code. To decode the chip further, contact the manufacturer or chip vendor.

SIMMs and DIMMs also have part numbers that can be difficult to decode. Unfortunately, there is no industry standard for numbering these modules, so you will have to contact the manufacturer if you want to decode the numbers. Figure 5.9 shows how Micron Technology (also Crucial Technology) SIMMs are encoded.

FIG. 5.9 Typical SIMM part numbers for Micron (Crucial Technology) SIMMs.

Memory Banks

Memory chips (DIPs, SIMMs, SIPPs, and DIMMs) are organized in *banks* on motherboards and memory cards. You should know the memory bank layout and position on the motherboard and memory cards.

You need to know the bank layout when adding memory to the system. In addition, memory diagnostics report error locations by byte and bit addresses, and you must use these numbers to locate which bank in your system contains the problem.

The banks usually correspond to the data bus capacity of the system's microprocessor. Table 5.10 shows the widths of individual banks based on the type of PC.

Table 5.10 Memory Bank Widths on Different Systems

Processor	Data Bus	Memory Bank Size (No Parity)	Memory Bank Size (Parity)	30- Pin SIMMs perBank	72- Pin SIMMs per Bank	168-Pin SIMMs per Bank
8088	8-bit	8-bits	9-bits	1	N/A	N/A
8086	16-bit	16-bits	18-bits	2	N/A	N/A
286	16-bit	16-bits	18-bits	2	N/A	N/A
386SX, SL, SLC	16-bit	16-bits	18-bits	2	N/A	N/A
386DX	32-bit	32-bits	36-bits	4	1	N/A
486SLC, SLC2	16-bit	16-bits	18-bits	2	N/A	N/A
486SX, DX, DX2, DX4	32-bit	32-bits	36-bits	4	1	N/A
Pentium	64-bit	64-bits	72-bits	8	2	1
Pentium Pro, PII	64-bit	64-bits	72-bits	8	2	1

The number of bits for each bank can be made up of single chips, SIMMs, or DIMMs. Modern systems don't use individual chips, but only SIMMs or DIMMs, instead. If the system has a 16-bit processor such as a 386SX, it would probably use 30-pin SIMMs and have two SIMMs per bank. All the SIMMs in a single bank must be the same size and type.

A 486 system would require four 30-pin SIMMs or one 72-pin SIMM to make up a bank. A single 72-pin SIMM is 32 bits wide or 36 bits wide if it supports parity. You can often tell whether or not a SIMM supports parity by counting its chips. To make a 32-bit SIMM, you could use 32 individual 1-bit-wide chips, or you could use eight individual 4-bit-wide chips to make up the data bits. If the system used parity, then four extra bits would be required (36 bits total), so you would see one more 4-bit-wide or four individual 1-bit-wide chips added to the bank for the parity bits.

As you might imagine, 30-pin SIMMs are less than ideal for 32-bit or 64-bit systems (i.e., 486 or Pentium) because you must use them in increments of four or eight per bank! Because these SIMMs are available in 1M, 4M, and 16M capacities today, this means that a single bank of 30-pin SIMMs in a 486 system would be 4M, 16M, or 64M; for a Pentium system, this would be 8M, 32M, or 128M of memory, with no in-between amounts. Using 30-pin SIMMs in 32- and 64-bit systems artificially constrains memory configurations and such systems are not recommended. If a 32-bit system (such as any PC with a 486 processor) uses 72-pin SIMMs, each SIMM represents a separate bank, and the SIMMs can be added or removed on an individual basis rather than in groups of four, as would be required with 30-pin SIMMs. This makes memory configuration much easier and more flexible. In modern 64-bit systems that use SIMMs, two 72-pin SIMMs are required per bank.

DIMMs are ideal for Pentium and higher systems, as the 64-bit width of the DIMM exactly matches the 64-bit width of the Pentium processor data bus. This means that each DIMM represents an individual bank, and they can be added or removed one at a time.

The physical orientation and numbering of the SIMMs or DIMMs used on a motherboard is arbitrary and determined by the board's designers. Documentation covering your system or card comes in very handy. You can determine the layout of a motherboard or adapter card through testing, but this takes time and may be difficult, particularly after you have a problem with a system.

RAM Chip Speed

PC memory speeds vary from about 10ns to 200ns. When you replace a failed memory module, you must install a module of the same type and speed as the failed module. You can substitute a chip with a different speed only if the speed of the replacement chip is equal to or faster than that of the failed chip.

Some people have had problems when "mixing" chips because they used a chip that did not meet the minimum required specifications (for example, refresh timing specifications) or was incompatible in pinout, depth, width, or design. Chip access time always can be less (that is, faster) as long as your chip is the correct type and meets all other specifications.

Substituting faster memory usually doesn't provide improved performance because the system still operates the memory at the same speed. In systems not engineered with a great deal of "forgiveness" in the timing between memory and system, however, substituting faster memory chips might improve reliability.

To place more emphasis on timing and reliability, Intel has created standards for the new high-speed 100MHz memory supported by their newer chipsets. This is called the PC/100 standard, and it is a standard by which memory modules can be certified to perform within Intel's timing and performance guidelines. At 100MHz, there is not very much "forgiveness" or slop allowable in memory timing.

The same common symptoms result when the system memory has failed or is simply not fast enough for the system's timing. The usual symptoms are frequent parity check errors or a system that does not operate at all. The POST also might report errors. If you're unsure of what chips to buy for your system, contact the system manufacturer or a reputable chip supplier.

See "Parity Checking," p. 347

Gold Versus Tin

Many people don't understand the importance of the SIMM and DIMM electrical contacts in a computer system. Both SIMMs and DIMMs are available in gold or tin-plated contact form. I initially thought that gold contact SIMMs or DIMMs were the best way to go for reliability in all situations, but that is not true. To have the most reliable system, you must install SIMMs or DIMMs with gold-plated contacts into gold-plated sockets and SIMMs or DIMMs with tin-plated contacts into only tin-plated sockets.

If you don't heed this warning and install memory with gold-plated contacts into tin sockets or vice versa, you will experience memory errors at a future date. In my experiences, this will occur between six months to one year after installation. I have encountered this several times in my own systems and in several systems I have serviced. I have even been asked to assist one customer in a lawsuit, where the customer purchased several hundred machines from a vendor, and severe memory failures began appearing in most of the machines approximately a year after delivery. The cause was traced to dissimilar metals between the memory modules and sockets (gold SIMMs in tin sockets, in this case). The vendor refused to replace the SIMMs with tin-plated versions, hence the lawsuit.

Most Intel Pentium and 486 motherboards that were designed to accept 72-pin SIMMs have tin-plated sockets, so they must have tin-plated memory. Intel now specifically recommends *not* mixing dissimilar metals in a system's memory. Studies done by the connector manufacturers show that a type of corrosion called *fretting* occurs when tin comes in pressure contact with gold or any other metal. Fretting corrosion is where tin oxide transfers to the gold surface and hardens, eventually causing a high resistance connection. This occurs where gold comes into contact with tin, no matter how thick or thin the gold coating. Over time, depending on the environment fretting, corrosion can and will cause high resistance at the contact point and thus cause memory errors.

One would think that tin would make a poor connector material because it does readily oxidize. Even so, electrical contact is easily made between two tin surfaces under pressure because the oxides on both soft surfaces will bend and break, ensuring contact. This is especially true in SIMMs or DIMMs where a lot of pressure is placed on the contacts.

When tin and gold come into contact, because one surface is hard, the oxidation builds up and will not break easily under pressure. Increased contact resistance ultimately results in memory failures. The bottom line is that you should place only gold SIMMs into gold sockets, and only tin SIMMs into tin sockets.

The connector manufacturer AMP has published several documents from the AMP Contact Physics Research Department that discuss this issue, but there are two that are most applicable. One is titled *Golden Rules: Guidelines for the Use of Gold on Connector Contacts*, and the other is called *The Tin Commandments: Guidelines for the Use of Tin on Connector Contacts*. Both can be downloaded in

.PDF form from the AMP Web site at <http://www.ampincorporated.com/>. Here is an excerpt from the *Tin Commandments*, specifically commandment Number Seven, which states "7. Mating of tin-coated contacts to gold-coated contacts is not recommended." For further technical details, you can contact Intel or AMP at the sites I have recommended.

Certainly, the best type of setup would be gold to gold, as in having gold-plated SIMMs or DIMMs and gold-plated sockets on the motherboard. Most high-end file servers or other high-reliability systems are designed this way. In fact, most systems that require SDRAM DIMMs use gold-plated sockets and therefore require SDRAM DIMMs with gold-plated contacts.

If you have mixed metals in your memory now, the correct response would be to replace the memory modules with the appropriate contact type to match the socket. Another much less desirable solution would be to wait until problems appear (about six months to a year in my experience), then remove the modules, clean the contacts, reinstall, and repeat the cycle. This is probably fine if you are an individual with one or two systems, but it is *not* fine if you are maintaining hundreds of systems. Note that if your system does not feature parity or ECC memory (most systems sold today do not), when the problems do occur, you may not be able to immediately identify them as memory related (Global Protection Faults, crashes, file and data corruption, and so on).

One problem with cleaning is that the hard tin oxide deposits that form on the gold surface are difficult to remove, and often require abrasive cleaning (such as by using an eraser or crocus cloth). This should never be done dry because this will generate static discharges that can damage the chips. Instead, use a contact cleaner to lubricate the contacts, which when wet will minimize the potential for static discharge damage when rubbing the eraser or other abrasive on the surface.

To further forestall this problem from occurring, I highly recommend using a liquid contact enhancer and lubricant called Stabilant 22 from DW Electrochemicals when installing SIMMs or DIMMs. They have a detailed application note on their Web site on this subject if you are interested in more technical details.

Some people have accused me of being too picky by insisting that the memory match the socket. On several occasions, I have either returned memory or motherboards because the vendor putting them together did not have a clue this was a problem. When I tell some people about this, they tell me they have a lot of PCs out there with mixed metal contacts that run fine and, in many cases, have been doing so for many years.

Of course, that is certainly a poor argument against doing the right thing. There are a lot of people running SCSI buses that are way too long, with improper termination, and they say it runs "fine." Parallel port cables are by the spec limited to 10 feet in length, yet I see many have longer cables, which they claim work "fine." The IDE cable limit is 18 inches; that spec is violated and people get away with it, saying their drive runs just "fine." I see cheap, crummy power supplies that put out noise, hash, and loosely regulated voltages, and I have even measured up to 69 volts AC of ground leakage, and yet the systems were running "fine." I have encountered *numerous* systems without proper CPU heat sinks, or where the active heat sink (fan) was stalled, and the system ran "fine." This reminds me of when Johnny Carson would interview 100-year-old people and would often get them to admit that they drank heavily and smoked cigarettes every day, as if those are good practices that will ensure longevity!

The truth is I am often amazed at how poorly designed or implemented some systems are, and yet

they do seem to work... for the most part. The occasional lockup or crash is just written off by the user as "that's they way they all are." All except *my* systems, of course. In *my* systems, I adhere to proper design and engineering practices; in fact, I am often guilty of engineering or specifying overkill into things. Although it adds to the cost, they do seem to run better because of it.

In other words, what one individual can "get away" with does not change the laws of physics--this does not change the fact that for those supporting many systems or selling systems where the maximum in reliability and service life is desired, the gold/tin issue *does* matter.

Another issue that was brought to my attention was the thickness of the gold on the contacts; people are afraid that it is so thin, it will wear off after one or two insertions. Certainly, the choice of gold coating thickness depends on the durability required by the application; due to the high cost of gold, it is prudent to keep the gold coating thickness as low as is appropriate for the durability requirements.

An aid to durability is that the gold coatings are usually hardened by adding small amounts of cobalt or nickel to the gold. Such coatings are defined as "hard gold" and produce coatings with a low coefficient of friction and excellent durability characteristics. Hard gold-coated contacts can generally withstand hundreds to thousands of durability cycles without failing. The durability of hard gold coatings can be enhanced by using an underlayer having a hardness value that is greater than that of gold and which will provide mechanical support. Nickel is generally recommended as an underlayer for this purpose. Lubricants are also effective at increasing the durability of gold coatings. Generally, lubrication can increase the durability of a gold contact by an order of magnitude.

Increasing the thickness of a hard gold coating increases durability. The following laboratory results were obtained by AMP for the wear-through of a hard gold coating to the 1.3 micron (50 microinch) thick nickel underplate. The following data is for a 0.635 cm. (0.250 in.) diameter ball wiped a distance of 1.27 cm. (0.500 in.) under a normal force of 100 grams for each cycle.

Thickness Microns	Thickness Micro-in	Cycles to Failure
0.4	15	200
0.8	30	1000
1.3	50	2000

As you can see from this table, an 0.8 micron (30 microinch) coating of hard gold results in durability that is more than adequate for most connector applications, as it would allow 1,000 insertion and removal cycles before wearing through. I studied the specification sheets for DIMM and SIMM sockets produced by AMP and found that their gold-plated sockets come mostly as .000030 (30 microinches)-thick gold over .000050 (50 microinches)-thick nickel in the contact area. As far as I can determine, the coating on virtually all SIMM and DIMM contacts would be similar in thickness, allowing for the same durability.

AMP also has a few gold-plated sockets with specifications of .001020 (1,020 microinches)-thick gold over .001270 (1,270 microinches)-thick nickel. I'd guess that the latter sockets were for devices such as SIMM testers, where many insertions were expected over the life of the equipment. They could also be used in high vibration or very high humidity environments, as well.

For reference, all their tin-contact SIMM and DIMM sockets have the following connector plating specifications: .000030 (30 microinches) minimum thick tin on mating edge over .000050 (50

microinches) minimum thick nickel on entire contact.

