Chapter 7

Memory

This chapter looks at memory from both a physical and logical point of view. The chapter discusses the physical chips and SIMMs (Single Inline Memory Modules) that you can purchase and install. The chapter also looks at the logical layout of memory and defines 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 remains as perhaps the most difficult subject to grasp in the entire PC universe. This chapter contains much useful information that removes the mysteries associated with memory and enables you to get the most out of your system.

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. More than a decade after the first PC was introduced, even the newest Pentium-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 their 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

- Expanded memory (obsolete)
- Video RAM memory
- Adapter ROM and Special Purpose RAM
- Motherboard ROM BIOS

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 an AT system, the memory map extends beyond the 1M boundary and can continue to 16M on a system based on the 286 or higher processor, or as much as 4G (4,096M) on a 386DX or higher. Any memory past 1M is called *extended memory*.

Figure 7.1 shows the logical address locations for an IBM-compatible system. If the processor is running in Real Mode, then only the first megabyte is accessible. If the processor is in Protected Mode, then the full 16 or 4,096 megabytes are accessible. Each symbol is equal to 1K of memory, each line or segment is 64K, and this map shows the first two megabytes of system memory.

```
. = Program-accessible memory (standard RAM)
G = Graphics Mode Video RAM
M = Monochrome Text Mode Video RAM
C = Color Text Mode Video RAM
V = Video ROM BIOS (would be "a" in PS/2)
a = Adapter board ROM and special-purpose RAM (free UMA space)
r = Additional PS/2 Motherboard ROM BIOS (free UMA in non-PS/2 systems)
R = Motherboard ROM BIOS
b = IBM Cassette BASIC (would be "R" in IBM compatibles)
h = High Memory Area (HMA), if HIMEM.SYS is loaded.

Conventional (Base) Memory:

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
000000: ................................................................
010000: ................................................................
020000: ................................................................
030000: ................................................................
040000: ................................................................
050000: ................................................................
060000: ................................................................
070000: ................................................................
080000: ................................................................
090000: ................................................................

Upper Memory Area (UMA):

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
0B0000: MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0C0000: VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
0D0000: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---w
```

```
0E0000: rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbRRRRRRRR

Extended Memory:

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
100000: hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh

Extended Memory Specification (XMS) Memory:

110000: ................................................................
120000: ................................................................
130000: ................................................................
140000: ................................................................
150000: ................................................................
160000: ................................................................
170000: ................................................................
180000: ................................................................
190000: ................................................................
1A0000: ................................................................
1B0000: ................................................................
1C0000: ................................................................
1D0000: ................................................................
1E0000: ................................................................
1F0000: ................................................................
```

Fig. 7.1
The logical memory map.

Note

To save space, this map is ended after the end of the second megabyte. In reality, this map continues to the maximum of 16M or 4,096M 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 at the time the first PC was introduced was 512K. The other 512K was reserved for use by the system itself, including the motherboard and adapter boards plugged into the system slots.

IBM decided after introducing the system 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 indicated as reserved for use by the graphics boards, other adapters, and the motherboard ROM BIOS.

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 on-screen, the electronic impulses that contain their images reside in this space. Video RAM is allotted the address range from A0000-BFFFF.
- The next 128K is reserved for the software programs, or adapter BIOS, that reside in read-only memory chips on the adapter boards plugged into the system slots. Most VGA-compatible video adapters use the first 32K of this area for their on-board BIOS. The rest can be used by any other adapters installed. Adapter ROM and special purpose RAM are 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 DOS takes over, also reside in this space. Most systems only use the last 64K of this space, leaving the first 64K free for remapping with memory managers. The motherboard BIOS is allotted the address range from E0000-FFFFF.

Not all the 384K of reserved memory is fully used on most AT-type systems. For example, according to IBM's definition of the PC standard, reserved video RAM begins at address A0000, which is right at the 640K boundary, but this reserved area may not be used, depending on which video adapter is installed in the system. Some adapters do not use the area beginning at A0000, instead using a higher address. Different video adapters use varying amounts of RAM for their operations depending mainly on the mode they are in.

In fact, 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 device drivers (like ANSI.SYS) and memory-resident programs (like MOUSE.COM), which frees up the conventional memory they would otherwise require. 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. 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 ten floors, numbered from zero through nine; each

floor has 100 rooms, numbered from 00 to 99. The hotel also is broken up into segments of ten rooms each, and each segment can be specified by a two-digit number from 00 to 99. 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 that I am at segment 45 (room 450) offset 91 (450+91=541). That 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.

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 some of your system's memory to hold graphics or character information for display. Some adapters, like the VGA, also have on-board BIOS mapped into the system's space reserved for such types of adapters. Generally, the higher the resolution and color capabilities of the video adapter, the more system memory the video adapter uses. It is important to note that most VGA or Super VGA adapters have additional on-board memory used to handle the information currently displayed on-screen and to speed screen refresh.

In the standard system-memory map, a total of 128K is reserved for use by the video card to store currently displayed information. The reserved video memory is located in segments A000 and B000. The video adapter ROM uses additional upper memory space in segment C000.

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—which means all—of memory until the video adapter RAM is encountered. The use of 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 7.1 shows the maximum amount of memory available to DOS using the referenced video card.

Table 7.1 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, that can operate the 386+ Memory Management Unit (MMU), you can remap extended memory into this space.

The following sections examine how standard video adapters use the system's memory. Figures show where in a system the monochrome, EGA, VGA, and IBM PS/2 adapters use 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.

Monochrome Display Adapter Memory (MDA). Figure 7.2 shows where the original Monochrome Display Adapter (MDA) uses the system's memory. This adapter 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.

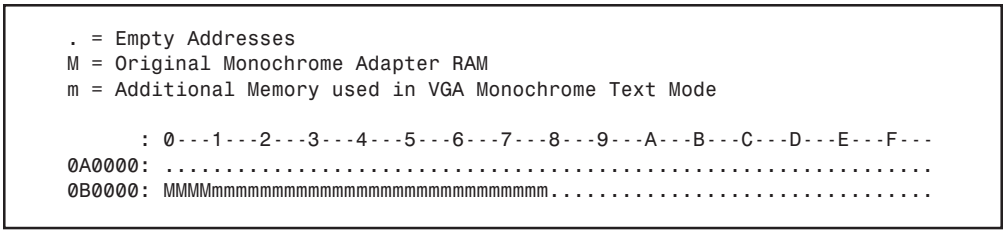

Fig. 7.2
The Monochrome Display Adapter memory map.

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 on-board BIOS, and instead is operated by driver programs found in the primary motherboard BIOS.

Color Graphics Adapter (CGA) Memory. Figure 7.3 shows where the Color Graphics Adapter (CGA) uses the system's memory. The 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.

```
. = Empty Addresses
C = Original Color Graphics Adapter (CGA) RAM
c = Additional Memory used in VGA Color Text Mode

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: ................................................................
0B0000: ................................CCCCCCCCCCCCCCCCcccccccccccccccc
```

Fig. 7.3
The Color Graphics Adapter (CGA) memory map.

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 on-board BIOS and is instead operated by driver programs found in the primary motherboard BIOS.

Enhanced Graphics Adapter (EGA) Memory. Figure 7.4 shows where the Enhanced Graphics Adapter (EGA) uses the system's memory. This adapter uses all 128K of the video RAM from A0000-BFFFF. The ROM code used to operate this adapter is on the adapter itself and consumes 16K of memory from C0000-C3FFF.

```
. = Empty Addresses
G = Enhanced Graphics Adapter (EGA) Graphics Mode Video RAM
M = EGA Monochrome Text Mode Video RAM
C = EGA Color Text Mode Video RAM
V = Standard EGA Video ROM BIOS
R = Standard Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
0B0000: MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0C0000: VVVVVVVVVVVVVVVV........................................................
0D0000: ................................................................
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.4
The Enhanced Graphics Adapter (EGA) memory map.

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, then 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, then 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. Figure 7.4 also shows the standard motherboard ROM BIOS so that you can get a picture of the entire UMA.

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 (Video Graphics Array) 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 on-board ROM (C0000 to C7FFF). Figure 7.5 shows the VGA adapter memory map.


```
. = Empty Addresses
G = Video Graphics Array (VGA) Adapter Graphics Mode Video RAM
M = VGA Monochrome Text Mode Video RAM
C = VGA Color Text Mode Video RAM
V = Standard VGA Video ROM BIOS
R = Standard Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
0B0000: MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0C0000: VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV................................
0D0000: ................................................................
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.5
The VGA (and Super VGA) adapter memory map.

You can see that the typical VGA card uses a full 32K of space for the on-board 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; and 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 7.5 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.

IBM created VGA, and the first systems to include VGA adapters were the PS/2 systems introduced in April 1987. These systems had the VGA adapter built directly into the motherboard. Because IBM had written both the video and motherboard BIOS, and they were building the VGA on the motherboard and not as a separate card, they incorporated the VGA BIOS driver code directly into the motherboard BIOS. This meant that segment C000 did not have an on-board video ROM as in most of the compatibles that would follow. Although this may sound like a bonus for the PS/2 systems—they still had the additional ROM code—it was merely located in segment E000 instead! In fact, the PS/2 systems used all of segment E000 for additional motherboard ROM BIOS code, whereas segment E000 remains empty in most compatibles.

Many compatibles today have their video adapter built into the motherboard. 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 stand-alone 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.