The bottom line is that the thickness of the coating in current SIMM and DIMM sockets and modules is not an issue for the expected use of these devices. The thickness of the plating used in current SIMM and DIMM systems is also not relevant with regard to the tin versus gold matability issue. The only drawback to a thinner gold plating is that it will wear after fewer insertion/removal cycles, exposing the nickel underneath and allowing the onset of fretting corrosion to occur.

In my opinion, this tin/gold issue will be even *more important* for those using DIMMs. This will be for two reasons. With SIMMs, you really have two connections for each pin (one on each side of the module), so if one of them goes high resistance, it will not matter; there is a built-in redundancy. With DIMMs, you have many, many more connections (168 versus 72), and no redundant connections. The chance for failure will be much greater. Also, most DIMM applications will be SDRAM, where the timing is down in the 15-10ns range for 66MHz and 100MHz boards, respectively. At these speeds, the slightest additional resistance in the connection *will* cause problems.

Because of these issues, for the future I would specify only motherboards that have gold-plated SDRAM DIMM connectors, and then, of course, use only SDRAMs with gold-plated contacts. I noted that, for example, Micron produces only SDRAM DIMMs with gold-plated contacts, and all the DIMM sockets I've seen so far are gold plated, as well. I would not currently recommend any motherboards using tin-lead SDRAM DIMM connectors, nor would I recommend purchasing tin-lead SDRAM DIMMs.

Memory Reliability

A part of the nature of memory is that it will inevitably fail. These failures are usually classified as two basic types: hard fails and soft errors.

The most well understood are hard fails, in which the chip is working and then, due to some flaw, physical damage, or other event, becomes damaged and experiences a permanent failure. Fixing this type of failure normally requires replacement of some part of the memory hardware, such as the chip, SIMM, or DIMM. Hard error rates are known as HERs.

The other more insidious type of failure is the soft error. A *soft error* is a nonpermanent failure that may never reoccur, or occur at infrequent intervals. (Soft fails are effectively "fixed" by powering the system off and back on.) Soft error rates are known as SERs.

About 20 years ago, Intel made a discovery that shook the memory industry with respect to soft errors. They found that alpha-particles were causing an unacceptably high rate of soft errors or Single Event Upsets (SEUs, as they are sometimes called) in the 16K DRAMs that were available at the time. Because alpha-particles are low-energy particles that can be stopped by something as thin and light as a sheet of paper, it became clear that for alpha-particles to cause a DRAM soft error, they would have to be coming from within the semiconductor material. Testing showed that there were trace elements of thorium and uranium found in the plastic and ceramic chip packaging materials used at the time. This forced all the memory manufacturers to evaluate their manufacturing process to produce materials free from contamination.

Today, the memory manufacturers have all but totally eliminated the alpha-particle source of soft

errors; due to this, many were believing that this was justification for the industry trend to drop parity checking. The argument is that, for example, a 16M memory subsystem built with 4M technology would experience a soft error due to alpha-particles only about once every 16 years! The real problem with this thinking is that it is seriously flawed, and many system manufacturers and vendors were coddled into removing parity and other memory fault-tolerant techniques from their systems even though soft errors continue to be an ongoing problem. There are more recent discoveries that prove that alpha-particles are now only a small fraction of the cause of DRAM soft errors!

As it turns out, the biggest cause of soft errors today are cosmic rays. IBM researchers began investigating the potential of terrestrial cosmic rays in causing soft errors similar to alpha-particles. The difference is that cosmic rays are very high-energy particles and cannot be stopped by sheets of paper or other more powerful types of shielding. The leader in this line of investigation was Dr. J. F. Ziegler of the IBM Watson Research Center in Yorktown Heights, New York. He has produced landmark research into the understanding of cosmic rays and their influence on soft errors in memory.

One example of the magnitude of the cosmic ray soft-error phenomenon demonstrated that with a certain sample of non-IBM DRAMs, the Soft Error Rate (SER)--as measured under real-life conditions, and with the benefit of millions of device hours of testing-- at sea level was measured at 5950 FIT (Failures in Time, which is measured at 1 billion hours) per chip. In an average system with 36 memory chips among the SIMMs, this would result in a soft error occurring every six months! In power user or server systems with a larger amount of memory, this could mean an error per month or more. When the exact same test setup and DRAMs were moved to an underground vault shielded by over 50 feet of rock, thus eliminating all cosmic rays, there were absolutely no soft errors recorded! This not only demonstrates how problematic cosmic rays can be, but it also proves that the packaging contamination and alpha-particle problem has indeed been solved.

Cosmic ray-induced errors are even more of a problem in SRAMs than DRAMS. This is because the amount of charge required to flip a bit in an SRAM cell is less than is required to flip a DRAM cell capacitor. Cosmic rays are also more of a problem for higher-density memory. As chip density increases, it becomes easier for a stray particle to flip a bit. It has been predicted by some that the Soft Error Rate of a 64M DRAM will be double that of a 16M chip, and a 256M DRAM will have a rate four times higher.

Unfortunately, the PC industry has largely failed to recognize this new cause of memory errors. The random and intermittent nature of a soft error can be much more easily explained away by electrostatic discharge, power surges, or unstable software, especially right after a new release of an operating system or major application.

Studies have shown that the soft-error rate for ECC systems is on the order of 30 times greater than the hard-error rate. This is not surprising to those familiar with the full effects of cosmic ray-generated soft errors. Data now exists that suggests some 16M DRAM chip designs will actually experience soft errors numbering in the 24,000 FIT (Failures In Time - per 1 billion hours) range. This would result in a soft error about once a month for most systems today!

How can we deal with these errors? Just ignoring them is certainly not the best way to deal with them, but unfortunately that is what many system manufacturers and vendors are doing today. The best way to deal with this problem is to increase the system's fault tolerance. This means implementing ways of detecting and possibly correcting errors in PC systems. There are basically three levels and techniques for fault tolerance used in modern PCs:

- Non-parity
- Parity
- ECC (Error Correcting Code)

Non-parity systems have no fault tolerance at all. The reason they are even used is because they have the lowest inherent cost. No additional memory is necessary as is the case with parity or ECC techniques. Because a parity-type data byte has nine bits versus eight for non-parity, memory cost is 12.5 percent higher. Also the non-parity memory controller is simplified because it does not need the logic gates to calculate parity or ECC check bits. Portable systems that place a premium on minimizing power might benefit from the reduction in memory power due to fewer DRAM chips. Finally, the memory system data bus is narrower, which reduces the amount of data buffers. The statistical probability of memory failures in a modern office desktop computer is now estimated at about one error every few months, more or less depending on how much memory you have.

This error rate may be tolerable for low-end systems that are not used for mission-critical applications. In this case, their extreme market cost sensitivity probably can't justify the extra cost of parity or ECC memory, and such errors will then have to be tolerated.

At any rate, employing no fault tolerance in a system is simply gambling that memory errors are unlikely, and if they do occur, result in an inherent cost less than the additional hardware necessary for error detection. However, the disadvantage is that the errors can lead to a serious problem such as calculating the wrong value to go into a bank check, or in the case of a system being used as a server, a memory error forcing a system hang and bringing down all LAN-resident client systems with subsequent loss of productivity. Finally, with a non-parity or non-ECC memory system, problem traceability is difficult, which is not the case with parity or ECC. These techniques at least isolate to a memory source as the culprit, thus reducing both the time and cost of problem resolutions.

Parity Checking. One standard IBM set for the industry is that the memory chips in a bank of nine each handle one bit of data: eight bits per character plus one extra bit called the parity bit. The parity bit enables memory-control circuitry to keep tabs on the other eight bits--a built-in cross-check for the integrity of each byte in the system. If the circuitry detects an error, the computer stops and displays a message informing you of the malfunction. If you are running a newer operating system such as Windows or OS/2, a parity error will generally manifest itself as a locked system. When you reboot, the BIOS should detect the error and display the appropriate error message.

SIMMs and DIMMs are available both with and without parity bits. Originally, all PC systems used parity-checked memory to ensure accuracy. Starting in 1994, a disturbing trend developed in the PC-compatible marketplace. Most vendors began shipping systems without parity checking or any other means of detecting or correcting errors! These systems can use cheaper non-parity SIMMs, which saves about 10-15 percent on memory costs for a system. Parity memory results in increased initial system cost due primarily to the additional memory bits involved. Parity cannot correct system errors, but, because parity can detect errors, it can make the user aware of memory errors when they occur. This has two basic benefits:

- Parity guards against the consequences of faulty calculations based on incorrect data.

- Parity pinpoints the source of errors, which assists in problem resolution, thus improving system serviceability.

PC systems can easily be designed to function using either parity or non-parity memory. The cost of implementing parity as an option on a motherboard is virtually nothing; the only cost involved is in actually purchasing the parity SIMMs or DIMMs, which are about 10-15 percent more expensive than non-parity versions. This would enable a system manufacturer to offer their system purchasers the choice of parity if they feel the additional cost is justified for their particular application.

Unfortunately, several of the big names began selling systems without parity to reduce the price of their systems, and they did not make it well-known that the lower cost was due to parity memory no longer being included as standard. This began happening mostly in 1994-1995, and has continued until very recently with few people understanding the full implications. After one or two major vendors did this, most of the others were forced to follow to remain price-competitive.

Because nobody wanted to announce this information, it remained as a sort of dirty little secret within the industry. Originally, when this happened, you could still specify parity memory when you ordered a system, even though the default configurations no longer included it. There would be a 10-15 percent surcharge on the memory, but those who wanted reliable, trustworthy systems could at least get them, provided they knew to ask, of course. Then a major bomb hit the industry, in the form of the Intel Triton 430FX Pentium chipset, which was the first major chipset on the market that did not support parity checking at all! This also became the most popular chipset of its time, and was found in virtually all Pentium motherboards sold in the 1995 timeframe. This set a disturbing trend for the next few years. All but one of Intel's Pentium processor chipsets after the 430FX also did not support parity-checked memory; the only one that did was the 430HX Triton II. The good news is that this trend now seems to be over, and the system manufacturers and users are finally coming around to the importance of fault-tolerant memory. As such, all the Pentium Pro and Pentium II chipsets currently available do support parity and ECC, and more and more people are realizing that this type of memory is worthwhile. I do expect new low-end chipsets even for the Pentium II to emerge that will not support parity, but these are designed for the sub-\$1,000 market, where price and not system integrity is the most important concern.

Let's look at how parity checking works, and then examine in more detail the successor to parity checking, called ECC (Error Correcting Code), which can not only detect but correct memory errors on-the-fly.

IBM originally established the *odd parity* standard for error checking. The following explanation may help you understand what is meant by odd parity. As the eight individual bits in a byte are stored in memory, a parity generator/checker, which is either part of the CPU or located in a special chip on the motherboard, evaluates the data bits by counting the number of 1s in the byte. If an even number of 1s is in the byte, the parity generator/checker creates a 1 and stores it as the ninth bit (parity bit) in the parity memory chip. That makes the total sum for all nine bits an odd number. If the original sum of the eight data bits is an odd number, the parity bit created is 0, keeping the 9-bit sum an odd number. The value of the parity bit is always chosen so that the sum of all nine bits (eight data bits plus one parity bit) is an odd number. Remember that the eight data bits in a byte are numbered 0 1 2 3 4 5 6 7. The following examples may make it easier to understand:

Data bit number:	0	1	2	3	4	5	6	7	Parity bit
Data bit value:	1	0	1	1	0	0	1	1	0

In this example, because the total number of data bits with a value of 1 is an odd number (5), the parity bit must have a value of 0 to ensure an odd sum for all nine bits.

The following is another example:

```
Data bit number: 0 1 2 3 4 5 6 7 Parity bit
Data bit value:  0 0 1 1 0 0 1 1    1
```

In this example, because the total number of data bits with a value of 1 is an even number (4), the parity bit must have a value of 1 to create an odd sum for all nine bits.

When the system reads memory back from storage, it checks the parity information. If a (9-bit) byte has an even number of bits with a parity bit value of 1, that byte must have an error. The system cannot tell which bit has changed, or if only a single bit has changed. If three bits changed, for example, the byte still flags a parity-check error; if two bits changed, however, the bad byte may pass unnoticed. The following examples show parity-check messages for three types of systems:

```
For the IBM PC:                PARITY CHECK x
For the IBM XT:                PARITY CHECK x    YYYYYY (z)
For the IBM AT and late model XT: PARITY CHECK x    YYYYYY
```

Where *x* is 1 or 2:

1 = Error occurred on the motherboard

2 = Error occurred in an expansion slot

YYYYYY represents a number from 00000 through FFFFFF that indicates, in hexadecimal notation, the byte in which the error has occurred.

Where (*z*) is (S) or (E):

(S) = Parity error occurred in the system unit

(E) = Parity error occurred in the expansion chassis

NOTE

An expansion chassis was an option IBM sold for the original PC and XT systems to add more expansion slots. This unit consisted of a backplane motherboard with eight slots, one of which contained a special extender/receiver card cabled to a similar extender/receiver card placed in the main system. Due to the extender/receiver cards in the main system and the expansion chassis, the net gain was six slots.

When a parity-check error is detected, the motherboard parity-checking circuits generate a *non-maskable interrupt* (NMI), which halts processing and diverts the system's attention to the error. The NMI causes a routine in the ROM to be executed. The routine clears the screen and then displays a message in the upper-left corner of the screen. The message differs depending on the type of

computer system. On some older IBM systems, the ROM parity-check routine halts the CPU. In such a case, the system locks up, and you must perform a hardware reset or a power-off/power-on cycle to restart the system. Unfortunately, all unsaved work is lost in the process.

Most systems do not halt the CPU when a parity error is detected; instead, they offer you a choice of either rebooting the system or continuing as though nothing happened. Additionally, these systems may display the parity error message in a different format from IBM, although the information presented is basically the same. For example, many systems with a Phoenix BIOS display these messages:

```
Memory parity interrupt at xxxx:xxxx  
Type (S)hut off NMI, Type (R)eboot, other keys to continue
```

or

```
I/O card parity interrupt at xxxx:xxxx  
Type (S)hut off NMI, Type (R)eboot, other keys to continue
```

The first of these two messages indicates a motherboard parity error (Parity Check 1), and the second indicates an expansion-slot parity error (Parity Check 2). Notice that the address given in the form `xxxx:xxxx` for the memory error is in a segment:offset form rather than a straight linear address such as with IBM's error messages. The segment:offset address form still gives you the location of the error to a resolution of a single byte.

You have three ways to proceed after viewing this error message.

- You can press S, which shuts off parity checking and resumes system operation at the point where the parity check first occurred.
- Pressing R forces the system to reboot, losing any unsaved work.
- Pressing any other key causes the system to resume operation with parity checking still enabled.

If the problem occurs, it is likely to cause another parity-check interruption. In most cases, it is most prudent to press S, which disables the parity checking so that you can then save your work. It would be best in this case to save your work to a floppy disk to prevent the possible corruption of a hard disk. You should also avoid overwriting any previous (still good) versions of whatever file you are saving, because in fact you may be saving a bad file due to the memory corruption. Because parity checking is now disabled, your save operations will not be interrupted. Then you should power the system off, restart it, and run whatever memory diagnostics software you have to try and track down the error. In some cases, the POST finds the error on the next restart, but, in most cases, you need to run a more sophisticated diagnostics program, perhaps in a continuous mode, to locate the error.

The AMI BIOS displays the parity error messages in the following forms:

```
ON BOARD PARITY ERROR ADDR (HEX) = (xxxxxx)
```

or

```
OFF BOARD PARITY ERROR ADDR (HEX) = (xxxxxx)
```

These messages indicate that an error in memory has occurred during the POST, and the failure is located at the address indicated. The first one indicates the error occurred on the motherboard, whereas the second message indicates an error in an expansion slot adapter card. The AMI BIOS also can display memory errors in the following manner:

```
Memory Parity Error at xxxxxx
```

or

```
I/O Card Parity Error at xxxxxx
```

These messages indicate that an error in memory has occurred at the indicated address during normal operation. The first one indicates a motherboard memory error, and the second indicates an expansion slot adapter memory error.

Although many systems enable you to continue processing after a parity error, and even allow for the disabling of further parity checking, continuing to use your system after a parity error is detected can be dangerous. The idea behind letting you continue using either method is to give you time to save any unsaved work before you diagnose and service the computer, but be careful how you do this.

CAUTION

When you are notified of a memory parity error, remember the parity check is telling you that memory has been corrupted. Do you want to save potentially corrupted data over the good file from the last time you saved? Definitely not! Make sure that you save your work to a different file name. In addition, after a parity error, save only to a floppy disk if possible and avoid writing to the hard disk; there is a slight chance that the hard drive could become corrupted if you save the contents of corrupted memory.

After saving your work, determine the cause of the parity error and repair the system. You may be tempted to use an option to shut off further parity checking and simply continue using the system as if nothing were wrong. Doing so resembles unscrewing the oil pressure warning indicator bulb on a car with an oil leak so that the oil pressure light won't bother you anymore!

ECC (Error Correcting Code). ECC goes a big step beyond simple parity error detection. Rather than just detecting an error, ECC allows a single bit error to be corrected, which means the system can continue on without interruption and without corrupting data. ECC as implemented in most PCs can only detect and not correct double-bit errors. Because studies have indicated that approximately 98 percent of memory errors are single-bit variety, the most commonly used type of ECC is one in which the attendant memory controller detects and corrects single-bit errors in an accessed data word (double-bit errors can be detected, but not corrected). This type of ECC is known as SEC-DED and requires an additional seven check bits over 32 bits in a 4-byte system and eight check bits in an 8-byte system. ECC in a 4-byte system obviously costs more than non-parity or parity, but in an 8-byte system, ECC and parity costs are equal.

ECC entails the memory controller calculating the check bits on a memory-write operation,

performing a compare between the read and calculated check-bits on a read operation and, if necessary, correcting bad bit(s). The additional ECC logic in the memory controller is not very significant in this age of inexpensive, high-performance VLSI logic, but ECC actually affects memory performance on writes. This is because the operation must be timed to wait for the calculation of check bits and, when the system waits for corrected data, reads. On a partial-word write, the entire word must first be read, the affected byte(s) rewritten, and then new check bits calculated. This turns partial-word write operations into slower read-modify writes.

Most memory errors are of a single-bit nature, which are correctable by ECC. Incorporating this fault-tolerant technique provides high system reliability and attendant availability. An ECC-based system is a good choice for servers, workstations, or mission-critical applications in which the cost of a potential memory error outweighs the additional memory and system cost to correct it, along with ensuring that it does not detract from system reliability. What I'm saying is that if you value your data and use your system for important (to you) tasks, then you'll want ECC memory.

By designing a system that allows the user to make the choice of ECC, parity, or non-parity, the user can choose the level of fault tolerance desired, as well as how much they want to gamble with their data.

Installing Memory Upgrades

Adding memory to a system is one of the most useful upgrades that you can perform, and also one of the least expensive, especially when you consider the increased capabilities of Windows 95/98, Windows NT, and OS/2 when you give them access to more memory. In some cases, doubling the memory can virtually double the speed of a computer.

This section discusses adding memory, including selecting memory chips, installing memory chips, and testing the installation.

Upgrade Options and Strategies

Adding memory can be an inexpensive solution; at this writing, the cost of memory has fallen to about \$1.50 per megabyte. A small dose can give your computer's performance a big boost.

How do you add memory to your PC? You have three options, listed in order of convenience and cost:

- Adding memory in vacant slots on your motherboard.
- Replacing your current motherboard's memory with higher-capacity memory.
- Purchasing a memory expansion card (not a cost-effective or performance-effective solution for any system currently on the market).

Adding expanded memory to PC- or XT-type systems is not a good idea, mainly because an expanded memory board with a couple of megabytes of expanded memory installed can cost more than the entire system is worth. Also, this memory does not function for Windows, and a PC- or XT-class system cannot run OS/2. Instead, purchase a more powerful system--for example, an inexpensive

Pentium II 266--with greater expansion capabilities.

If you decide to upgrade to a more powerful computer system, you normally cannot salvage the memory from a PC or XT system. The 8-bit memory boards are useless in 16-bit systems, and the speed of the memory chips usually is inadequate for newer systems. All new systems use high-speed SIMM or DIMM modules rather than chips. A pile of 150ns (nanoseconds), 64K, or 256K chips is useless if your next system is a high-speed system that uses SIMMs or memory devices faster than 70ns.

Be sure to weigh carefully your future needs for computing speed and for a multitasking operating system (OS/2, Windows 95/98, Windows NT, or Linux, for example) with the amount of money that you spend to upgrade current equipment.

Before you add RAM to a system (or replace defective RAM chips), you must determine the memory chips required for your system. Your system documentation contains this information.

If you need to replace a defective SIMM or DIMM and do not have the system documentation, you can determine the correct module for your system by inspecting the ones that are already installed. Each module has markings that indicate the module's capacity and speed. RAM capacity and speed are both discussed in detail earlier in this chapter.

If you do not have the documentation for your system and the manufacturer does not offer technical support, open your system case and carefully write down the markings that appear on your memory chips. Then, contact a local computer store or module vendor such as Kingston, Micron (Crucial), PNY, or others for help in determining the proper RAM chips for your system. Adding the wrong modules to a system can make it as unreliable as leaving a defective module installed and trying to use the system in that condition.

NOTE

Before upgrading a system beyond 64M of RAM, be sure that your chipset supports caching of more than 64M. Adding RAM beyond the amount that your system can cache will slow performance rather than increase it. See the section "Cache Memory--SRAM" earlier in this chapter and the discussion of chipsets in Chapter 4 for a more complete explanation of this common system limitation.

Selecting and Installing Motherboard Memory with Chips, SIMMs, or DIMMs

If you are upgrading a motherboard by adding memory, follow the manufacturer's guidelines on which memory chips or modules to purchase. As you learned earlier, memory comes in various form factors, including individual chips known as DIP memory chips, SIMMs (single inline memory modules), and DIMMs. Your computer may use one or possibly a mixture of these form factors.

No matter what type of memory chips you have, the chips are installed in memory banks. A *memory bank* is a collection of memory chips that make up a block of memory. Each bank of memory is read by your processor in one pass. A memory bank does not work unless it is filled with memory chips. (Banks are discussed in more detail earlier in this chapter in the section "Memory Banks.")

Pentium, Pentium Pro, and Pentium II computers also normally have between two and four banks of

memory, but each bank usually requires two 72-pin (32- or 36-bit) SIMMs or one 168-pin DIMM.

Installing extra memory on your motherboard is an easy way to add memory to your computer. Most systems have at least one vacant memory bank in which you can install extra memory at a later time and speed your computer.

Replacing SIMMS and DIMMs with Higher Capacity

If all the SIMM or DIMM slots on your motherboard are occupied, your best option is to remove an existing bank of memory and replace it with higher memory. For example, if you have a 16M of RAM in a Pentium system with four SIMM slots filled with 4M 72-pin SIMMs, you may be able to replace all four SIMMs with 8M SIMMs and increase the system memory to 32M. Alternately, you could keep two of the existing 4M SIMMs and replace two with 8M SIMMs, increasing the total RAM to 24M. Remember that just as when you install SIMMs, they need to be replaced in a full bank and all the SIMMs in the bank need to be the same capacity. The multiple module requirement is not an issue with DIMMs as in all current systems; a DIMM constitutes at least one full bank of memory.

However, just because there are higher-capacity SIMMs or DIMMs available that are the correct pin count to plug into your motherboard, don't automatically assume the higher-capacity memory will work. The chipset and BIOS of your system will set limits on the capacity of the memory you can use. Check your system or motherboard documentation to see what size SIMMs or DIMMs will work with it before purchasing the new RAM.

Adding Adapter Boards

Memory expansion boards typically are a last-resort way to add memory. For many systems (such as older models from Compaq) with proprietary local bus memory-expansion connectors, you must purchase all memory-expansion boards from that company. Similarly, IBM used proprietary memory connectors in many of the PS/2 systems. For other industry-standard systems that use nonproprietary memory expansion, you can purchase from hundreds of vendors memory-expansion boards that plug into the standard bus slots.

Unfortunately, any memory expansion that plugs into a standard bus slot runs at bus speed rather than at full-system speed. For this reason, most systems today provide standard SIMM or DIMM connector sockets directly on the motherboard so that the memory can be plugged directly into the system's local bus. Using memory adapter cards in these systems only slows them down. Other systems use proprietary local bus connectors for memory-expansion adapters, which can cause additional problems and expense when you have to add or service memory. Upgrading a system by using a memory adapter card is no longer cost-effective, either. Because most systems and motherboards built on 486 or later processors can be upgraded with SIMMs or DIMMs, the only systems that would be upgraded with an adapter card are so old, it would be less expensive to replace the entire motherboard than to upgrade these old systems.

Installing Memory

This section discusses installing memory --specifically, new SIMM or DIMM modules. The section also covers the problems that you are most likely to encounter and how to avoid them. You will also get information on configuring your system to use new memory.

When you install or remove memory, you are most likely to encounter the following problems:

- Electrostatic discharge
- Broken or bent pins
- Incorrectly seated SIMMs and DIMMs
- Incorrect switch and jumper settings

To prevent electrostatic discharge (ESD) when you install sensitive memory chips or boards, do not wear synthetic-fiber clothing or leather-soled shoes. Remove any static charge that you are carrying by touching the system chassis before you begin, or better yet, wear a good commercial grounding strap on your wrist. You can order one from an electronics parts store or mail-order house. A grounding strap consists of a conductive wristband grounded at the other end by a wire clipped to the system chassis. Leave the system unit plugged in--but turned off--to keep it grounded.

CAUTION

Be sure to use a properly designed commercial grounding strap; do not make one yourself. Commercial units have a one-megohm resistor that serves as protection if you accidentally touch live power. The resistor ensures that you do not become the path of least resistance to the ground and therefore become electrocuted. An improperly designed strap can cause the power to conduct through you to the ground, possibly killing you.

Broken or bent leads are another potential problem associated with installing individual memory chips (DIPs) or SIPP modules. Fortunately, this is not a problem you will encounter installing a SIMM or DIMM. Sometimes, the pins on new chips are bent into a V, making them difficult to align with the socket holes. If you notice this problem on a DIP chip, place the chip on its side on a table, and press gently to bend the pins so that they are at a 90-degree angle to the chip. For a SIPP module, you may want to use needle-nose pliers to carefully straighten the pins so that they protrude directly down from the edge of the module, with equal amounts of space between pins. Then, you should install the chips in the sockets one at a time.

CAUTION

Straightening the pins on a DIP chip or SIPP module is not difficult work, but if you are not careful, you could easily break off one of the pins, rendering the chip or memory module useless. Use great care when you straighten the bent pins on any memory chip or module. You can use chip-insertion and pin-straightening devices to ensure that the pins are straight and aligned with the socket holes; these inexpensive tools can save you a great deal of time.

Each memory chip or module must be installed to point in a certain direction. Each device has a polarity marking on one end. This is normally a notch on SIMMs or DIMMs, and may be a notch, circular indentation, or mark on any individual chip. The chip socket may have a corresponding notch. Otherwise, the motherboard may have a printed legend that indicates the orientation of the chip. If the socket is not marked, you should use other chips as a guide. The orientation of the notch

indicates the location of Pin 1 on the device. Aligning this notch correctly with the others on the board ensures that you do not install the chip backward. Gently set each chip into a socket, ensuring that every pin is properly aligned with the connector into which it fits. Then push the chip in firmly with both thumbs until the chip is fully seated.