When the VGA first appeared on the scene, it was built into the PS/2 motherboard. So that you could add VGA to other systems, IBM also introduced at that time the very first VGA card. It was called the PS/2 Display Adapter, which was somewhat confusing at the time because it was designed for standard ISA bus-compatible systems and not for the PS/2s, which mostly used the new Micro Channel Architecture bus and already had VGA built-in. Nevertheless, this card was sold to anybody who wanted to add VGA to their ISA bus system. The IBM VGA card (PS/2 Display Adapter) was an 8-bit ISA card that had the same IBM-designed video chip and circuits as were used on the PS/2 motherboard.

If you were involved with the PC industry at that time, 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 simply bought the IBM VGA card—PS/2 Display Adapter. At that time, the card listed for $595, but I could usually expect a 30 percent discount in purchasing it at the local computer store. That is still about as much as you would pay for the best Local Bus Super VGA cards on the market today. In fact, you can now buy basic VGA clone cards for less than $20 that are actually faster and better than the original IBM VGA card!

I remember purchasing the IBM VGA card, not only because it was so expensive, but also because the card actually proved somewhat difficult to purchase! When I arrived to buy it at my local IBM authorized retailer (after first calling to see that they had one in stock), I asked for it by name. I said: "I am here to purchase an IBM PS/2 Display Adapter." The salesman asked what type of system I was going to install it in. I replied that I was going to put it in the AT clone I had assembled. Then I met resistance! The salesman told me that I could not use that in my clone because it was designed for PS/2 systems only. I told him he was mistaken and that the card is in fact designed for any IBM-compatible system with an ISA bus and is supposed to give the system the same graphics capability of the new PS/2s. The salesman maintained that the card was only for PS/2 systems, and I finally had to say "Look, I have cash, will you sell me the card or not?" Lo and behold I emerged from the store victorious, with the card in hand!

Although the card worked very well, and although I never did find any compatibility problems, I did later run into some interesting problems with the memory use of this card. 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 7.6 shows the memory map of the IBM PS/2 Display Adapter (IBM's VGA card).

```
. = Empty Addresses
G = Video Graphics Array (VGA) Adapter Graphics Mode Video RAM
M = VGA Monochrome Text Mode Video RAM
C = VGA Color Text Mode Video RAM
V = IBM VGA Video ROM BIOS
S = IBM VGA Scratch Pad memory (used by the card)
R = Standard Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
0B0000: MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0C0000: VVVVVVVVVVVVVVVVVVVVVVVV..SSSSSS........SS......................
0D0000: ................................................................
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.6
IBM's somewhat strange 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, as well as 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 on-board 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 got 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 underfoot. 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 the IBM VGA card and had conflicts with other adapters, now you know why! 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.

As a side 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.

As you can see in the preceding figures, each type of video adapter on the market uses two types of memory: video RAM, which stores the display information; and ROM code, which controls the adapter, must exist somewhere in the system's memory. The ROM code built into the motherboard ROM on standard PC and AT systems controls adapters such as the MDA and CGA. All the EGA and VGA adapters for the PC and AT systems use the full 128K of video RAM (not all at once of course) and up to 32K of ROM space at the beginning of segment C000. IBM's technical-reference manuals say that the memory between C0000 and C7FFF is reserved specifically for ROM on video adapter boards. Note that the VGA and MCGA built into the motherboards of the PS/2 systems have the ROM-control software built into the motherboard ROM in segments E000 and F000 and require no other code space in segment C000. Also note that you can often use your memory manager software (such as what comes with DOS) to map extended memory into the monochrome display area (32K worth), which can get you an extra 32K region for loading drivers and resident programs.

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), which have fused circuits so that the PC cannot alter them. ROM is useful for permanent programs that always must be present while the system is running. Graphics boards, hard disk controllers, communications boards, and expanded memory boards, for example, are adapter boards that might use some of this memory.

On systems based on the 386 CPU chip or higher, memory managers like the MS-DOS 6 MEMMAKER, IBM DOS RAMBOOST, or aftermarket programs like QEMM by Quarterdeck, can load device drivers and memory-resident programs into unused regions in the UMA.

Video Adapter BIOS. Although 128K of upper memory beginning at segment C000 is reserved for use by the video adapter BIOS, not all this space is used by various video adapters commonly found on PCs. Table 7.2 details the amount of space used by the BIOS on each type of common video adapter card.

Table 7.2 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)

Some more advanced graphics accelerator cards on the market do use most or all the 128K of upper memory beginning at segment C000 to speed the repainting of graphics displays in Windows, OS/2, or other graphical user interfaces (GUIs). In addition, these graphics cards may contain 2M of on-board memory in which to store currently displayed data and more quickly fetch new screen data as it is sent to the display by the CPU.

Hard Disk Controller and SCSI Host Adapter BIOS. The upper memory addresses C0000 to DFFFF also are used for the BIOS contained on many hard drive controllers. Table 7.3 details the amount of memory and the addresses commonly used by the BIOS contained on hard drive adapter cards.

Table 7.3 Memory Addresses Used by Different Hard Drive Adapter Cards

Disk Adapter Type	Onboard BIOS Size	BIOS Address Range
IBM XT 10M Controller	8K	C8000-C9FFF
IBM XT 20M Controller	4K	C8000-C8FFF
Most XT Compatible Controllers	8K	C8000-C9FFF
Most AT Controllers	None	Drivers in Motherboard BIOS
Most IDE Adapters	None	Drivers in Motherboard BIOS
Most ESDI Controllers	16K	C8000-CBFFF
Most SCSI Host Adapters	16K	C8000-CBFFF

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 on-board BIOS allow the BIOS starting address to be easily moved in the C000 and D000 segments. The locations listed in table 7.3 are only the default addresses that most of these cards use. If the default address is already in use by another card, then you have to consult the documentation for the new card to see how to change the BIOS starting address to avoid any conflicts.

Figure 7.7 shows an example memory map for an Adaptec AHA-1542CF SCSI adapter.

```
. = Empty Addresses
G = Video Graphics Array (VGA) Adapter Graphics Mode Video RAM
M = VGA Monochrome Text Mode Video RAM
C = VGA Color Text Mode Video RAM
V = Standard VGA Video ROM BIOS
H = SCSI Host Adapter ROM BIOS
R = Standard Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
0B0000: MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0C0000: VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV................................
0D0000: ................................................HHHHHHHHHHHHHHHH
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.7
Adaptec AHA-1542CF 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 CBFFF, which represents an 80K free region. Using the EMM386 driver that comes with DOS, memory can be mapped into these two regions for loading memory-resident drivers and programs. Unfortunately, because programs cannot be split across regions, the largest 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. Fortunately, it is possible to move this particular SCSI adapter, although doing so does require that several switches be reset on the card itself. One great thing about this Adaptec card is that a sticker is placed directly on the card detailing all the switch settings! This means that

you don't have to go hunting for a manual that may not be nearby. More adapter card manufacturers should place this information right on the card.

After changing the appropriate switches to move the SCSI adapter BIOS to start at C8000, the optimized map would look like figure 7.8.

```
. = Empty Addresses
G = Video Graphics Array (VGA) Adapter Graphics Mode Video RAM
M = VGA Monochrome Text Mode Video RAM
C = VGA Color Text Mode Video RAM
V = Standard VGA Video ROM BIOS
H = SCSI Host Adapter ROM BIOS
R = Standard Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
0B0000: MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0C0000: VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVHHHHHHHHHHHHHHHH................
0D0000: ................................................................
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.8
Adaptec AHA-1542CF 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 today. Shared memory was first devised by IBM because they found 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 Upper Memory Area 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 7.9 shows the default memory addresses for the IPL ROM and shared memory of an IBM Token Ring network adapter.

```
. = Empty Addresses
G = Video Graphics Array (VGA) Adapter Graphics Mode Video RAM
M = VGA Monochrome Text Mode Video RAM
C = VGA Color Text Mode Video RAM
V = Standard VGA Video ROM BIOS
I = Token Ring Network Adapter IPL ROM
N = Token Ring Network Adapter Shared RAM
R = Standard Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
0B0000: MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0C0000: VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV................IIIIIIII........
0D0000: ................................NNNNNNNNNNNNNNNN................
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.9
Token Ring network adapter default memory map.