SIMM memory is oriented by a notch on one side of the module that is not present on the other side, as shown in Figure 5.9. The socket has a protrusion that must fit into this notched area on one side of the module. This protrusion makes it impossible to install a SIMM backward unless you break the connector. Figure 5.10 shows a detailed blowup of the backside of the SIMM, showing the notch and the locking clip. Figure 5.11 shows a closeup of a DIMM (note that it, too, is keyed to ensure proper expansion socket installation).

FIG. 5.10 *The notch on this SIMM is shown on the left end. Insert the SIMM at an angle and then tilt it forward until the locking clips snap into place.*

Similarly, DIMMs are keyed by notches along the bottom connector edge that are offset from center so that they can be inserted in only one direction, as shown in Figure 5.12. SIPP modules, however, do not plug into a keyed socket; you have to orient them properly. The system documentation can be helpful if the motherboard has no marks to guide you. You also can use existing SIPP modules as a guide.

The DIMM ejector tab locks into place in the notch on the side of the DIMM when it is fully inserted. Some DIMM sockets have ejector tabs on both ends. When installing SIMMs and DIMMs, take care not to force the module into the socket. If the module does not slip easily into the slot and then snap into place, there is a good chance the module is not oriented or aligned correctly. Forcing the module could break the module or the socket. If the retaining clips on the socket break, the memory will not be held firmly in place and there is a good chance you will experience random memory errors because the module will not make consistent electrical contact if it is loose.

FIG. 5.11 *This closeup shows the backside of a SIMM inserted in the SIMM socket with the notch aligned, the locking clip locked, and the hole in the SIMM aligned with the tab that sticks out from the socket.*

FIG. 5.12 *DIMMs keys match the protrusions in the DIMM sockets.*

As explained earlier in this chapter, DIMMs can come in several different varieties, including unbuffered or buffered and 3.3v or 5v. Buffered DIMMs have additional buffer chips on them to interface to the motherboard. Unfortunately, these buffer chips slow the DIMM down and are not effective at higher speeds. For this reason, all PC systems use unbuffered DIMMs. The voltage is simple--DIMM designs for PCs are almost universally 3.3v. If you install a 5v DIMM in a 3.3v socket, it would be damaged, but fortunately keying in the socket and on the DIMM will prevent that.

Modern PC systems only use unbuffered 3.3v DIMMs. Apple and other non-PC systems may use the buffered 5v versions. Fortunately, the key notches along the connector edge of a DIMM are spaced differently for RFU and buffered DIMMs and 5.0v DIMMs, as shown in Figure 5.13. This prevents inserting a DIMM of the wrong type into a socket.

FIG. 5.13 *168-pin DRAM DIMM notch key definitions.*

Before installing memory, make sure that the system power is off. Then remove the PC cover and any installed cards. SIMMs and DIMMs snap easily into place, but chips can be more difficult to install. A chip-installation tool is not required, but it can make inserting the chips into sockets much easier. To remove chips, use a chip extractor or small screwdriver. Never try removing a RAM chip with your fingers, because you can bend the chip's pins or poke a hole in your finger with one of the pins. You remove SIMMs and DIMMs by releasing the locking tabs and either pulling or rolling them out of their sockets.

After adding the memory and putting the system back together, you may have to run the CMOS setup and resave with the new amount of memory. Most newer systems automatically detect the new amount of memory and reconfigure this for you. Most newer systems also don't require setting any jumpers or switches on the motherboard to configure them for your new memory. Older motherboards that used individual RAM chips rather than SIMMs or DIMMs for memory are more likely to need changes to jumper and switch settings. If you install individual RAM chips on a motherboard and the CMOS does not recognize the new RAM, check the system documentation to see if changes to motherboard jumpers or switches are required.

NOTE

Information on installing memory on older memory-expansion cards can be found in Chapter 7 of the sixth Edition of Upgrading and Repairing PCs on the CD-ROM accompanying this book.

After configuring your system to work properly with the additional memory, you should run a memory-diagnostics program to ensure the proper operation of the new memory. At least two and sometimes three memory-diagnostic programs are available for all systems. In order of accuracy, these programs are:

- POST (Power-On Self Test)
- Disk-based advanced diagnostics software

The POST is used every time you power up the system.

Many additional diagnostics programs are available from aftermarket utility software companies. More information on aftermarket testing facilities can be found in Chapter 17, "Diagnostics, Testing, and Maintenance."

The System Logical Memory Layout

The original PC had a total of 1M of addressable memory, and the top 384K of that was reserved for use by the system. Placing this reserved space at the top (between 640K and 1024K instead of at the bottom, between 0K and 640K) led to what today is often called the *conventional memory barrier*. The constant pressures on system and peripheral manufacturers to maintain compatibility by never breaking from the original memory scheme of the first PC has resulted in a system memory structure that is (to put it kindly) a mess. Almost two decades after the first PC was introduced, even the newest Pentium II-based systems are limited in many important ways by the memory map of the first PCs.

Someone who wants to become knowledgeable about personal computers must at one time or another come to terms with the types of memory installed on his system--the small and large pieces of different kinds of memory, some accessible by software application programs, and some not. The following sections detail the different kinds of memory installed on a modern PC. The kinds of memory covered in the following sections include the following:

- Conventional (Base) memory
- Upper Memory Area (UMA)
- High Memory Area (HMA)
- Extended memory (XMS)
- Expanded memory (obsolete)
- Video RAM memory (part of UMA)
- Adapter ROM and Special Purpose RAM (part of UMA)
- Motherboard ROM BIOS (part of UMA)

Subsequent sections also cover preventing memory conflicts and overlap, using memory managers to optimize your system's memory, and making better use of memory. In a 16-bit or higher system, the memory map extends beyond the 1M boundary and can continue to 16M on a system based on the 286 or higher processor, 4G (4,096M) on a 386DX or higher, or as much as 64G (65,536M) on a Pentium II. Any memory past 1M is called *extended memory*.

Figure 5.14 shows the logical address locations for a 16-bit or higher system. If the processor is running in real mode, only the first megabyte is accessible. If the processor is in protected mode, the full 16M, 4,096M, or 65,536M are accessible. Each symbol is equal to 1K of memory, and each line or segment is 64K. This map shows the first two megabytes of system memory.

See "Processor Modes," p. 43

NOTE

To save space, this map is ended after the end of the second megabyte. In reality, this map continues to the maximum of addressable memory.

Conventional (Base) Memory

The original PC/XT-type system was designed to use 1M of memory workspace, sometimes called *RAM (random access memory)*. This 1M of RAM is divided into several sections, some of which have special uses. DOS can read and write to the entire megabyte, but can manage the loading of programs only in the portion of RAM space called *conventional memory*, which was 512K at the time the first PC was introduced. The other 512K was reserved for use by the system, including the motherboard and adapter boards plugged into the system slots.

After introducing the system, IBM decided that only 384K was needed for these reserved uses, and the company began marketing PCs with 640K of user memory. Thus, 640K became the standard for memory that can be used by DOS for running programs, and is often termed the *640K memory barrier*. The remaining memory after 640K was reserved for use by the graphics boards, other adapters, and the motherboard ROM BIOS.

This barrier largely affects 16-bit software such as DOS and Windows 3.1, and is much less of a factor with 32-bit software and operating systems such as Windows 95/98, NT, and so on.

Upper Memory Area (UMA)

The term *Upper Memory Area (UMA)* describes the reserved 384K at the top of the first megabyte of system memory on a PC/XT and the first megabyte on an AT-type system. This memory has the addresses from A0000 through FFFFF. The way the 384K of upper memory is used breaks down as follows:

- The first 128K after conventional memory is called *video RAM*. It is reserved for use by video adapters. When text and graphics are displayed onscreen, the electronic impulses that contain their images reside in this space. Video RAM is allotted the address range from A0000-BFFFF.

FIG. 5.14 *The logical memory map of the first 2M.*

- The next 128K is reserved for the adapter BIOS that resides in read-only memory chips on some adapter boards plugged into the bus slots. Most VGA-compatible video adapters use the first 32K of this area for their onboard BIOS. The rest can be used by any other adapters installed. Many network adapters also use this area for special-purpose RAM called Shared Memory. Adapter ROM and special-purpose RAM is allotted the address range from C0000-DFFFF.
- The last 128K of memory is reserved for motherboard BIOS (the basic input/output system, which is stored in read-only RAM chips or ROM). The POST (Power-On Self Test) and bootstrap loader, which handles your system at bootup until the operating system takes over, also reside in this space. Most systems only use the last 64K (or less) of this space, leaving the first 64K or more free for remapping with memory managers. Some systems also include the CMOS Setup program in this area. The motherboard BIOS is allotted the address range from E0000-FFFFFF.

Not all the 384K of reserved memory is fully used on most 16-bit and higher systems. For example, according to the PC standard, reserved video RAM begins at address A0000, which is right at the 640K boundary. Normally, this is used for VGA graphics modes, while the monochrome and color text modes use B0000-B7FFF and B8000-BFFFF, respectively. Older non-VGA adapters only used memory in the B0000 segment. Different video adapters use varying amounts of RAM for their operations, depending mainly on the mode they are in. To the processor, however, it always appears as the same 128K area no matter how much RAM is really on the video card. This is managed by bank switching areas of memory on the card in and out of the A0000-BFFFF segments.

Although the top 384K of the first megabyte was originally termed *reserved memory*, it is possible to use previously unused regions of this memory to load 16-bit device drivers (such as ANSI.SYS) and

memory-resident programs (such as MOUSE.COM), which frees up the conventional memory they would otherwise require. Note that 32-bit device drivers such as those used with Windows 95/98, NT, and so forth are not affected by this as they load into extended memory. The amount of free UMA space varies from system to system, depending on the adapter cards installed on the system. For example, most SCSI adapters and network adapters require some of this area for built-in ROMs or special-purpose RAM use.

Segment Addresses and Linear Addresses. One thing that can be confusing is the difference between a segment address and a full linear address. The use of segmented address numbers comes from the internal structure of the Intel processors, and is used primarily by older, 16-bit operating systems. They use a separate register for the segment information and another for the offset. The concept is very simple. For example, assume that I am staying in a hotel room, and somebody asks for my room number. The hotel has 10 floors, numbered from zero through nine; each floor has 100 rooms, numbered from 00 to 99. A segment is defined as any group of 100 rooms starting at a multiple of 10, and indicated by a two-digit number. So, a segment address of 54 would indicate the actual room 540, and you could have an offset of 00 to 99 rooms from there.

Thus in this hotel example, each segment is specified as a two-digit number from 00 to 99, and an offset can be specified from any segment starting with a number from 00 to 99 as well.

As an example, let's say I am staying in room 541. If the person needs this information in segment:offset form, and each number is two digits, I could say that I am staying at a room segment starting address of 54 (room 540), and an offset of 01 from the start of that segment. I could also say that I am in room segment 50 (room 500), and an offset of 41. You could even come up with other answers, such as I am at segment 45 (room 450) offset 91 ($450+91=541$). Here is an example of how this adds up:

Segment	Offset	Total
54	01	541
50	41	541
45	91	541

As you can see, although the particular segment and offset are different, they all add up to the same room address. In the Intel x86 processors, a similar scheme is used where a segment and offset are added internally to produce the actual address. It can be somewhat confusing, especially if you are writing assembly language or machine language software!

This is exactly how segmented memory in an Intel processor works. Notice that the segment and offset numbers essentially overlap on all digits except the first and last. By adding them together with the proper alignment, you can see the linear address total.

With 32-bit operating systems, segment addresses are not an issue. A linear address is one without segment:offset boundaries, such as saying room 541. It is a single number and not comprised of two numbers added together. For example, a SCSI host adapter might have 16K ROM on the card addressed from D4000 to D7FFF. These numbers expressed in segment:offset form are D400:0000 to D700:0FFF. The segment portion is composed of the most significant four digits, and the offset portion is composed of the least significant four digits. Because each portion overlaps by one digit,

the ending address of its ROM can be expressed in four different ways, as follows:

```

D000:7FFF =  D000  segment
            + 7FFF  offset
            -----
            = D7FFF  total
D700:0FFF =  D700  segment
            + 0FFF  offset
            -----
            = D7FFF  total
D7F0:00FF =  D7F0  segment
            + 00FF  offset
            -----
            = D7FFF  total
D7FF:000F =  D7FF  segment
            + 000F  offset
            -----
            = D7FFF  total

```

As you can see in each case, although the segment and offset differ slightly, the total ends up being the same. Adding together the segment and offset numbers makes possible even more combinations, as in the following examples:

```

D500:2FFF =  D500  segment
            + 2FFF  offset
            -----
            = D7FFF  total
D6EE:111F =  D6EE  segment
            + 111F  offset
            -----
            = D7FFF  total

```

As you can see, several combinations are possible. The correct and generally accepted way to write this address as a linear address is D7FFF, whereas most would write the segment:offset address as D000:7FFF. Keeping the segment mostly zeros makes the segment:offset relationship easier to understand and the number easier to comprehend. If you understand the segment:offset relationship to the linear address, you now know why when a linear address number is discussed it is five digits, whereas a segment number is only four.

Video RAM Memory. A video adapter installed in your system uses a portion of your system's low memory to hold graphics or character information for display, but normally only when in basic VGA mode. A video card may have 4M or 8M on board, but most of that is used by the video chipset on the card and not directly accessed by your processor. When in basic VGA mode, such as when at a DOS prompt or when running in Windows safe mode, your processor can directly access up to 128K of the video RAM from address A0000-BFFFFh. All modern video cards also have onboard BIOS normally addressed at C0000-C7FFFh, which is part of the memory space reserved for adapter card BIOS. Generally, the higher the resolution and color capabilities of the video adapter, the more system memory the video adapter uses, but again that additional memory (past 128K) is not normally accessible by the processor. Instead, the system tells the video chip what should be displayed, and the video chip generates the picture by putting data directly into the video RAM on the card.

In the standard system-memory map, a total of 128K is reserved for use by the video card to store currently displayed information when in basic VGA mode. The reserved video memory is located in segments A000 and B000. The video adapter ROM uses additional upper memory space in segment

C000. Even with the new multiple monitor feature in Windows 98, only one video card (the primary video card) is in the memory map; all others use no low system memory.

The location of video adapter RAM is responsible for the 640K DOS *conventional memory barrier*. DOS can use all available contiguous memory in the first megabyte of memory until the video adapter RAM is encountered. The use of ancient video adapters such as the MDA and CGA allows DOS access to more than 640K of system memory. The video memory *wall* begins at A0000 for the EGA, MCGA, and VGA systems, but the MDA and CGA do not use as much video RAM, which leaves some space that can be used by DOS and programs. The previous segment and offset examples show that the MDA adapter enables DOS to use an additional 64K of memory (all of segment A000), bringing the total for DOS program space to 704K. Similarly, the CGA enables a total of 736K of possible contiguous memory. The EGA, VGA, or MCGA is limited to the normal maximum of 640K of contiguous memory because of the larger amount used by video RAM. The maximum DOS-program memory workspace, therefore, depends on which video adapter is installed. Table 5.11 shows the maximum amount of memory available to DOS using the referenced video card.

Table 5.11 DOS Memory Limitations Based on Video Adapter Type

Video Adapter Type	Maximum DOS Memory
Monochrome Display Adapter (MDA)	704K
Color Graphics Adapter (CGA)	736K
Enhanced Graphics Adapter (EGA)	640K
Video Graphics Array (VGA)	640K
Super VGA (SVGA)	640K
eXtended Graphics Array (XGA)	640K

Using this memory to 736K might be possible depending on the video adapter, the types of memory boards installed, ROM programs on the motherboard, and the type of system. You can use some of this memory if your system has a 386 or higher processor. With memory manager software (such as EMM386 that comes with DOS and Windows), which can operate the Memory Management Unit (MMU) found in those processors, you can remap extended memory into this space. Remember that Windows 9x will automatically use this memory if available.

The following sections examine how standard video adapters use the system's memory. This map is important because it may be possible to recognize some of this as unused in some systems, which may free up more space for software drivers to be loaded.

Obsolete Video Adapter Types. The MDA, CGA, and EGA video adapter types are not used in any current systems. The following sections give a brief description of the memory usage of these adapters so that you can see the progression of memory usage as adapters have evolved. If you still service or use systems with these video adapters, please see the Sixth edition of the book on the CD-ROM for the complete memory maps for this obsolete hardware.

Monochrome Display Adapter Memory (MDA). The original Monochrome Display Adapter (MDA) uses only a 4K portion of the reserved video RAM from B0000-B0FFF. Because the ROM code used to operate this adapter is actually a portion of the motherboard ROM, no additional ROM space is used in segment C000.

Note that although the original Monochrome Display Adapter only used 4K of memory starting at B0000, a VGA adapter running in Monochrome emulation mode (Mono Text Mode) activates 32K of RAM at this address. A true Monochrome Display Adapter has no onboard BIOS, and instead is operated by driver programs found in the primary motherboard BIOS.

Color Graphics Adapter (CGA) Memory. The Color Graphics Adapter (CGA) uses a 16K portion of the reserved video RAM from B8000-BBFFF. Because the ROM code used to operate this adapter is a portion of the motherboard ROM, no additional ROM space is used in segment C000.

The CGA card leaves memory from A0000-B7FFF free, which can be used by memory managers for additional DOS memory space. However, this precludes using any graphics mode software such as Windows. The original CGA card only used 16K of space starting at B8000, whereas a VGA adapter running in CGA emulation (Color Text) mode can activate 32K of RAM at this address. The original CGA card has no onboard BIOS and is instead operated by driver programs found in the primary motherboard BIOS.

Enhanced Graphics Adapter (EGA) Memory. The Enhanced Graphics Adapter (EGA) uses all 128K of the video RAM from A0000-BFFFF. The ROM code used to operate this adapter is on the adapter and consumes 16K of memory from C0000-C3FFF.

The original IBM EGA card only used 16K of ROM space at C0000. Aftermarket compatible EGA adapters can use additional ROM space up to 32K total. The most interesting thing to note about EGA (and this applies to VGA adapters, as well) is that segments A000 and B000 are not all used at all times. For example, if the card is in a graphics mode, only segment A000 would appear to have RAM installed, whereas segment B000 would appear completely empty. If you switched the mode of the adapter (through software) into Color Text mode, segment A000 would instantly appear empty, and the last half of segment B000 would suddenly "blink on." The monochrome text mode RAM area would practically never be used on a modern system, because little or no software would ever need to switch the adapter into that mode.

The EGA card became somewhat popular after it appeared, but this was quickly overshadowed by the VGA card that followed. Most of the VGA characteristics with regard to memory are the same as the EGA because the VGA is backward compatible with EGA.

Video Graphics Array (VGA) Memory. All VGA-compatible cards, including Super VGA cards, are almost identical to the EGA in terms of memory use. Just as with the EGA, they use all 128K of the video RAM from A0000-BFFFF, but not all at once. Again, the video RAM area is split into three distinct regions, and each of these regions is used only when the adapter is in the corresponding mode. One minor difference with the EGA cards is that virtually all VGA cards use the full 32K allotted to them for onboard ROM (C0000 to C7FFF). Figure 5.15 shows the VGA adapter memory map.

You can see that the typical VGA card uses a full 32K of space for the onboard ROM containing driver code. Some VGA cards may use slightly less, but this is rare. Just as with the EGA card, the video RAM areas are only active when the adapter is in the particular mode designated. In other words, when a VGA adapter is in graphics mode, only segment A000 is used; when it is in color text mode, only the last half of segment B000 is used. Because the VGA adapter is almost never run in monochrome text mode, the first half of segment B000 remains unused (B0000-B7FFF). Figure 5.15

also shows the standard motherboard ROM BIOS so that you can get a picture of how the entire UMA is laid out with this adapter.

FIG. 5.15 *The VGA (and Super VGA) adapter memory map.*

Systems that use the LPX (Low Profile) motherboard design in an LPX- or Slimline-type case incorporate the video adapter into the motherboard. In these systems, even though the video BIOS and motherboard BIOS may be from the same manufacturer, they are always set up to emulate a standard VGA-type adapter card. In other words, the video BIOS appears in the first 32K of segment C000 just as if a standalone VGA-type card were plugged into a slot. The built-in video circuit in these systems can be easily disabled via a switch or jumper, which then allows a conventional VGA-type card to be plugged in. By having the built-in VGA act exactly as if it were a separate card, disabling it allows a new adapter to be installed with no compatibility problems that might arise if the video drivers had been incorporated into the motherboard BIOS.

If you were involved with the PC industry in 1987, you might remember how long it took for clone video card manufacturers to accurately copy the IBM VGA circuits. It took nearly two years (almost to 1989) before you could buy an aftermarket VGA card and expect it to run everything an IBM VGA system would with no problems. Some of my associates who bought some of the early cards inadvertently became members of the video card manufacturer's "ROM of the week" club! They were constantly finding problems with the operation of these cards, and many updated and patched ROMs were sent to try and fix the problems. Not wanting to pay for the privilege of beta testing the latest attempts at VGA compatibility, I bit the bullet and took the easy way out. I bought the IBM VGA card (PS/2 Display Adapter) for \$595. Note that a top line AGP video card today normally runs for less than this!

Although the card worked very well in most situations, I did find compatibility problems with the memory use of this card and a few other cards, mainly SCSI adapters. This was my first introduction to what I call *scratch pad memory* use by an adapter. I found that many different types of adapters may use some areas in the UMA for mapping scratch pad memory. This refers to memory on the card that stores status information, configuration data, or any other temporary type information of a variable nature. Most cards keep this scratch pad memory to themselves and do not attempt to map it into the processor's address space. But some cards do place this type of memory in the address space so that the driver programs for the card can use it. Figure 5.16 shows the memory map of the IBM PS/2 Display Adapter (IBM's original VGA card).

FIG. 5.16 *IBM's ISA-bus VGA card (PS/2 Display Adapter) memory map.*

There is nothing different about this VGA card and any other with respect to the video RAM area. What is different is that the ROM code that operates this adapter only consumes 24K of memory from C0000-C5FFF. Also strange is the 2K "hole" at C6000, and the 6K of scratch pad memory starting at C6800, and the additional 2K of scratch pad memory at CA000. In particular, the 2K "straggler" area really caught me off guard when I installed a SCSI host adapter in this system that had a 16K onboard BIOS with a default starting address of C8000. I immediately ran into a conflict that completely disabled the system. In fact, it would not boot, had no display at all, and could only beep out error codes that indicated that the video card had failed. I first thought that I had somehow "fried" the card, but removing the new SCSI adapter had everything functioning normally. I also could get the system to work with the SCSI adapter and an old CGA card substituting for the VGA, so I immediately knew a conflict was afoot. This scratch pad memory use was not documented clearly

in the technical-reference information for the adapter, so it was something that I had to find out by trial and error. If you have ever had two cards that don't work together in the same system, this is one example of a potential conflict that can disable the system completely!

See "Adapter Memory Configuration and Optimization," p. 385

Needless to say, nothing could be done about this 2K of scratch pad memory hanging out there, and I had to work around it as long as I had this card in the system. I solved my SCSI adapter problem by merely moving the SCSI adapter BIOS to a different address.

NOTE

I have seen other VGA-type video adapters use scratch pad memory, but they have all kept it within the C0000-C7FFF 32K region allotted normally for the video ROM BIOS. By using a 24K BIOS, I have seen other cards with up to 8K of scratch pad area, but none--except for IBM's--in which the scratch pad memory goes beyond C8000.

Adapter ROM and Special Purpose RAM Memory. The second 128K of upper memory beginning at segment C000 is reserved for the software programs, or BIOS (basic input/output system), on the adapter boards plugged into the system slots. These BIOS programs are stored on special chips known as read-only memory (ROM). Most adapters today use EEPROM (Electrically Erasable Programmable ROM) or flash ROM, which can both be erased and reprogrammed right in the system without removing the chip or card. Updating the flash ROM is as simple as running the update program you get from the manufacturer and following the directions onscreen. It pays to check periodically with your card manufacturers to see if they have flash ROM updates for their cards.

ROM is useful for semi-permanent programs that always must be present while the system is running, and especially for booting. Graphics boards, hard disk controllers, communications boards, and expanded memory boards, for example, are adapter boards that might have adapter ROM. These adapter ROMs are in a separate area of memory from the VGA video RAM area and the motherboard ROM as well.

On systems based on the 386 CPU chip or higher, memory managers that are included with DOS 6 or third-party programs can load device drivers and memory-resident programs into unused regions in the UMA.

To actually move the RAM usage on any given adapter requires that you consult the documentation for the card. Most older cards require that specific switches or jumpers be changed, and the settings will probably not be obvious without the manual. Most newer cards, especially those that are Plug and Play, allow these settings to be changed by software that either comes with the card, or the Device Manager program that goes with some of the newer operating systems such as Windows 95/98 or NT 5.0 and newer.

Video Adapter BIOS. The video adapter BIOS handles the translation between the video card and basic VGA mode graphics instructions. Remember that your system is in a basic VGA mode during boot, and whenever you are running Windows in safe mode. Although 128K of upper memory beginning at segment C000 is reserved for use by adapter BIOSs, not all this space is used by various video adapters commonly found on PCs. Table 5.12 details the amount of space used by the BIOS on each type of common video adapter card.

Table 5.12 Memory Used by Different Video Cards

Type of Adapter	Adapter BIOS Memory Used
Monochrome Display Adapter (MDA)	None - Drivers in Motherboard BIOS
Color Graphics Adapter (CGA)	None - Drivers in Motherboard BIOS
Enhanced Graphics Adapter (EGA)	16K onboard (C0000-C3FFF)
Video Graphics Array (VGA)	32K onboard (C0000-C7FFF)
Super VGA (SVGA)	32K onboard (C0000-C7FFF)

Depending on the basic VGA mode selected (Color Text, Monochrome Text, or VGA Graphics), the video card will use all the 128K of upper memory beginning at segment C000. In addition, these graphics cards may contain up to 8M or more of onboard memory for use in their native high-resolution modes in which to store currently displayed data and more quickly fetch new screen data for display.

Hard Disk Controller and SCSI Host Adapter BIOS. The upper memory addresses C0000 to DFFFF also are used for the built-in BIOS contained on many hard drive and SCSI controllers. Table 5.13 details the amount of memory and the addresses commonly used by the BIOS contained on hard drive adapter cards.

Table 5.13 Memory Addresses Used by Different Hard Drive Adapter Cards

Disk Adapter Type	Onboard BIOS Size	BIOS Address Range
Most XT Compatible Controllers	8K	C8000-C9FFF
Most AT Controllers	None	Drivers in Motherboard BIOS
Most Standard IDE Adapters	None	Drivers in Motherboard BIOS
Most Enhanced IDE Adapters	16K	C8000-CBFFF
Some SCSI Host Adapters	16K	C8000-CBFFF
Some SCSI Host Adapters	16K	DC000-DFFFF

The hard drive or SCSI adapter card used on a particular system may use a different amount of memory, but it is most likely to use the memory segment beginning at C800 because this address is considered part of the IBM standard for personal computers. Virtually all the disk controller or SCSI adapters today that have an onboard BIOS allow the BIOS starting address to be easily moved in the C000 and D000 segments. The locations listed in Table 5.13 are only the default addresses that most of these cards use. If the default address is already in use by another card, you have to consult the documentation for the new card to see how to change the BIOS starting address to avoid any conflicts.

Figure 5.17 shows an example memory map for an Adaptec AHA-2940 SCSI adapter.