I have also included the standard VGA video BIOS in Figure 7.9 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 7.9 would fit both at its default BIOS address of DC000 as well as 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 7.10 shows an optimum setup with both the Token Ring adapter and the SCSI adapter in the same machine.

```
. = Empty Addresses
G = Video Graphics Array (VGA) Adapter Graphics Mode Video RAM
M = VGA Monochrome Text Mode Video RAM
C = VGA Color Text Mode Video RAM
V = Standard VGA Video ROM BIOS
H = SCSI Host Adapter ROM BIOS
I = Token Ring Network Adapter IPL ROM
N = Token Ring Network Adapter Shared RAM
R = Standard Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0A0000: GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
0B0000: MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0C0000: VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVHHHHHHHHHHHHHHHHHHAAAAAAAAAAAAAAAA
0D0000: IIIIIIII................................................................
      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.10
Adaptec AHA-1542CF SCSI adapter with optimized memory use.

This configuration allows a single 120K Upper Memory Block (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 then 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, which would function but is hardly optimum.

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 (Basic Input-Output System, which is usually stored in a read-only memory chip). 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 like a disk drive. The main functions of the programs stored in the motherboard ROM 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.
- 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.
- The *Basic Input-Output System* (BIOS) is the software interface, or master control program, to all the hardware in the system. With the BIOS, a program easily can access features in the system by calling on a standard BIOS program module instead of talking directly to the device.

Both segments E000 and F000 in the memory map are considered reserved for the motherboard BIOS, but only some AT-type systems actually use this entire area. PC/XT-type systems require only segment F000 and enable adapter card ROM or RAM to use segment E000. Most AT systems use all of F000 for the BIOS, and may decode but not use any of segment E000. By decoding an area, the AT motherboard essentially grabs control of the addresses, which precludes installing any other hardware in this region. In other words, it is not possible to install any other adapters to use this area. That is why 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 Memory Management Unit (MMU) in the processor to map RAM from extended memory into segment E000 as an Upper Memory Block and subsequently use it for loading software. This is a nice solution to what otherwise would be wasted memory. Under DOS, the EMM386 driver controls the MMU remapping functions.

PC/XTSystem BIOS. Many different ROM-interface programs are in the IBM motherboards, but the location of these programs is mostly consistent. The following figures show the ROM BIOS memory use in segments E000 and F000.

Figure 7.11 shows the memory use in an IBM PC and XT with a Type 1 (256K) motherboard.

```
. = Empty Addresses
b = IBM ROM (Cassette) BASIC Interpreter
R = Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: ......................bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbRRRRRRRR
```

Fig. 7.11
Motherboard ROM BIOS memory use in an original IBM PC and XT.

Figure 7.12 shows the memory use in an XT with a 640K motherboard as well as in the PS/2 Model 25 and Model 30. These systems have additional BIOS code compared to the original PC and XT. Note that Cassette BASIC remains in the same addresses.

```
. = Empty Addresses
b = IBM ROM (Cassette) BASIC Interpreter
R = Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbRRRRRRRR
```

Fig. 7.12
Motherboard ROM BIOS memory use in IBM XT (Type 2) and PS/2 Models 25 and 30.

Figure 7.13 shows the motherboard ROM BIOS memory use in most PC- or XT-compatible systems. These systems lack the Cassette BASIC Interpreter found in IBM's BIOS.

```
. = Empty Addresses
R = Standard XT Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: ..............................RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR

In compatible XT type systems, the BIOS length may vary, but 32K is common.
```

Fig. 7.13
Motherboard ROM BIOS memory use in most PC- or XT-compatibles.

AT System BIOS. Figure 7.14 shows the motherboard BIOS use in an IBM AT or XT-286 system.

```
. = Empty Addresses
b = IBM ROM (Cassette) BASIC Interpreter
R = Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.14
Motherboard ROM BIOS memory use in an IBM AT and XT-286.

Figure 7.15 shows the motherboard ROM BIOS memory use of most AT-compatible systems. These systems lack the IBM Cassette BASIC, but do usually include a built-in Setup program.

```
. = Empty Addresses
R = Standard Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: ................................................................
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Fig. 7.15
Motherboard ROM BIOS memory use of most AT-compatible systems.

Note that the standard AT-compatible 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 UMB space.

IBM PS/2 ROM-BIOS. The motherboard BIOS memory use by PS/2 models with a 286 or higher processor, including ISA and MCA systems, is shown in figure 7.16. This shows that PS/2 systems use more of the allocated ROM space for their motherboard ROM BIOS. The extra code contains VGA drivers (PS/2 systems have built-in VGA adapters), and additional code to run the system in protected mode.

```
. = Empty Addresses
b = IBM ROM (Cassette) BASIC Interpreter
R = Motherboard ROM BIOS

      : 0---1---2---3---4---5---6---7---8---9---A---B---C---D---E---F---
0E0000: RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
0F0000: RRRRRRRRRRRRRRRRRRRRRRRRbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbRRRRRRRR
```

Fig. 7.16
IBM PS/2 motherboard ROM BIOS memory map.

Note that some of the newest PS/2 systems no longer have the Cassette BASIC interpreter in ROM. In those systems, additional BIOS code fills that area.

PS/2 Models with 286 and higher processors have additional Advanced BIOS code, to be used when the systems are running protected-mode operating systems such as OS/2. Non-PS/2 systems, that don't have the Advanced BIOS code, can still easily run OS/2, but must load the equivalent of the Advanced BIOS software from disk rather than having it loaded from ROM.

Some compatible systems with a SCSI hard disk and host adapter need a special protected-mode driver for the adapter to work under OS/2 because the on-board BIOS runs only in real mode and not in protected mode. IBM SCSI adapters, however, are unique in that they include both real- and protected-mode BIOS software on-board and operate hard disks under any operating system with no need for drivers. Again, this is not really that much of a bonus or feature because this same type of protected mode operating driver code can simply be loaded from disk (bootup occurs in real mode) during the OS/2 boot process. In fact, many of the Advanced BIOS routines originally in the PS/2 BIOS have been enhanced, so that many of the built-in BIOS routines are being discarded and superseded by loaded routines anyway. The way things have turned out, there is simply no inherent advantage to the Advanced BIOS code in the PS/2 systems.

IBM Cassette Basic. The ROM maps of most IBM compatibles equal the IBM system with which they are compatible—with the exception of the Cassette BASIC portion. This section examines the origins of Cassette BASIC (also called ROM BASIC) and investigates reasons why you would ever see this in a modern system. A variety of related error messages from compatible systems are explored as well.

It may come as a surprise to some personal computer users, but the original IBM PC actually had a jack on the rear of the system for connecting a cassette tape recorder. This was to be used for loading programs and data to or from a cassette tape. At the time the PC was introduced, the most popular personal computer was the Apple II, which also had the cassette tape connection. Tapes were used at the time because floppy drives were very costly, and hard disks were not even an option yet. Floppy drives came down in price quickly at the time, and the cassette port never appeared on any subsequent IBM systems. The cassette port also never appeared on any compatible system.

The original PC came standard with only 16K of memory in the base configuration. No floppy drives were included, so you could not load or save files from disks. Most computer users at the time would either write their own programs in the BASIC (Beginners All-purpose Symbolic Instruction Code) language or run programs written by others. Various versions of the BASIC language were available, but the Microsoft version had become the most popular. Having a Microsoft BASIC interpreter was essential in the early days of personal computing, yet these interpreters would take 32K of memory when loaded. That meant you would need 32K of RAM just for the language interpreter and additional memory for your program and data. To make the system cheaper and to conserve memory, IBM contacted Microsoft and licensed the MS-BASIC interpreter.

They then built the BASIC interpreter directly into the motherboard ROM BIOS, where it occupied the 32K address range F6000-FDFFF. This meant that although the system only came with 16K, you could use all 16K for your own programs and data because no additional memory was required to run the BASIC language interpreter. To save and load programs, this BASIC language was designed to access the cassette port on the back of the system.

If you purchased a floppy drive for your PC system, you would also need DOS (Disk Operating System) to run it. Because the BASIC built into the ROM on the PC motherboard did not have the code required to operate the floppy drive for storing and retrieving files, an extension or overlay to the ROM BASIC interpreter was located on the DOS disk. Called Advanced BASIC, or BASICA.COM, this program on the DOS disk was actually constructed as an overlay to the ROM BASIC, and would not function independently. Because no compatibles ever had ROM BASIC (no compatibles ever had a cassette tape recorder port either), the BASICA extensions found on the IBM DOS disks would not run in any compatible. At that time, there was no such thing as a separately available generic MS-DOS as there is today.

Compaq was one of the first to solve this problem. They went to Microsoft and licensed the DOS separately from IBM, thus producing their own version called Compaq DOS. On the Compaq DOS disks, they included a version of the Microsoft BASIC interpreter called GWBASIC.EXE (Graphics Workstation BASIC) that was complete as a stand-alone program. In other words, it was not constructed as an overlay for the ROM BASIC (which was absent from the Compaq), but was a version complete all by itself. This stand-alone GWBASIC version of the MS-BASIC interpreter would run on any system, IBM or compatible, because it did not depend on the existence of the ROM BASIC, as did IBM's BASICA program. In some strange way, perhaps the reliance of the IBM BASICA on the IBM ROM BASIC was one way to make the IBM PS/2 different from the compatibles. Because any compatible vendor could license the complete BASIC interpreter (and DOS as well) directly from Microsoft, the IBM ROM BASIC became an unimportant feature.

What is really strange is that IBM kept this ROM BASIC and BASICA relationship all the way through most of the PS/2 systems! The 486 PS/2 system I am using right now came standard with a built-in SCSI adapter and a standard 400M SCSI drive. (I have long since outgrown that drive and in fact have just installed a 4G drive in this portable machine!) And yet this system still has the ROM BASIC wasting 32K of space! I liken this to humans having an appendix. The ROM BASIC in the IBM systems is a sort of vestigial organ—a left-over that had some use in prehistoric ancestors, but that has no function today!

You can catch a glimpse of this ROM BASIC on IBM systems that have it by disabling all the disk drives in the system. In that case, with nothing to boot from, most IBM systems unceremoniously dump you into the strange (vintage 1981) ROM BASIC screen. When this occurs, the message looks like this:

```
The IBM Personal Computer Basic
Version C1.10 Copyright IBM Corp 1981
62940 Bytes free
Ok
```

Many people used to dread seeing this because it usually meant that your hard disk had failed to be recognized! Because no compatible systems ever had the Cassette BASIC interpreter in ROM, they had to come up with different messages to display for the same situations in which an IBM system would invoke this BASIC. Compatibles that have an AMI BIOS in fact display a confusing message as follows:

```
NO ROM BASIC - SYSTEM HALTED
```

This message is a BIOS error message that is displayed by the AMI BIOS when the same situations occur that would cause an IBM system to dump into Cassette BASIC, which of course is not present in an AMI BIOS (or any other compatible BIOS for that matter). Other BIOS versions display different messages. For example, under the same circumstances, a Compaq BIOS displays the following:

```
Non-System disk or disk error
replace and strike any key when ready
```

This is somewhat confusing on Compaq's part, because this very same (or similar) error message is contained in the DOS Boot Sector, and would normally be displayed if the system files were missing or corrupted.

In the same situations that you would see Cassette BASIC on an IBM system, a system with an Award BIOS would display the following:

```
DISK BOOT FAILURE, INSERT SYSTEM DISK AND PRESS ENTER
Phoenix BIOS systems will display either:
No boot device available -
strike F1 to retry boot, F2 for setup utility
```

or

```
No boot sector on fixed disk -
strike F1 to retry boot, F2 for setup utility
```

The first or second Phoenix message appears depending on exactly which error actually occurred.

Although the message displayed varies from BIOS to BIOS, the cause is the same for all of them. Two things can generally cause any of these messages to be displayed, and they both relate to specific bytes in the Master Boot Record, which is the first sector of a hard disk at the physical location Cylinder 0, Head 0, Sector 1.

The first problem relates to a disk that has either never been partitioned or has had the Master Boot Sector corrupted. During the boot process, the BIOS checks the last two bytes in the Master Boot Record (first sector of the drive) for a "signature" value of 55AAh. If the last two bytes are not 55AAh, then an Interrupt 18h is invoked, which calls the subroutine that displays the message you received as well as the others indicated or on an IBM system that invokes Cassette (ROM) BASIC itself.

The Master Boot Sector (including the signature bytes) is written to the hard disk by the DOS FDISK program. Immediately after you low-level format a hard disk, all the sectors are initialized with a pattern of bytes, and the first sector does NOT contain the 55AAh signature. In other words, these ROM error messages are exactly what you see if you attempt to boot from a hard disk that has been low-level formatted, but has not yet been partitioned.

Now consider the second situation that can cause these messages, and potentially your other problem causing the same message. If the signature bytes are correct, then the BIOS executes the Master Partition Boot Record code, which performs a test of the Boot Indicator Bytes in each of the four partition table entries. These bytes are at offset 446 (1BEh), 462 (1CEh), 478 (1DEh), and 494 (1EEh) respectively. They are used to indicate which of the four possible partition table entries contain an active (bootable) partition. A value of 80h in any of these byte offsets indicates that table contains the active partition, whereas all other values must be 00h. If more than one of these bytes is 80h (indicating multiple active partitions), or any of the byte values is anything other than 80h or 00h, then you see the following error message:

```
Invalid partition table
```

If all these four Boot Indicator Bytes are 00h, indicating no active (bootable) partitions, then you also see Cassette BASIC on an IBM system or the other messages indicated earlier depending on which BIOS you have. This is exactly what occurs if you were to remove the existing partitions from a drive using FDISK, but had not created new partitions on the drive, or had failed to make one of the partitions Active (bootable) with FDISK before rebooting your system.

In the latest PS/2 systems, IBM has finally done away with the ROM BASIC once and for all. DOS versions through 5.x from IBM included the BASICA interpreter, however with the introduction of DOS 6, IBM eliminated BASICA because the newer systems did not have the ROM portion anyway. Microsoft included GWBASIC in MS-DOS versions up to and including 4.x and eliminated it in later versions. All DOS 5 and higher versions (IBM and MS) have a special crippled version of the Microsoft QuickBASIC Compiler now included rather than GWBASIC or BASICA. The compiler is crippled so that you can compile programs in memory, but cannot create EXE files on disk. The run-time compiler executes virtually all the older interpreted BASIC programs, so newer systems have no need for the old GWBASIC or BASICA interpreters. If you want the full version of QuickBASIC that will compile a program and save it as an EXE file, you must purchase the full version of QuickBASIC.

ROM Versions. Over the years, the BIOS in various PC models has undergone changes almost always associated with either a new system or a new motherboard design for an existing system. There are several reasons for these changes. The introduction of the XT, for example, gave IBM a good opportunity to correct a few things in the system BIOS and also add necessary new features, such as automatic support for a hard disk. IBM retrofitted many of the same changes into the PC's BIOS at the same time.

Because an in-depth knowledge of the types of BIOS is something a programmer might find useful, IBM makes this information available in the technical-reference manuals sold for each system. A new ROM BIOS technical-reference manual covers all IBM systems in one book. Complete ROM BIOS listings (with comments) accompanied early IBM system documentation, but that information is not supplied for later IBMs.

Even if you aren't a programmer, however, certain things about system BIOS are important to know. IBM has had many different BIOS versions for the PC and PS/2 families. Sometimes a system has different versions of BIOS over the course of a system's availability. For example, at least three versions of BIOS exist for each of the PC, XT, and AT systems. Because a few important changes were made in the software stored in BIOS, knowing which BIOS is in your system can be useful.

To determine which ROM BIOS module is installed in your system, first check your system documentation. If you have a 386 or later system, a few lines of text at the top of your screen during the POST probably identifies the manufacturer, BIOS revision number, and date of manufacture. The BIOS version you have also may be indicated by the manufacturer's name, version number, and date encoded in the chip.

On IBM systems, the ROM BIOS contains an ID byte (the second-to-last byte). The value of the byte at memory location FFFFE (hexadecimal) corresponds to the system type or model. Table 7.4 shows information about the different ROM BIOS versions that have appeared in various IBM systems.

Table 7.4 IBM BIOS Model, Submodel, and Revision Codes

System Type	CPU Speed	Clock	Bus Type/ Width
PC	8088	4.77 MHz	ISA/8
PC	8088	4.77 MHz	ISA/8
PC	8088	4.77 MHz	ISA/8
PC-XT	8088	4.77 MHz	ISA/8
PC-XT	8088	4.77 MHz	ISA/8
PC-XT	8088	4.77 MHz	ISA/8
PC*jr*	8088	4.77 MHz	ISA/8
PC Convertible	8088	4.77 MHz	ISA/8
PS/2 25	8086	8 MHz	ISA/8
PS/2 30	8086	8 MHz	ISA/8
PS/2 30	8086	8 MHz	ISA/8
PS/2 30	8086	8 MHz	ISA/8
PC-AT	286	6 MHz	ISA/16
PC-AT	286	6 MHz	ISA/16
PC-AT	286	8 MHz	ISA/16
PC-XT 286	286	6 MHz	ISA/16
PS/1	286	10 MHz	ISA/16
PS/2 25 286	286	10 MHz	ISA/16
PS/2 30 286	286	10 MHz	ISA/16
PS/2 30 286	286	10 MHz	ISA/16
PS/2 35 SX	386SX	20 MHz	ISA/16
PS/2 35 SX	386SX	20 MHz	ISA/16
PS/2 40 SX	386SX	20 MHz	ISA/16
PS/2 40 SX	386SX	20 MHz	ISA/16
PS/2 L40 SX	386SX	20 MHz	ISA/16
PS/2 50	286	10 MHz	MCA/16
PS/2 50	286	10 MHz	MCA/16
PS/2 50Z	286	10 MHz	MCA/16
PS/2 50Z	286	10 MHz	MCA/16
PS/2 55 SX	386SX	16 MHz	MCA/16
PS/2 55 LS	386SX	16 MHz	MCA/16
PS/2 57 SX	386SX	20 MHz	MCA/16
PS/2 60	286	10 MHz	MCA/16
PS/2 65 SX	386SX	16 MHz	MCA/16
PS/2 70 386	386DX	16 MHz	MCA/32
PS/2 70 386	386DX	16 MHz	MCA/32
PS/2 70 386	386DX	16 MHz	MCA/32
PS/2 70 386	386DX	20 MHz	MCA/32

ROM BIOS Date	ID Byte	Submodel Byte	Rev.	ST506 Drive Types
04/24/81	FF	—	—	—
10/19/81	FF	—	—	—
10/27/82	FF	—	—	—
11/08/82	FE	—	—	—
01/10/86	FB	00	01	—
05/09/86	FB	00	02	—
06/01/83	FD	—	—	—
09/13/85	F9	00	00	—
06/26/87	FA	01	00	26
09/02/86	FA	00	00	26
12/12/86	FA	00	01	26
02/05/87	FA	00	02	26
01/10/84	FC	—	—	15
06/10/85	FC	00	01	23
11/15/85	FC	01	00	23
04/21/86	FC	02	00	24
12/01/89	FC	0B	00	44
06/28/89	FC	09	02	37
08/25/88	FC	09	00	37
06/28/89	FC	09	02	37
03/15/91	F8	19	05	37
04/04/91	F8	19	06	37
03/15/91	F8	19	05	37
04/04/91	F8	19	06	37
02/27/91	F8	23	02	37
02/13/87	FC	04	00	32
05/09/87	FC	04	01	32
01/28/88	FC	04	02	33
04/18/88	FC	04	03	33
11/02/88	F8	0C	00	33
?	F8	1E	00	33
07/03/91	F8	26	02	None
02/13/87	FC	05	00	32
02/08/90	F8	1C	00	33
01/29/88	F8	09	00	33
04/11/88	F8	09	02	33
12/15/89	F8	09	04	33
01/29/88	F8	04	00	33

(continues)

Table 7.4 Continued

System Type	CPU Speed	Clock	Bus Type/ Width
PS/2 70 386	386DX	20 MHz	MCA/32
PS/2 70 386	386DX	20 MHz	MCA/32
PS/2 70 386	386DX	25 MHz	MCA/32
PS/2 70 386	386DX	25 MHz	MCA/32
PS/2 70 486	486DX	25 MHz	MCA/32
PS/2 70 486	486DX	25 MHz	MCA/32
PS/2 P70 386	386DX	16 MHz	MCA/32
PS/2 P70 386	386DX	20 MHz	MCA/32
PS/2 P75 486	486DX	33 MHz	MCA/32
PS/2 80 386	386DX	16 MHz	MCA/32
PS/2 80 386	386DX	20 MHz	MCA/32
PS/2 80 386	386DX	25 MHz	MCA/32
PS/2 90 XP 486	486SX	20 MHz	MCA/32
PS/2 90 XP 486	487SX	20 MHz	MCA/32
PS/2 90 XP 486	486DX	25 MHz	MCA/32
PS/2 90 XP 486	486DX	33 MHz	MCA/32
PS/2 90 XP 486	486DX	50 MHz	MCA/32
PS/2 95 XP 486	486SX	20 MHz	MCA/32
PS/2 95 XP 486	487SX	20 MHz	MCA/32
PS/2 95 XP 486	486DX	25 MHz	MCA/32
PS/2 95 XP 486	486DX	33 MHz	MCA/32
PS/2 95 XP 486	486DX	50 MHz	MCA/32

The ID byte, Submodel byte, and Revision numbers are in hexadecimal.
— = This feature is not supported.
None = Only SCSI drives are supported.
? = Information is unavailable.

The date that the BIOS module design was completed is important: It has the same meaning as a version number for software. (IBM later began to code an official revision number, as indicated in the last column of table 7.4) You can display the date by entering the following four-statement BASIC program:

```
10 DEF SEG=&HF000
20 For X=&HFFF5 to &HFFFF
30 Print Chr$(Peek(X));
40 Next
```

An easier way to display the date is to use the DOS DEBUG program. First run DEBUG at the DOS prompt by entering the following command:

```
DEBUG
```

ROM BIOS Date	ID Byte	Submodel Byte	Rev.	ST506 Drive Types
04/11/88	F8	04	02	33
12/15/89	F8	04	04	33
06/08/88	F8	0D	00	33
02/20/89	F8	0D	01	33
12/01/89	F8	0D	?	?
09/29/89	F8	1B	00	?
?	F8	50	00	?
01/18/89	F8	0B	00	33
10/05/90	F8	52	00	33
03/30/87	F8	00	00	32
10/07/87	F8	01	00	32
11/21/89	F8	80	01	?
?	F8	2D	00	?
?	F8	2F	00	?
?	F8	11	00	?
?	F8	13	00	?
?	F8	2B	00	?
?	F8	2C	00	?
?	F8	2E	00	?
?	F8	14	00	?
?	F8	16	00	?
?	F8	2A	00	?

Then at the debug "-" prompt, enter the following command to display the ROM BIOS date:

```
D FFFF:5 L 8
```

If you are interested in a ROM upgrade for your system, you can contact the motherboard or even the BIOS manufacturer, such as Phoenix, Award, or AMI. Occasionally, the BIOS may be updated to fix problems or add support for new peripherals.

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 1 megabyte 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, or Pentium system, the extended memory limit is 4G (4,096M).

For an AT system to address memory beyond the first megabyte, the processor must be in *protected mode*—the native mode of these newer 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 1 megabyte pieces (each its own real-mode session) and for several of these sessions to be running simultaneously in protected areas of memory. 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, which is termed *multitasking*, requires software that can manage each program, keeping them from crashing into one another. OS/2 does this now, and the new Windows 4 will allow this as well.

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-, or Pentium-based system, is really functioning as little more than a turbo PC. The 286 can emulate the 8086 or 8088, but it cannot operate in protected mode at the same time.

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 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. In other words, HIMEM.SYS must be loaded for Windows to function.

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 and recent versions of DOS, including DOS 6. Other memory managers, like QEMM, also convert extended memory into XMS-specification memory when you add their device drivers to CONFIG.SYS.

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 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 High Memory Area 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 to the 386, 486, and Pentium.

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 1st 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 a 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 that 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 small 64K window established in the Upper Memory Area (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.

Segment D000 in the first megabyte usually is used for mapping. Lotus, Intel, and Microsoft—founders of the LIM specification for expanded memory (LIM EMS)—decided to use this segment because it is largely unused by most adapters. Programs must be written specially to take advantage of this segment-swapping scheme, and then only data normally can be placed in this segment because it is above the area of contiguous memory (640K) that DOS can use. For example, a program cannot run while it is swapped out and therefore not visible by the processor. This type of memory generally is useful only in systems that do not have extended (processor-addressable) memory available to them.

The figure 7.17 shows how expanded memory fits with conventional and extended memory.

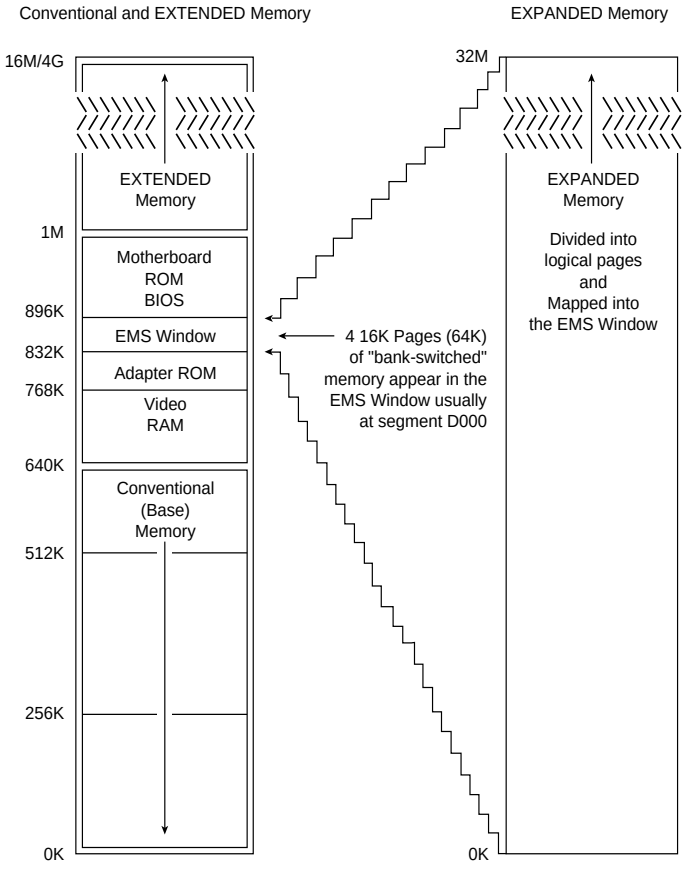

Fig. 7.17
Conventional, extended, and expanded memory.

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 sold widely 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, have 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 as well as 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 Memory Management Unit (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 being 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.

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.

ROM Shadowing

Computers based on the 386 or higher CPU chip, which provide 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 on-board 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. Because 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 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. The shadowing is accomplished by using the powerful Memory Management Unit (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 at exactly the same addresses where 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 farther and offer you the capability to enable or disable shadowing in (usually 16K) increments throughout the remainder of the C000 and D000 segments.

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 (that 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, then 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.

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 4M of RAM (which is 4,096K) installed show only 3,712K installed during the POST or when running Setup. This showing indicates that 4,096K – 3,712K = 384K of missing memory! Some systems may show 3,968K with the same 4M installed, which works out to 4,096K – 3,968K = 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 total shown during the POST. In most systems with 4,096K (4M), you have 640K base and 3072K extended. In some systems, Setup reports 640K base and 3328K 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. I currently take six to nine hours of class time in my seminars to fully explore, explain, and exploit memory, but this explanation must be more brief! Say that you have a 486 system with two installed 72-pin (36-bit) 1M SIMMs. This setup results in a total installed memory of 2M in two separate banks because the processor has a 32-bit data bus, and one parity bit is required for every eight data bits. Each SIMM is a single bank in this system. Note that most cheaper 486 systems use the 30-pin (9-bit) SIMMs of which four are required to make a single bank. The first bank (or SIMM in this case) starts at address 000000 (the first Meg), and the second starts at 100000 (the second Meg).

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, as shown in the following list:

- 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) and 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 that you can only shadow ROM, never RAM, so if any card (such as a network card for example) has a RAM buffer in the C8000-DFFFF area, you must not shadow the RAM buffer addresses or the card does 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. In my example system with 2M, no remapping would result in 640K of base memory and 1,024K of extended memory, for a total of 1,664K of usable RAM—384K short of the total (2,048K – 384K).

More-advanced 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 that PS/2 systems have the video BIOS integrated with the motherboard BIOS in E0000-FFFFF, so no separate video BIOS exists to shadow as compared to other systems. In my example system with two 1M 36-bit SIMMs, the 256K not used for shadowing would be remapped to 200000-23FFFF, which is the start of the third megabyte. This affects diagnostics because if you had any memory error reported in those addresses (200000-23FFFF), it would indicate a failure in the FIRST SIMM, even though the addresses point to the end of installed extended memory. The addresses from 100000-1FFFFF would be in the second SIMM, and the 640K base memory 000000-09FFFF 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 to 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 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 then 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 that 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 (9-bit) SIMMs or one 72-pin (36-bit) SIMM.

Adapter Memory Configuration and Optimization

Ideally, adapter boards would be simple "plug-and-play" devices that require you to merely plug the adapter into a motherboard slot and then use it. However, sometimes it almost seems that adapter boards (except on EISA and MCA systems) are designed as if they were the only adapter likely to be present on a system. Rather than adapter boards being plug-and-play, they usually require you to know the upper memory addresses and IRQ and DMA channels already on your system and then 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 them if they do occur. In addition, these sections discuss moving adapter memory to resolve conflicts and provide some ideas on optimizing adapter memory use.

Adding adapters to EISA and MCA systems is somewhat more simple because these system architectures feature *auto-configure adapter boards*. In other words, EISA and MCA systems work with adapters to determine available upper memory addresses, IRQs, and DMA channels and automatically configure all adapters to work optimally together.