FIG. 5.17 *Adaptec AHA-2940U SCSI adapter default memory use.*

Note how this SCSI adapter fits in here. Although no conflicts are in the UMA memory, the free regions have been fragmented by the placement of the SCSI BIOS. Because most systems do not have

any BIOS in segment E000, that remains as a free 64K region. With no other adapters using memory, this example shows another free UMB (Upper Memory Block) starting at C8000 and continuing through DBFFF, which represents an 80K free region. Using the EMM386 driver that comes with DOS or Windows, memory can be mapped into these two regions for loading 16-bit memory-resident drivers and programs. Note that this will have no effect on 32-bit Windows 95/98 or NT drivers, as those are loaded into extended memory automatically. Unfortunately, because programs cannot be split across regions, the largest 16-bit driver program you could load is 80K, which is the size of the largest free region. It would be much better if you could move the SCSI adapter BIOS so that it was next to the VGA BIOS, as this would bring the free UMB space to a single region of 144K. It is much easier and more efficient to use a single 144K region than two regions of 80K and 64K, respectively. Note again that manipulating memory in this manner is not necessary with 32-bit drivers, because they are automatically loaded into extended memory anyway.

Fortunately, it is possible to move this particular SCSI adapter because it is a Plug-and-Play card. This means you can control the resource settings via the Win95/98 Device Manager.

If you have an older non-Plug-and-Play card, then you will likely need the documentation to discover the switch and jumper settings for the card, as most manufacturers don't clearly label them. If you can't find your original documentation, you can consult the Micro House PC Hardware Library on the CD with this book, which contains an extensive database of adapter cards, motherboards, and hard drives showing jumper settings, configuration information, manufacturer information, and so on.

Most cards sold since '96 have been Plug and Play, meaning that they are fully software configurable.

After changing the appropriate switches to move the SCSI adapter BIOS to start at C8000, the optimized map would look like Figure 5.18.

FIG. 5.18 *Adaptec AHA-2940U SCSI adapter with optimized memory use.*

Notice how the free space is now a single contiguous block of 144K. This represents a far more optimum setup than the default settings.

Network Adapters. Network adapter cards also can use upper memory in segments C000 and D000. The exact amount of memory used and the starting address for each network card vary with the type and manufacturer of the card. Some network cards do not use any memory at all. A network card might have two primary uses for memory. They are as follows:

- IPL (Initial Program Load or Boot) ROM
- Shared Memory (RAM)

An *IPL ROM* is usually an 8K ROM that contains a bootstrap loader program that allows the system to boot directly from a file server on the network. This allows the removal of all disk drives from the PC, creating a diskless workstation. Because no floppy or hard disk would be in the system to boot from, the IPL ROM gives the system the instructions necessary to locate an image of the operating system on the file server and load it as if it were on an internal drive. If you are not using your system as a diskless workstation, it would be beneficial to disable any IPL ROM or IPL ROM socket on the adapter card. Note that many network adapters do not allow this socket to be disabled, which means that you lose the 8K of address space for other hardware, even if the ROM chip is removed from the

socket!

Shared memory refers to a small portion of RAM contained on the network card that is mapped into the PC's upper memory area. This region is used as a memory window onto the network and offers very fast data transfer from the network card to the system. IBM pioneered the use of shared memory for its first Token Ring network adapters, and now shared memory is in common use among other companies' network adapters. Shared memory was first devised by IBM because it found that transfers using the DMA channels were not fast enough in most systems. This had mainly to do with some quirks in the DMA controller and bus design, which especially affected 16-bit ISA bus systems. Network adapters that do not use shared memory will either use DMA or Programmed I/O (PIO) transfers to move data to and from the network adapter.

Although shared memory is faster than either DMA or PIO for ISA systems, it does require 16K of UMA space to work. Most standard performance network adapters use PIO because this makes them easier to configure, and they require no free UMA space, whereas most high performance adapters will use shared memory. The shared memory region on most network adapters that use one is usually 16K in size and may be located at any user-selected 4K increment of memory in segments C000 or D000.

Figure 5.19 shows the default memory addresses for the IPL ROM and shared memory of an IBM Token Ring network adapter, although many other network adapters such as Ethernet adapters would be similar. One thing to note is that most Ethernet adapters use either a DMA channel or standard PIO (Programmed I/O) commands to send data to and from the network, and don't use shared memory like many Token Ring cards.

FIG. 5.19 *Network adapter default memory map.*

I have also included the standard VGA video BIOS in Figure 5.19 because nearly every system would have a VGA-type video adapter as well. Note that these default addresses for the IPL ROM and the shared memory can easily be changed by reconfiguring the adapter. Most other network adapters are similar in that they also would have an IPL ROM and a shared memory address, although the sizes of these areas and the default addresses may be different. Most network adapters that incorporate an IPL ROM option can disable the ROM and socket such that those addresses are not needed at all. This helps to conserve UMA space and prevent possible future conflicts if you are never going to use the function.

Notice in this case that the SCSI adapter used in Figure 5.20 would fit both at its default BIOS address of DC000, and the optimum address of C8000. The Token-Ring shared memory location is not optimum and causes the UMB space to be fragmented. By adjusting the location of the shared memory, this setup can be greatly improved. Figure 5.20 shows an optimum setup with both the Token-Ring adapter and the SCSI adapter in the same machine.

FIG. 5.20 *Adaptec AHA-2940U SCSI adapter and network adapter with optimized memory use.*

This configuration allows a single 120K UMB that can very efficiently be used to load software drivers. Notice that the IPL ROM was moved to D0000, which places it as the last item installed before the free memory space. This is because if the IPL function is not needed, it can be disabled and the UMB space would increase to 128K and still be contiguous. If the default settings are used for both the SCSI and network adapters, the UMB memory would be fragmented into three regions of

16K, 40K, and 64K. The memory would still function, but it is hardly an optimum situation.

Note that the Plug-and-Play feature in Win95/98 and NT 5.0 does not attempt to optimize memory use, only to resolve conflicts. Of course, with 32-bit drivers, the location and size of free Upper Memory Blocks don't really matter, as 32-bit drivers will be loaded into extended memory. If you are still occasionally running DOS-based programs such as games, you will want to manually optimize the upper memory area configuration for maximum memory and performance.

Other ROMs in the Upper Memory Area. In addition to the BIOS for hard drive controllers, SCSI adapters, and network cards, upper memory segments C000 and D000 are used by some terminal emulators, security adapters, memory boards, and various other devices and adapter boards. Some adapters may require memory only for BIOS information, and others may require RAM in these upper memory segments. For information on a specific adapter, consult the manufacturer's documentation.

Motherboard BIOS Memory. The last 128K of reserved memory is used by the motherboard BIOS. The BIOS programs in ROM control the system during the boot-up procedure and remain as drivers for various hardware in the system during normal operation. Because these programs must be available immediately, they cannot be loaded from a device such as a disk drive. Four main programs stored in most motherboard ROMs are as follows:

- *Power-On Self Test*, the POST, is a set of routines that tests the motherboard, memory, disk controllers, video adapters, keyboard, and other primary system components. This routine is useful when you troubleshoot system failures or problems.

See "The Power-On Self Test (POST)," p. 984

- The *bootstrap loader* routine initiates a search for an operating system on a floppy disk or hard disk. If an operating system is found, it is loaded into memory and given control of the system.

See "The Boot Process," p. 1042

- The *BIOS* is the software interface to all the hardware in the system. The BIOS is a collection of individual drivers for the hardware in the system. With the BIOS, a program easily can access features in the system by calling on a standard BIOS driver function instead of talking directly to the device.

See "BIOS," p. 208

- The *CMOS Setup* program is a menu-driven application that is used for system configuration and setup. It is normally activated at boot time by pressing the proper key. This program is used to set basic system configuration parameters and BIOS features, motherboard and chipset features, user preferences and security features (passwords), and in some cases to run limited diagnostics. Not all systems have the CMOS Setup program in ROM; some must load it from a floppy disk or hard disk instead.

Both segments E000 and F000 in the memory map are considered reserved for the motherboard

BIOS, but only some systems actually use this entire area. Older 8-bit systems require only segment F000 and enable adapter card ROM or RAM to use segment E000. Most 16-bit or greater systems use all of F000 for the BIOS, and may decode but not use any of segment E000. By decoding an area, the motherboard essentially grabs control of the addresses, which precludes installing any other hardware in this region. In other words, it is not possible to configure adapter cards to use this area. That is why you will find that most adapters that use memory simply do not allow any choices for memory use in segment E000. Although this may seem like a waste of 64K of memory space, any 386 or higher system can use the powerful MMU in the processor to map RAM from extended memory into segment E000 as an Upper Memory Block, and subsequently use it for loading 16-bit driver software if necessary. This is a nice solution to what otherwise would be wasted memory, although it is not important if you use 32-bit drivers.

Many different ROM-interface programs are in the IBM motherboards, but the location of these programs is mostly consistent.

Figure 5.21 shows the motherboard ROM BIOS memory use of most 16-bit or greater systems.

FIG. 5.21 *Motherboard ROM BIOS memory use of most systems.*

Note that the standard system BIOS uses only segment F000 (64K). In almost every case, the remainder of the BIOS area (segment E000) is completely free and can be used as UMA block space.

Extended Memory

As mentioned previously in this chapter, the memory map on a system based on the 286 or higher processor can extend beyond the 1M boundary that exists when the processor is in real mode. On a 286 or 386SX system, the extended memory limit is 16M; on a 386DX, 486, Pentium, Pentium MMX, or Pentium Pro system, the extended memory limit is 4G (4,096M). Systems based on the Pentium II processor have a limit of 64G (65,536M).

For a system to address memory beyond the first megabyte, the processor must be in *protected mode*--the native mode of 286 and higher processors. On a 286, only programs designed to run in protected mode can take advantage of extended memory. 386 and higher processors offer another mode, called *virtual real mode*, which enables extended memory to be, in effect, chopped into 1M pieces (each its own real-mode session). Virtual real mode also allows for several of these sessions to be running simultaneously in protected areas of memory. These can be seen as DOS prompt sessions or windows within Windows 95/98, NT, or OS/2. Although several DOS programs can be running at once, each still is limited to a maximum of 640K of memory because each session simulates a real-mode environment, right down to the BIOS and Upper Memory Area. Running several programs at once in virtual real mode, called *multitasking*, requires software that can manage each program and keep them from crashing into one another. OS/2, Windows 95/98, and Windows NT all do this.

The 286 and higher CPU chips also run in what is termed *real mode*, which enables full compatibility with the 8088 CPU chip installed on the PC/XT-type computer. Real mode enables you to run DOS programs one at a time on an AT-type system just like you would on a PC/XT. However, an AT-type system running in real mode, particularly a 386-, 486-, Pentium-, Pentium Pro-, or Pentium II-based system, is really functioning as little more than a turbo PC. In real mode, these processors can emulate the 8086 or 8088, but they cannot operate in protected mode at the same time. For that reason, the 386 and above also provide a virtual real mode that operates under protected mode. This

allows for the execution of real-mode programs under the control of a protected-mode operating system, such as Win95/98, NT, or OS/2.

NOTE

Extended memory is basically all memory past the first megabyte, which can only be accessed while the processor is in protected mode.

XMS Memory. The extended memory specification (XMS) was developed in 1987 by Microsoft, Intel, AST Corp., and Lotus Development to specify how programs would use extended memory. The XMS specification functions on systems based on the 286 or higher and allows real-mode programs (those designed to run in DOS) to use extended memory and another block of memory usually out of the reach of DOS.

Before XMS, there was no way to ensure cooperation between programs that switched the processor into protected mode and used extended memory. There was also no way for one program to know what another had been doing with the extended memory because none of them could see that memory while in real mode. HIMEM.SYS becomes an arbitrator of sorts that first grabs all the extended memory for itself and then doles it out to programs that know the XMS protocols. In this manner, several programs that use XMS memory can operate together under DOS on the same system, switching the processor into and out of protected mode to access the memory. XMS rules prevent one program from accessing memory that another has in use. Because Windows 3.x is a program manager that switches the system to and from protected mode in running several programs at once, it has been set up to require XMS memory to function. Windows 95 operates mostly in protected mode, but still calls on real mode for access to many system components. Windows NT is a true protected-mode operating system, as is OS/2.

Extended memory can be made to conform to the XMS specification by installing a device driver in the CONFIG.SYS file. The most common XMS driver is HIMEM.SYS, which is included with Windows 3.x and later versions of DOS, starting with 4.0 and up. Windows 95/98 and NT automatically allow XMS functions in DOS prompt sessions, and you can configure full-blown DOS-mode sessions to allow XMS functions as well.

High Memory Area (HMA) and the A20 line. The High Memory Area (HMA) is an area of memory 16 bytes short of 64K in size, starting at the beginning of the first megabyte of extended memory. It can be used to load device drivers and memory-resident programs to free up conventional memory for use by real-mode programs. Only one device driver or memory-resident program can be loaded into HMA at one time, no matter what its size. Originally, this could be any program, but Microsoft decided that DOS could get there first, and built capability into DOS 5 and newer versions.

The HMA area is extremely important to those who use DOS 5 or higher because these DOS versions can move their own kernel (about 45K of program instructions) into this area. This is accomplished simply by first loading an XMS driver (such as HIMEM.SYS) and adding the line **DOS=HIGH** to your CONFIG.SYS file. Taking advantage of this DOS capability frees another 45K or so of conventional memory for use by real-mode programs by essentially moving 45K of program code into the first segment of extended memory. Although this memory was supposed to be accessible in protected mode only, it turns out that a defect in the design of the original 286 (which, fortunately, has been propagated forward to the more recent processors as a "feature") accidentally allows access to most of the first segment of extended memory while still in real mode.

The use of the HMA is controlled by the HIMEM.SYS or equivalent driver. The origins of this memory usage are interesting because they are based on a bug in the original 286 processor carried forward through even the Pentium II.

The problem started from the fact that memory addresses in Intel processors are dictated by an overlapping segment and offset address. By setting the segment address to FFFF--which itself specifies an actual address of FFFF0, which is 16 bytes from the end of the first megabyte--and then specifying an offset of FFFF, which is equal to 64K, you can create a memory address as follows:

```

    FFFF  segment
+   FFFF  offset
-----
= 10FFEF total

```

This type of address is impossible on an 8088 or 8086 system that has only 20 address lines and therefore cannot calculate an address that large. By leaving off the leading digit, these processors interpret the address as 0FFEF, in essence causing the address to "wrap around" and end up 16 bytes from the end of the first 64K segment of the first megabyte. The problem with the 286 and higher was that when they were in real mode, they were supposed to operate the same way, and the address should wrap around to the beginning of the first megabyte also. Unfortunately, a "bug" in the chip left the 21st address line active (called the A20 line), which allowed the address to end up 16 bytes from the end of the first 64K segment in the second megabyte. This memory was supposed to be addressable only in protected mode, but this bug allowed all but 16 bytes of the first 64K of extended memory to be addressable in real mode.

Because this bug caused problems with many real-mode programs that relied on the wrap to take place, when IBM engineers designed the AT, they had to find a way to disable the A20 line while in real mode, but then re-enable it when in protected mode. They did this by using some unused pins on the 8042 keyboard controller chip on the motherboard. The 8042 keyboard controller was designed to accept scan codes from the keyboard and transmit them to the processor, but there were unused pins not needed strictly for this function. So IBM came up with a way to command the keyboard controller to turn on and off the A20 line, thus enabling the "defective" 286 to truly emulate an 8088 and 8086 while in real mode.

Microsoft realized that you could command the 8042 keyboard controller to turn back on the A20 line strictly for the purpose of using this "bug" as a feature, which enabled you to access the first 64K of extended memory (less 16 bytes) without having to go through the lengthy and complicated process of switching into protected mode. Thus, HIMEM.SYS and the High Memory Area was born! HIMEM.SYS has to watch the system to see if the A20 line should be off for compatibility, or on to enable access to the HMA or while in protected mode. In essence, HIMEM becomes a control program that manipulates the A20 line through the 8042 keyboard controller chip.

Expanded Memory

Some older programs can use a type of memory called *Expanded Memory Specification* or EMS memory. Unlike conventional (the first megabyte) or extended (the second through 16th or 4,096th megabytes) memory, expanded memory is *not* directly addressable by the processor. Instead, it can only be accessed through a 64K window and small 16K pages established in the UMA. Expanded memory is a segment or bank-switching scheme in which a custom memory adapter has a large

number of 64K segments onboard, combined with special switching and mapping hardware. The system uses a free segment in the UMA as the home address for the EMS board. After this 64K is filled with data, the board rotates the filled segment out and a new, empty segment appears to take its place. In this fashion, you have a board that can keep on rotating in new segments to be filled with data. Because only one segment can be seen or operated on at one time, EMS is very inefficient for program code and is normally only used for data.

Figure 5.22 shows how expanded memory fits with conventional and extended memory.

Intel originally created a custom-purpose memory board that had the necessary EMS bank-switching hardware. They called these boards *Above Boards*, and they were widely sold many years ago. EMS was designed with 8-bit systems in mind and was appropriate for them because they had no capability to access extended memory. 286 and newer systems, however, possess the capability to have 15 or more megabytes of extended memory, which is much more efficient than the goofy (and slow) bank-switching EMS scheme. The Above Boards are no longer being manufactured, and EMS memory--as a concept and functionally--is extremely obsolete.

If you have any antique software that still requires EMS memory, you are advised to upgrade to newer versions that can use extended memory directly. It is also possible to use the powerful MMU of the 386 and higher processors to convert extended memory to function like LIM EMS, but this should only be done if there is no way to use the extended memory directly. EMM386 can convert extended to expanded, and in fact was originally designed for this purpose, although today it is more likely to be used to map extended memory into the UMA for the purposes of loading drivers and not for EMS. The EMM386 driver is included with DOS versions 5 and newer as well as with Windows. If you have several versions on hand, as a rule, always use the newest one.

FIG. 5.22 *Conventional, extended, and expanded memory.*

Preventing ROM BIOS Memory Conflicts and Overlap

As detailed in previous sections, C000 and D000 are reserved for use by adapter-board ROM and RAM. If two adapters have overlapping ROM or RAM addresses, usually neither board operates properly. Each board functions if you remove or disable the other one, but they do not work together.

With many adapter boards, you can change the actual memory locations to be used with jumpers, switches, or driver software, which might be necessary to allow two boards to coexist in one system. This type of conflict can cause problems for troubleshooters. You must read the documentation for each adapter to find out what memory addresses the adapter uses and how to change the addresses to allow coexistence with another adapter. Most of the time, you can work around these problems by reconfiguring the board or changing jumpers, switch settings, or software-driver parameters. This change enables the two boards to coexist and stay out of each other's way.

Additionally, you must ensure that adapter boards do not use the same IRQ (Interrupt Request Line), DMA (direct memory access) channel, or I/O Port address. You can easily avoid adapter board memory, IRQ, DMA channel, and I/O Port conflicts by creating a chart or template to mock up the system configuration by penciling on the template the resources already used by each installed adapter. You end up with a picture of the system resources and the relationship of each adapter to the others. This procedure helps you anticipate conflicts and ensures that you configure each adapter board correctly the first time. The template also becomes important documentation when you

consider new adapter purchases. New adapters must be configurable to use the available resources in your system.

See "System Resources," p. 269

If your system has Plug-and-Play capabilities, and you use PnP adapters, it will be able to resolve conflicts between the adapters by moving the memory usage on any conflict. Unfortunately, this routine is not intelligent and still requires human intervention--that is, manual specification of addresses in order to achieve the most optimum location for the adapter memory.

ROM Shadowing

Computers based on the 386 or higher CPU chip, which provides memory access on a 32- or 64-bit path, often use a 16-bit data path for system ROM BIOS information. In addition, adapter cards with onboard BIOS may use an 8-bit path to system memory. On these high-end computers, using a 16- or 8-bit path to memory is a significant bottleneck to system performance. In addition to these problems of width, most actual ROM chips are available in maximum speeds far less than what is available for the system's dynamic RAM. For example, the fastest ROMs available are generally 150ns to 200ns, whereas the RAM in a modern system is rated at 60ns or faster in most cases.

Because of the fact that ROM is so slow, any system accesses to programs or data in ROM cause many additional wait states to be inserted. These wait states can slow the entire system down tremendously, especially considering that many of the 16-bit driver programs used constantly by DOS reside in the BIOS chips found on the motherboard and many of the installed adapters. Fortunately, a way was found to transfer the contents of the slow 8- or 16-bit ROM chips into much faster 32-bit main memory. This is called *shadowing the ROMs*.

Virtually all 386 and higher systems enable you to use what is termed *shadow memory* for the motherboard and possibly some adapter ROMs as well. Shadowing essentially moves the programming code from slow ROM chips into fast 32-bit system memory. Shadowing slower ROMs by copying their contents into RAM can greatly speed up these BIOS routines--sometimes making them four to five times faster.

Shadowing is accomplished by using the powerful MMU in the 386 and higher processors. With the appropriate instructions, the MMU can take a copy of the ROM code, place it in RAM, and enable the RAM such that it appears to the system in exactly the same addresses at which it was originally located. This actually disables the ROM chips themselves, which are essentially shut down. The system RAM that is now masquerading as ROM is fully write-protected so that it acts in every way just like the real ROM, with the exception of being much faster, of course! Most systems have an option in the system setup to enable shadowing for the motherboard BIOS (usually segment F000) and the video BIOS (usually the first 32K of segment C000). Some systems will go further and offer you the capability to enable or disable shadowing in increments (usually 16K) throughout the remainder of the C000 and D000 segments.

NOTE

The important thing to note about shadowing is that if you enable shadowing for a given set of addresses, anything found there when the system is booting will be copied to RAM and locked in place. If you were to do this to a memory range that had a network

adapter's shared memory mapped into it, the network card would cease to function. You must only shadow ranges that contain true ROM and no RAM.

Some systems do not offer shadowing for areas other than the motherboard and video BIOS. In these systems, you can use a memory manager such as EMM386 (which comes with DOS and Windows) to enable shadowing for any range you specify. It is preferable to use the system's own internal shadowing capabilities first because the system shadowing uses memory that would otherwise be discarded. Using an external memory manager such as EMM386 for shadowing costs you a small amount of extended memory, equal to the amount of space you are shadowing.

If you enable shadowing for a range of addresses, and one or more adapters or the system in general no longer works properly, you may have scratch pad memory or other RAM within the shadowed area, which is not accessible as long as the shadowing remains active. In this case, you should disable the shadowing for the system to operate properly. If you can figure out precisely which addresses are ROM and which are RAM within the Upper Memory Area, you can selectively shadow only the ROM for maximum system performance.

Note that shadowing ROMs is not very important when running a 32-bit operating system such as Win95/98 or NT. This is because those operating systems only use the 16-bit BIOS driver code during booting; then they load 32-bit replacement drivers into faster extended memory and use them. Thus, shadowing normally only affects DOS or other 16-bit software and operating systems.

Total Installed Memory Versus Total Usable Memory

One thing that most people don't realize is that not all the SIMM or other RAM memory you purchase and install in a system will be available. Because of some quirks in system design, the system usually has to "throw away" up to 384K of RAM to make way for the Upper Memory Area.

For example, most systems with 16M of RAM (which is 16,384K) installed show a total of only 16,000K installed during the POST or when running Setup. This indicates that $16,384K - 16,000K = 384K$ of missing memory! Some systems may show 16,256K with the same 16M installed, which works out to $16,384K - 16,256K = 128K$ missing.

If you run your Setup program and check out your base and extended memory values, you will find more information than just the single figure for the total shown during the POST. In most systems with 4,096K (4M), you have 640K base and 3,072K extended. In some systems, Setup reports 640K base and 3,328K extended memory, which is a bonus. In other words, most systems come up 384K short, but some come up only 128K short.

This shortfall is not easy to explain, but it is consistent from system to system. Say that you have a 486 system with two installed 72-pin (32-bit) 16M SIMMs. This results in a total installed memory of 32M in two separate banks because the processor has a 32-bit data bus. Each SIMM is a single bank in this system. Note that most cheaper 486 systems use the 30-pin (8-bit) SIMMs, of which four are required to make a single bank. The first bank (or SIMM, in this case) starts at address 0000000h (the start of the first megabyte), and the second starts at 1000000 (the start of the 17th megabyte).

One of the cardinal rules of memory is that you absolutely cannot have two hardware devices wired to the same address. This means that 384K of the first memory bank in this system would be in direct conflict with the video RAM (segments A000 and B000), any adapter card ROMs (segments C000

and D000), and of course the motherboard ROM (segments E000 and F000). This means that all SIMM RAM that occupies these addresses must be shut off or the system will not function! Actually, a motherboard designer can do three things with the SIMM memory that would overlap from A0000-FFFFF:

- Use the faster RAM to hold a copy of any slow ROMs (shadowing), disabling the ROM in the process.
- Turn off any RAM not used for shadowing, eliminating any UMA conflicts.
- Remap any RAM not used for shadowing, adding to the stack of currently installed extended memory.

Most systems shadow the motherboard ROM (usually 64K), the video ROM (32K), and simply turn off the rest. Some motherboard ROMs allow additional shadowing to be selected between C8000-DFFFF, usually in 16K increments.

NOTE

You can only shadow ROM, never RAM, so if any card (such as a network card) has a RAM buffer in the C8000-DFFFF area, you must not shadow the RAM buffer addresses or the card will not function. For the same reason, you cannot shadow the A0000-BFFFF area because this is the video adapter RAM buffer.

Most motherboards do not do any remapping, which means that any of the 384K not shadowed is simply turned off. That is why enabling shadowing does not seem to use any memory. The memory used for shadowing would otherwise be discarded in most systems. These systems would appear to be short by 384K compared to what is physically installed in the system. For example, in a system with 32M, no remapping would result in 640K of base memory and 31,744K of extended memory, for a total of 32,384K of usable RAM--384K short of the total (32,768K-384K).

Some systems shadow what they can and then remap any segments that do not have shadowing into extended memory so as not to waste the non-shadowed RAM. PS/2 systems, for example, shadow the motherboard BIOS area (E0000-FFFFF or 128K in these systems) and remap the rest of the first bank of SIMM memory (256K from A0000-DFFFF) to whatever address follows the last installed bank.

NOTE

Note that most new systems don't allow selecting ROM shadowing functions, nor do they do any remapping. These days, the small amount of memory gain just isn't worth the trouble in system engineering and design to accomplish it.

In my example system with two 16M 32-bit SIMMs, the 256K not used for shadowing would be remapped to 2000000-203FFFF, which is the start of the 33rd megabyte. This affects diagnostics because if you had any memory error reported in those addresses (2000000-203FFFF), it would indicate a failure in the first SIMM, even though the addresses point to the end of installed extended memory. The addresses from 1000000-1FFFFFF would be in the second SIMM, and the 640K base memory 0000000-009FFFF would be back in the first SIMM. As you can see, figuring out how the SIMMs are mapped into the system is not easy!

Most systems that do remapping can only remap an entire segment if no shadowing is going on within it. The video RAM area in segments A000 and B000 can never contain shadowing, so at least 128K can be remapped to the top of installed extended memory in any system that supports remapping. Because most systems shadow in segments F000 (motherboard ROM) and C000 (video ROM), these two segments cannot be remapped. This leaves 256K maximum for remapping. Any system remapping the full 384K must not be shadowing at all, which would slow down the system and is not recommended. Shadowing is always preferred over remapping, and remapping what is not shadowed is definitely preferred over simply turning off the RAM.