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 and DOS 6 or higher versions. You also can download MSD from the Microsoft BBS (see the vendor list for the number). This utility examines your system configuration and determines not only the upper memory used by your adapters, but also the IRQs used by each of these adapters. Many other utilities can accomplish the same task, but most people already have a copy of MSD—whether they know it or not!

After you run MSD 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. 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.

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 also 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, which is, of course, preferable to an overlap, but still is not the best use of upper memory.

Users who want to make the most of their upper memory might consider studying the documentation for each adapter board installed on their 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 look 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 upper memory area.

Taking Advantage of Unused Upper Memory. On systems based on the 386 or higher CPU chip, memory-resident programs and device drivers can be moved into the upper memory area by using a memory manager like the DOS 6.x MEMMAKER utility or Quarterdeck's QEMM. 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 and QEMM 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, and then run either MEMMAKER from the DOS prompt or use the installation program on the QEMM diskette. Both programs install required device drivers in your CONFIG.SYS file, and then begin optimizing your memory configuration. Both do an outstanding job of freeing up conventional memory, although QEMM can free more conventional memory automatically than most other utilities. With careful fine-tuning, an individual using only the raw DOS HIMEM.SYS and EMM386.EXE drivers can perform feats of memory management that no automatic program can do!

The following sections cover using memory management software to optimize conventional memory, as well as 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.

Using HIMEM.SYS (DOS). The DOS device driver HIMEM.SYS, which has been included with Windows 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.

The XMS extended memory specification was developed by Microsoft, Intel, AST Corp., and Lotus Development in 1987 and specifies how programs can use memory beyond the first megabyte on systems based on the 286 CPU chip or higher. The XMS specification also allows real-mode programs (those designed to run in DOS) to use extended memory in several different ways.

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 Upper Memory Area (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 *Using MS-DOS 6* or your DOS manual.

MS-DOS 6.x MEMMAKER. You can increase the amount of conventional memory available to software applications 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 Upper Memory Area so that DOS can load memory-resident programs and drivers into the UMA. Unfortunately, this required an extensive knowledge of the upper memory configuration of a particular system, as well as trial and error to see what programs can 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 and 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. 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 high memory area (HMA)
- Maps free XMS memory into unused regions in the Upper Memory Area (UMA) as *Upper Memory Blocks* (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 as well as to remap the keys on your keyboard. If you are primarily a Windows user and do not spend much time at the DOS prompt, then you can eliminate ANSI.SYS from your CONFIG.SYS file to free up the memory the driver is using.

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, exit from any other programs you are running; start your network or any memory-resident programs and device drivers you absolutely need; and at the DOS prompt, type the following:

MEMMAKER

The MEMMAKER setup runs in two modes—Express and Custom. Express setup is preferable for those 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 and 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.EXE 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 like 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.

For detailed information on using the MEMMAKER utility, consult your DOS 6 manual or Que's *Using MS-DOS 6.2,* Special Edition. If you have MS-DOS, you can get help on MEMMAKER by typing **HELP MEMMAKER** at the DOS prompt.

IBM-DOS 6.x RAMBoost. The IBM-DOS 6.x RAMBoost utility, licensed from Central Point, which supplies some of the DOS 6 utilities, works much like MEMMAKER to free up additional conventional memory. After you make backup copies of CONFIG.SYS and

AUTOEXEC.BAT and strip down these files to only what you need to load, enter the following at the DOS prompt:

RAMSETUP

RAMBoost calculates the best way to load your memory-resident programs and device drivers into UMBs. RAMBoost gives results roughly equivalent to the MS-DOS 6 MEMMAKER utility. On many systems, it frees up 620K of conventional memory.

Third-Party Memory Managers. Although MEMMAKER and RAMBoost do a good job of freeing-up conventional memory on most systems, memory management utilities like Quarterdeck's QEMM, and Qualitas' 386MAX can do a better job on many systems with more complex configurations, and therefore, numerous memory-resident programs and device drivers. The following sections provide information about QEMM and 386MAX.

Quarterdeck QEMM. One of the strengths of QEMM is how simple it is to install and use. Before running the QEMM INSTALL program, 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 exit any program you are running. At the DOS prompt, log in to the drive where the QEMM install diskette is located and run the INSTALL program. QEMM copies its files to the C:\QEMM directory (or another directory if you want).

Then the INSTALL program loads the Optimize utility, which calculates the upper memory needed for your memory-resident programs and device drivers and determines the proper region of upper memory for each. During this process, your system is rebooted several times (or when prompted, you may have to turn off your system and then restart it). When Optimize is finished, you can type **MEM** at the DOS prompt to find out how much free conventional memory your system has.

After QEMM is installed and running on your system, each time you add a memory-resident program or device driver, or any time you add or remove an adapter board (which might change the configuration of upper memory), you need to again run OPTIMIZE. For additional information on installing and running QEMM, and for troubleshooting help, consult your QEMM user manual.

One of the best features of QEMM is that it comes with a system configuration diagnostic utility called MANIFEST. This program is much like MSD, but offers more information and detail in many areas.

Qualitas 386MAX. Before running the 386MAX INSTALL program, make a backup copy of your CONFIG.SYS and AUTOEXEC.BAT files in case you need them later to restore your system. Then exit any program you are running, log in to the drive with the 386MAX install diskette, and run INSTALL. 386MAX copies its files to the C:\386MAX subdirectory (or any directory you choose). Then the INSTALL program loads the MAXIMIZE utility, which determines where in upper memory to place each memory-resident program and device driver.

When MAXIMIZE finishes, you can type **MEM** at the DOS prompt to find out how much free conventional memory your system has.

As with MEMMAKER and QEMM, 386MAX uses codes to specify which region of upper memory each memory-resident program and device driver is loaded into. Whenever you add a new device driver or memory-resident program to your system, or when you add or remove an adapter, you must again run MAXIMIZE.

Physical Memory

The CPU and motherboard architecture dictates a computer's physical memory capacity. The 8088 and 8086, with 20 address lines, can use as much as 1M (1,024K) 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, and Pentium CPUs have a full set of 32 address lines; they can keep track of a staggering 4 gigabytes of memory.

When the 286, 386, 486, and Pentium chips emulate the 8088 chip (as they do when running a single DOS program), 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, 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.

Pentium-based systems can address as much as 4 gigabytes of memory. To put these memory-addressing capabilities into perspective, 4 gigabytes (4,096M) of memory costing the going rate of about $30 per megabyte for fast (60 nanoseconds or less) RAM chips would total $122,880!! Of course, you could probably negotiate a much better price with a chip vendor if you planned to buy 4 gigabytes of SIMMs! Even if you could afford all this memory, the largest SIMMs available today are 72-pin versions with 64M capacity. Most Pentium motherboards have only four to eight SIMM sockets, which allows a maximum of 256M to 512M if all four or eight sockets are filled. Not all systems accept all SIMMs, so you might have a limitation of less than this amount for many Pentium and 486 systems.

On many systems, accessing RAM chips installed directly on a motherboard is faster than accessing memory through an expansion slot. Even without considering this speed advantage, you have the advantage of saving slots. The more memory chips you can get on the motherboard, the fewer adapter slots you need to use. A system that does not have a memory expansion slot faces a large reduction in speed if you use a memory expansion board made for a standard 16-bit slot.

Some 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 and you have more than 16M of memory, you have the potential for problems because the ISA bus only allows DMA access to 16M. Attempted transfers beyond 16M cause the system to crash. Most modern 486 motherboards enable you to install a maximum of 64M on the motherboard using four 16M SIMMs.

Because the PC hardware design reserves the top 384K of the first megabyte of system memory for use by the system itself, you have access to 640K for your programs and data. The use of 384K by the system results in the 640K conventional memory limit. The amount of conventional memory you can actually use for programs depends on the memory used by device drivers (such as ANSI.SYS) and memory-resident programs (such as MOUSE.COM) you load in your CONFIG.SYS and AUTOEXEC.BAT files. Device drivers and memory-resident programs usually use conventional memory.

RAM Chips

A *RAM chip* temporarily stores programs when they are running and 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 instructs DOS to bring an EXE or COM disk file into RAM, and computer programs reside in RAM as long as they are running. The CPU executes programmed instructions in RAM. RAM stores your keystrokes when you use a word processor. RAM stores numbers used in calculations. The CPU also stores results in RAM. Telling a program to save your data instructs the program to store RAM contents on your hard drive as a file.

If you decide to purchase more RAM, you need the information on RAM chips and their speeds presented in the following sections to help ensure that you don't slow down your computer when you add memory.

Physical Storage and Organization

RAM chips can be physically integrated into the motherboard or adapter board in several forms. Older systems used individual memory chips, called *Dual In-line Pin* (DIP) chips, that were plugged into sockets or soldered directly to a board. Most modern systems use a memory package called a *Single In-line Memory Module* (SIMM). These modules combine several chips on a small circuit board plugged into a retaining socket. A SIPP, or Single In-line Pin Package, is similar to a SIMM, but it uses pins rather than an edge connector to connect to the motherboard. 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.

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 is a list of available RAM chips and their capacities:

16K by 1 bit	These devices, used in the original IBM PC with a Type 1 motherboard, have a small capacity compared with the current standard. You won't find much demand for these chips except from owners of original IBM PC systems.
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, such as the popular vintage AST 6-pack boards, use these chips also.
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 on top of one another and soldered together. True 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, as well as most compatible systems, used these chips.
1M by 1 bit (or 256K by 4 bits)	These 1M chips are very popular in systems today because of their low cost. These chips are most often used in SIMMs because of their ease of installation and service. (See the section entitled "Single In-Line Memory Modules (SIMMs)" for more information.)
4M by 1 bit (or 1M by 4 bits)	Four-megabit chips have gained popularity recently and now are used in many compatible motherboards and memory cards. 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)	16-megabit chips are gaining popularity in Pentium-based systems in which as much as 256 megabytes can be installed on some systems. The use of these chips in SIMMs allow for 16M or larger capacities for a single SIMM.
64M by 1 bit (or 16M by 4 bits)	64-megabit chips are the most recent on the market. These chips allow enormous SIMM capacities of 64M or larger! Because of the high expense and limited availability of these chips, you see them only in the most expensive systems.

Figure 7.18 shows a typical memory chip. Each marking on the chip is significant.

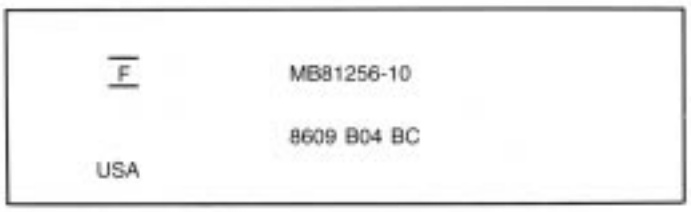

Fig. 7.18
The markings on a typical memory chip.

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 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, in the IBM XT 286, which is an AT-type 16-bit system, the last 128K bytes of memory on the motherboard consist of a bank with only six chips; four are 64K-by-4 bits wide, and two parity chips are 1 bit wide, storing 18 bits.

In figure 7.18, 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 if you can tell who that is, or perhaps a memory chip vendor.

Memory Banks

Memory chips (DIPs, SIMMs, and SIPPs) 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 7.5 shows the widths of individual banks based on the type of PC:

Table 7.5 Memory Bank Widths on Different Systems

Processor	Data Bus	Bank Size (w/Parity)	30-Pin SIMMs per Bank	72-Pin SIMMs per Bank
8088	8-bit	9-bits	1	1 (4 banks)
8086	16-bit	18-bits	2	1 (2 banks)
286	16-bit	18-bits	2	1 (2 banks)
386SX, SL, SLC	16-bit	18-bits	2	1 (2 banks)
386DX	32-bit	36-bits	4	1
486SLC, SLC2	16-bit	18-bits	2	1 (2 banks)
486SX, DX, DX2, DX4	32-bit	36-bits	4	1
Pentium	64-bit	72-bits	8	2

The number of bits for each bank can be made up of single chips or SIMMs. For example, in a 286 system that would use an 18-bit bank, you could make up a bank of 18 individual 1-bit-wide chips, or you could use four individual 4-bit-wide chips to make up the data bits, and two individual 1-bit-wide chips for the parity bits. Most modern systems do not use chips, but instead use SIMMs. If the system has an 18-bit bank, then it likely would use 30-pin SIMMs and have two SIMMs per bank. All the SIMMs in a single bank must be the same size and type. As you can see, the 30-pin SIMMs are less than ideal for 32-bit systems because you must use them in increments of four per bank! Because these SIMMs are available in 1M and 4M capacities today, this means that a single bank has to be 4M or 16M of memory, with no in-between amounts. Using 30-pin SIMMs in 32-bit systems artificially constricts memory configurations and such systems are not recommended. If a 32-bit system uses 72-pin SIMMs, then each SIMM represents a separate bank, and the SIMMs can be added or removed on an individual basis rather than in groups of four. This makes memory configuration much easier and more flexible.

Older systems often used individual chips. For example, the IBM PC Type 2 and XT Type 1 motherboard contains four banks of memory labeled Bank 0, 1, 2, and 3. Each bank uses nine 64K-by-1-bit chips. The total number of chips present is 4 times 9, or 36 chips, organized as shown in figure 7.19.

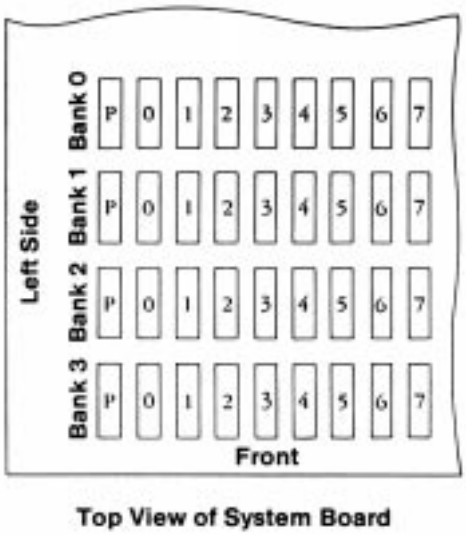

Fig. 7.19
A memory bank on a PC Type 2 or XT Type 1 motherboard.

This layout is used in many older 8-bit motherboards, including the Type 1 and 2 PC motherboards and the Type 1 and 2 XT motherboards. Most PC or XT clones also followed this scheme. Note that the parity chip is the leftmost chip in each bank on the XT motherboard.

The physical orientation used on a motherboard or memory card 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.

Parity Checking

One standard IBM has 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. Some modern SIMMs have only three chips, however, with each chip handling three of the nine bits.

IBM 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 special chip called a *74LS280 parity generator/checker* on the motherboard (or memory card) 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 chip 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
Parity 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
Parity 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 yyyyy (z)*

For the IBM AT and late model XT: PARITY CHECK *x yyyyy*

Where *x* is 1 or 2:

1 = Error occurred on the motherboard

2 = Error occurred in an expansion slot

yyyy represents a number from 00000 through FFFFF 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. (An NMI is a system warning that software cannot ignore.)

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.

Note that 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 recurs, 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) = (xxxxx)
```