Systems that have 384K of "missing" memory do not do remapping. If you want to determine if your system has any missing memory, all you need to know are three things. One is the total physical memory actually installed. The other two items can be discovered by running your Setup program. You want to know the total base and extended memory numbers recognized by the system. Then simply subtract the base and extended memory from the total installed to determine the missing memory. You will usually find that your system is "missing" 384K, but you may be lucky and have a system that remaps 256K of what is missing and thus shows only 128K of memory missing.

Virtually all systems use some of the missing memory for shadowing ROMs, especially the motherboard and video BIOS, so what is missing is not completely wasted. Systems "missing" 128K will find that it is being used to shadow your motherboard BIOS (64K from F0000-FFFFF) and video BIOS (32K from C0000-C8000). The remainder of segment C0000 (32K from C8000-CFFFF) is simply being turned off. All other segments (128K from A0000-BFFFF and 128K from D0000-EFFFF) are being remapped to the start of the fifth megabyte (400000-43FFFF). Most systems simply disable these remaining segments rather than take the trouble to remap them. Remapping requires additional logic and BIOS routines to accomplish, and many motherboard designers do not feel that it is worth the effort to reclaim 256K.

NOTE

If your system is doing remapping, any errors reported near the end of installed extended memory are likely in the first bank of memory because that is where they are remapped from. The first bank in a 32-bit system would be constructed of either four 30-pin (8-bit) SIMMs or one 72-pin (32-bit) SIMM. The first bank in a 64-bit system would be constructed of either two 72-pin (32-bit) SIMMs or one 168-pin (64-bit) DIMM.

Adapter Memory Configuration and Optimization

Ideally, all adapter boards would be Plug-and-Play devices that require you to merely plug the adapter into a motherboard slot and then use it. With the new Plug-and-Play specification, we are moving toward that goal. However, sometimes it almost seems that adapter boards are designed as if they were the only adapter likely to be present on a system. They usually require you to know the upper memory addresses and IRQ and DMA channels already on your system, and how to configure the new adapter so that it does not conflict with your already-installed adapters.

Adapter boards use upper memory for their BIOS and as working RAM. If two boards attempt to use the same BIOS area or RAM area of upper memory, a conflict occurs that can keep your system from booting. The following sections cover ways to avoid these potential conflicts and how to troubleshoot if they do occur. In addition, these sections discuss moving adapter memory to resolve conflicts and provide some ideas on optimizing adapter memory use.

How to Determine What Adapters Occupy the UMA. You can determine what adapters are using space in upper memory in the following two ways:

- Study the documentation for each adapter on your system to determine the memory addresses they use.
- Use a software utility that can quickly determine for you what upper memory areas your adapters are using.

The simplest way (although by no means always the most foolproof) is to use a software utility to determine the upper memory areas used by the adapters installed on your system. One such utility, Microsoft Diagnostics (MSD), comes with Windows 3.x and DOS 6 or higher versions. The Device Manager under System in the Windows 95 Control Panel also provides this information, as does the new System Information utility that comes with Windows 98. These utilities examine your system configuration and determine not only the upper memory used by your adapters, but also the IRQs used by each of these adapters.

True Plug-and-Play systems will also shut down one of the cards involved in a conflict to prevent a total system lockup. This may cause Windows to boot in safe mode.

After you run MSD, Device Manager, or another utility to determine your system's upper memory configuration, make a printout of the memory addresses used. Thereafter, you can quickly refer to the printout when you are adding a new adapter to ensure that the new board does not conflict with any devices already installed on your system.

Moving Adapter Memory to Resolve Conflicts. After you identify a conflict or potential conflict by studying the documentation for the adapter boards installed on your system or using a software diagnostic utility to determine the upper memory addresses used by your adapter boards, you may have to reconfigure one or more of your adapters to move the upper memory space used by a problem adapter.

Most adapter boards make moving adapter memory a somewhat simple process, enabling you to change a few jumpers or switches to reconfigure the board; in Plug-and-Play cards, you can use the configuration program that comes with the board or the Windows Device Manager to make the changes. The following steps help you resolve most conflicts that arise because adapter boards conflict with one another:

1. Determine the upper memory addresses currently used by your adapter boards and write them down.
2. Determine if any of these addresses are overlapping, which results in a conflict.
3. Consult the documentation for your adapter boards to determine which boards can be reconfigured so that all adapters have access to unique memory addresses.
4. Configure the affected adapter boards so that no conflict in memory addresses occurs.

For example, if one adapter uses the upper memory range C8000-CBFFF and another adapter uses the range CA000-CCFFF, you have a potential address conflict. One of these must be changed. Note that Plug-and-Play cards allow these changes to be made directly from the Windows Device Manager.

Optimizing Adapter Memory Use. On an ideal PC, adapter boards would always come configured so that the upper memory addresses they use immediately follow the upper memory addresses used by the previous adapter, with no overlap that would cause conflicts. Such an upper memory arrangement would not only be "clean," but would make it much more simple to use available upper memory for loading device drivers and memory-resident programs. However, this is not the case. Adapter boards often leave gaps of unused memory between one another--this is, of course, preferable to an overlap, but still is not the best use of upper memory.

Someone who wanted to make the most of their upper memory might consider studying the documentation for each adapter board installed on his or her system to determine a way to compact the upper memory used by each of these devices. For example, if it were possible on a particular system using the adapters installed on it, the use of upper memory could be more simple if you configured your adapter boards so that the blocks of memory they use fit together like bricks in a wall, rather than like a slice of Swiss cheese, as is the case on most systems. The more you can reduce your free upper memory to as few contiguous chunks as possible, the more completely and efficiently you can take advantage of the UMA.

Memory optimization in this manner is not really necessary if you are running 32-bit drivers as you would in a normal Win95/98 or NT system. This is mostly useful for running older DOS applications, games, and so on.

Taking Advantage of Unused Upper Memory

On systems using an older 16-bit operating system such as Windows 3.1 or DOS, memory-resident programs and device drivers can be moved into the UMA by using a memory manager such as the MEMMAKER utility or some other aftermarket utility. These memory management utilities examine the memory-resident programs and device drivers installed on your system, determine their memory needs, and then calculate the best way to move these drivers and programs into upper memory, thus freeing the conventional memory they used.

Using MEMMAKER is quite simple. Make a backup of your CONFIG.SYS and AUTOEXEC.BAT files so that you have usable copies if you need them to restore your system configuration; then run MEMMAKER from the DOS prompt. MEMMAKER will install required device drivers in your CONFIG.SYS file, and then begin optimizing your memory configuration. It does a decent job of freeing up conventional memory; however, with careful fine-tuning, an individual can perform feats of memory management using only the raw DOS HIMEM.SYS and EMM386.EXE drivers that no automatic program can do.

Only driver programs that run in the processor's real mode must be loaded within the first megabyte of memory. Because real-mode drivers are made up of 16-bit real mode program code, they cannot reside in extended memory, because only the first megabyte (base memory) is accessible when in real mode. DOS and Windows 3.x are 16-bit programs and utilize drivers that run in real mode, hence the need for the base memory optimization. With the number of drivers that people are using today, it can

be difficult to fit them all in the available UMA space while leaving enough base memory free to run applications.

Things are different with the newer operating systems. Windows 95/98, for example, uses primarily 32-bit protected mode drivers and program code, although there is still some 16-bit real-mode program code left. Windows NT and OS/2 are full 32-bit operating systems, and all their drivers and applications are made up of 32-bit protected mode instruction code. If you are using all 32-bit programs, then virtually no memory optimization is necessary in the first megabyte, as 32-bit programs are free to run in extended memory.

The following sections cover using memory-management software to optimize conventional memory, and additional ways to configure your system memory to make your system run as efficiently as possible. It is important to note that the DOS HIMEM.SYS and EMM386.EXE play an integral role in MEMMAKER's capability to move device drivers and memory-resident programs into upper memory. The next two sections describe using HIMEM.SYS and EMM386.EXE to configure extended and expanded memory.

NOTE

If you are running older 16-bit DOS applications under Windows 95, Windows NT, or OS/2, then you still need to know about base memory optimization. In these cases, you will be running the application in a DOS window, which, using the virtual real mode of the processor, can emulate the first megabyte of real-mode workspace. Using these 32-bit operating systems, you can customize how the application running in the DOS window sees the system, and how the system memory appears to be organized.

Using HIMEM.SYS (DOS). The DOS device driver HIMEM.SYS, which has been included with Windows and DOS 4.0 and higher, is used to configure extended memory to the XMS specification, as well as to enable the use of the first 64K of extended memory as the high memory area (HMA). HIMEM.SYS is installed by adding a line invoking the device driver to your CONFIG.SYS file.

Using EMM386.EXE (DOS). The program EMM386.EXE, which is included with DOS 5.0 and higher, is used primarily to map XMS memory (extended memory managed by HIMEM.SYS) into unused regions of the UMA. This allows programs to be loaded into these regions for use under DOS. EMM386 also has a secondary function of using XMS memory to emulate EMS version 4 memory, which can then be used by programs that need expanded memory. For more information on using EMM386.EXE, refer to Que's *Special Edition Using MS-DOS 6.2* or your DOS manual.

MS-DOS 6.x MEMMAKER. If you still run DOS applications and games that run in DOS, you can increase the amount of conventional memory available to software running in DOS on systems based on the 386 chip and above by running the MS DOS 6.x utility MEMMAKER. DOS 5 had the capability, using EMM386, to map extended memory into the UMA so that DOS could load memory-resident programs and drivers into the UMA. Unfortunately, this required an extensive knowledge of the upper memory configuration of a particular system, and trial and error to see what programs could fit into the available free regions. This process was difficult enough that many people were not effectively using their memory under DOS (and Windows).

To make things easier, when DOS 6 was released, Microsoft included a menu-driven program called MEMMAKER that determines the system configuration, automatically creates the proper EMM386

statements, and inserts them into the CONFIG.SYS file. By manipulating the UMA manually or through MEMMAKER and loading device drivers and memory-resident programs into upper memory, you can have more than 600K of free conventional memory.

Over the course of months or years of use, the installation programs for various software utilities often install so many memory-resident programs and device drivers in your AUTOEXEC.BAT and CONFIG.SYS files that you have too little conventional memory left to start all the programs you want to run. You may want to use MEMMAKER to free up more conventional memory for your programs. You can get help on MEMMAKER by typing **HELP MEMMAKER** at the DOS prompt. When you run the MEMMAKER utility, it automatically performs the following functions to free up more memory:

- Moves a portion of the DOS kernel into the HMA.
- Maps free XMS memory into unused regions in the UMA as UMBs, into which DOS can then load device drivers and memory-resident programs to free up the conventional memory these drivers and programs otherwise use.
- Modifies CONFIG.SYS and AUTOEXEC.BAT to cause DOS to load memory-resident programs and device drivers into UMBs.

Before running MEMMAKER, carefully examine your CONFIG.SYS and AUTOEXEC.BAT files to identify unnecessary device drivers and memory-resident programs. For example, the DOS device driver ANSI.SYS is often loaded in CONFIG.SYS to enable you to use color and other attributes at the DOS prompt and to remap the keys on your keyboard. If you are primarily a Windows user and do not spend much time at the DOS prompt, you can eliminate ANSI.SYS from your CONFIG.SYS file to free up the memory the driver is using.

TIP

TVER is another often-loaded driver that most people don't need. If you don't run utilities or programs that require a specific version of DOS, you can remove SETVER from your CONFIG.SYS file.

After you strip down CONFIG.SYS and AUTOEXEC.BAT to their bare essentials (it is advisable to make backup copies first), you are ready to run MEMMAKER to optimize your system memory. To run MEMMAKER, follow these steps:

1. Exit from any other programs you are running.
2. Start your network or any memory-resident programs and device drivers you absolutely need.
3. At the DOS prompt, type **MEMMAKER**.

The MEMMAKER setup runs in two modes--Express and Custom. Express setup is preferable for users who want to enable MEMMAKER to load device drivers and memory-resident programs into high memory with the minimum amount of user input, unless they have an EGA or VGA (but not a Super VGA) monitor. If you have an EGA or VGA monitor, choose Custom Setup and answer Yes in the advanced options screen where it asks whether MEMMAKER should use monochrome region (B0000-B7FFF) for running programs. Use the defaults for the rest of the options in Custom setup unless you are sure that one of the defaults is not correct for your system. Custom setup is probably

not a good idea unless you are knowledgeable about optimizing system memory, particular device drivers, and memory-resident programs on the system.

When MEMMAKER finishes optimizing the system memory, the following three lines are added to CONFIG.SYS:

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.E XE NOEMS
DOS=HIGH,UMB
```

In addition, MEMMAKER modifies each line in CONFIG.SYS and AUTOEXEC.BAT that loads a device driver or memory-resident program now being loaded into UMBs. Various `DEVICE=` lines in your CONFIG.SYS are changed to `DEVICEHIGH=`, and various lines in your AUTOEXEC.BAT have the `LH` (LoadHigh) command inserted in front of them. For example, the line `DEVICE=ANSI.SYS` is changed to `DEVICEHIGH=ANSI.SYS`. In your AUTOEXEC.BAT, lines such as `C:\DOS\DOSKEY` are changed to `LH C:\DOS\DOSKEY`. The `DEVICEHIGH` and `LH` commands load the device drivers and memory-resident programs into UMBs. MEMMAKER also adds codes to specify where in upper memory each program will be loaded. For example, after you run MEMMAKER, a statement like this might be added to your AUTOEXEC.BAT:

```
LH /L:1 C:\DOS\DOSKEY
```

The `/L:1` causes the resident program `DOSKEY` to load into the first UMB region. On many systems, MEMMAKER configures the system to free up 620K of conventional memory.