or

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

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 xxxxx
```

or

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

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 if misused. 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!

IBM PS/2 systems have a slightly different way of communicating parity-check errors than the older IBM systems. To indicate motherboard parity errors, the message looks like this:

```
110

xxxxx
```

To indicate parity errors from an expansion slot, the message looks like this:

```
111

xxxxx
```

In these messages, the *xxxxx* indicates the address of the parity error. As with most IBM systems, the system is halted after these messages are displayed, thus eliminating any possibility of saving work.

Single In-Line Memory Modules (SIMMs)

For memory storage, most modern systems have adopted the single in-line memory module (SIMM) 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, so removing and replacing individual memory chips is impossible. Instead, you must replace the entire SIMM if any part of it fails. The SIMM is treated as though it were one large memory chip.

IBM compatibles have two main physical types of SIMMs—30-pin (9 bits) and 72-pin (36 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. SIMMs are available both with and without parity bits. Until recently, all IBM-compatible systems used parity-checked memory to ensure accuracy. Other non-IBM-compatible systems like the Apple Macintosh have never used parity-checked memory. For example, Apple computers use the same 30-pin or 72-pin SIMMs as IBM systems, but Apple computers as a rule do not have parity-checking circuitry, so they can use slightly cheaper 30-pin SIMMs that are only eight bits wide instead of nine bits (eight data bits plus one parity bit) as is required on most IBM-compatible systems. They also can use 72-pin SIMMs that are only 32-bits wide rather than 36-bits (32 data bits plus 4 parity bits) as is required on most IBM compatibles. You can use the parity SIMMs in Apple systems; they will simply ignore the extra bits. If you use non-parity SIMMs in an IBM compatible that requires parity-checked memory, you instantly get memory errors, and the system cannot operate. If you service both IBM and Apple systems, you could simply stock only parity SIMMs because they can be used in either system.

Recently, a disturbing trend has developed in the IBM compatible marketplace. Some of the larger vendors have been shipping systems with parity checking disabled! These systems can use slightly cheaper non-parity SIMMs like the Apple systems. The savings amounts to about $10 per 4M SIMM, which can result in a savings to the manufacturer of about $20 for a typical 8M configuration. Because most buyers have no idea that the parity checking has been taken away (how often have you seen an ad that says “now

featuring NO PARITY CHECKING"?), the manufacturer can sell its system that much cheaper. Because several of the big names have started selling systems without parity, most of the others have been forced to follow to remain price competitive. Because nobody wants to announce this information, it has remained as a sort of dirty little secret within the industry! What is amazing is that the 386 and higher processors all contain the parity circuitry within them, so no additional circuits are needed on the motherboard. It is solely the cost of the parity chips on the SIMMs that is being saved.

How can they do this? Well most newer motherboards have a method by which parity checking can be disabled to accommodate non-parity SIMMs. Most older motherboards absolutely required parity SIMMs because there was no way to disable the parity checking. Some newer motherboards have a jumper to enable or disable the parity circuitry. Some include this as a SETUP option, and some systems check the memory for parity bits and if they are not detected in all banks, parity checking is automatically disabled. In these systems, installing a single non-parity SIMM normally causes parity checking to be disabled for all memory. I have often found parity checking to be the first alert to system problems, so I am not thrilled that virtually all newer systems come with non-parity SIMMs. Fortunately, this can be rectified by specifying parity SIMMs when you order a new machine. If you don't specify parity SIMMs, then surely you will get the cheaper non-parity versions. Also make sure that your motherboard has parity checking enabled as well, because most are not coming configured with it enabled.

Figures 7.20 and 7.21 show typical 30-pin (9-bit) and 72-pin (36-bit) SIMMs, respectively. The pins are numbered from left to right and are connected through to both sides of the module. Note that all dimensions are in both inches and millimeters (in parentheses).

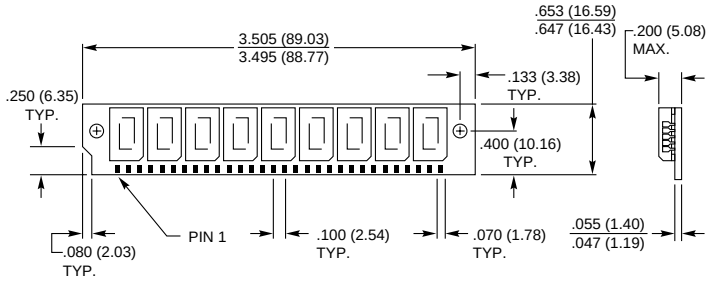

Fig. 7.20
A typical 30-pin (9-bit) SIMM.

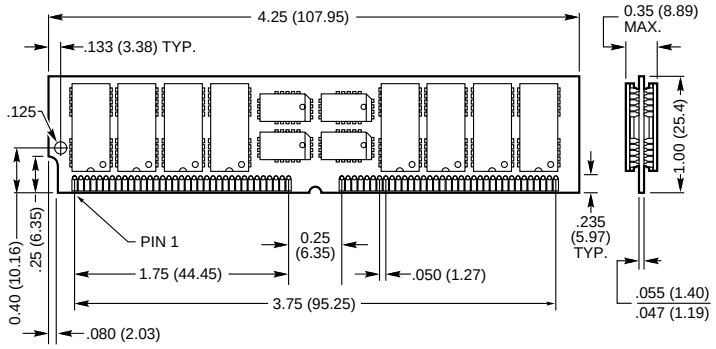

Fig. 7.21
A typical 72-pin (36-bit) SIMM.

A SIMM is extremely compact, considering the amount of memory it holds. SIMMs are available in several capacities, depending on the version. Table 7.6 lists the different capacities available for both the 30-pin and 72-pin SIMMs.

Table 7.6 SIMM Capacities

Capacity	Parity SIMM	Non-Parity SIMM
30-Pin SIMM Capacities		
256K	256K × 9	256K × 8
1M	1M × 9	1M × 8
4M	4M × 9	4M × 8
16M	16M × 9	16M × 8
72-Pin SIMM Capacities		
1M	256K × 36	256K × 32
2M	512K × 36	512K × 32
4M	1M × 36	1M × 32
8M	2M × 36	2M × 32
16M	4M × 36	4M × 32
32M	8M × 36	8M × 32
64M	16M × 36	16M × 32

Dynamic RAM SIMMs of each type and capacity are available in different speed ratings. These ratings are expressed in nanoseconds (billionths of a second, abbreviated ns). 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. 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 can generally purchase SIMMs rated at 80ns, 70ns, or 60ns. 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, then 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 or faster than what the system requires. Because often very little price difference exists between the different speed versions, I usually buy faster SIMMs than are needed for a particular application, as this may make them more usable in a future system that may require the faster speed.

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 Models 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 identify to 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 itself acts as if it were a single chip of 9-bits wide (with parity), and it really does not matter how this total is derived. Older SIMMs were constructed with nine individual 1-bit-wide chips to make up the total, whereas many newer SIMMs use two 4-bit-wide chips and one 1-bit-wide chip for parity, making a total of three chips on the SIMM. Accessing the 3-chip SIMMs can require adjustments to the refresh timing circuits on the motherboard, and many older motherboards could not cope. Most newer motherboards automatically handle the slightly different refresh timing of both the 3-chip or 9-chip SIMMs, and in this case the 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 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 9-chip versions should the 3-chip versions not work in an older system, it seems prudent to recommend sticking with the 9-chip variety in any older systems.

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 like 486 machines because they comprise an entire bank of memory (32 data bits plus 4 parity bits). When you configure a system that uses a 72-pin SIMM, you can usually add or remove memory in single SIMM modules (except on systems that use interleaved memory schemes to reduce wait states). The 30-pin SIMMs are clumsy when used in a system with a 32-bit memory architecture because these SIMMs must be added or removed in quantities of four to make up a complete bank. A 386SX or a 286 system would require only two 9-bit SIMMs for a single bank of memory so the 30-pin SIMMs are a better match.

Remember that some 486 systems (such as the PS/2 90 and 95 systems) use interleaved memory to reduce wait states. This requires a multiple of two 36-bit SIMMs because interleaved memory accesses are alternated between the SIMMs to improve performance.

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, then 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. For example, the IBM PS/2 Model 70-Axx and Bxx systems accept 72-pin SIMMs of 1M or 2M capacity, which are 80ns or faster. Although an 80ns 4M SIMM is available, it does not work in these systems. The PS/2 Model 55 SX and 65 SX, however, accept 1M, 2M, or 4M 72-pin SIMMs. 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.

SIMMs were designed to eliminate chip creep, which plagues systems with memory chips installed in sockets. *Chip creep* occurs when a chip works its way out of its socket, caused by the normal thermal expansion and contraction from powering a system on and off. Eventually, chip creep leads to poor contact between the chip leads and the socket, and memory errors and problems begin.

The original solution for chip creep was to solder all the memory chips to the printed circuit board. This approach, however, was impractical. Memory chips fail more frequently than most other types of chips and soldering chips to the board made the units difficult to service.

The SIMM incorporates the best compromise between socketed and soldered chips. The chips are soldered to the SIMM, but you can replace the socketed SIMM module easily. In addition, the SIMM is held tight to the motherboard by a locking mechanism that does not work loose from contraction and expansion, but is easy for you to loosen. This solution is a good one, but it can increase repair costs. You must replace what amounts in some cases to an entire bank rather than one defective chip.

For example, if you have a 486DX4 with one 60ns 8M SIMM that goes bad, you could replace it for about $250 or less from chip suppliers who advertise in the computer magazines. This is certainly more expensive than replacing a single 256K chip costing about $2 each. Of course, 8M SIMMs are used on systems never designed for single chips. It would take 288 256K chips to equal the memory storage of one 8M SIMM. Troubleshooting a problem with a single SIMM device is much easier than troubleshooting 288 discrete chips. In addition, one SIMM is more reliable than 288 individual chips!

All systems on the market today use SIMMs. Even Apple Macintosh systems use SIMMs. The SIMM is not a proprietary memory system but rather 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 7.7 and 7.8 show the interface connector pinouts for both 30-pin SIMM varieties, as well as 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 its modified 30-pin configuration.

Table 7.7 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 7.8 Standard 72-Pin SIMM Pinout

Pin	SIMM Signal Name
1	Ground
2	Data Bit 0
3	Data Bit 16
4	Data Bit 1
5	Data Bit 17
6	Data Bit 2
7	Data Bit 18
8	Data Bit 3
9	Data Bit 18
10	+5 Vdc
11	Column Address Strobe Parity
12	Address Bit 0
13	Address Bit 1
14	Address Bit 2
15	Address Bit 3
16	Address Bit 4
17	Address Bit 5
18	Address Bit 6
19	Reserved
20	Data Bit 4
21	Data Bit 20
22	Data Bit 5
23	Data Bit 21
24	Data Bit 6
25	Data Bit 22
26	Data Bit 7
27	Data Bit 23
28	Address Bit 7
29	Block Select 0
30	+5 Vdc
31	Address Bit 8
32	Address Bit 9
33	Row Address Strobe 3
34	Row Address Strobe 2
35	Parity Data Bit 2
36	Parity Data Bit 0
37	Parity Data Bit 1
38	Parity Data Bit 3
39	Ground

(continues)

Table 7.8 Continued

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

Note that the 72-pin SIMMs employ a set of four 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. This is very 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. Unfortunately, unlike the film standards, the presence detect

signaling is not a standard throughout the PC industry. Different system manufacturers sometimes use different configurations for what is expected on these 4 pins. Table 7.9 shows how IBM defines these pins.

Table 7.9 72-Pin SIMM Presence Detect Pins

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

N/C = No Connection (open)
Gnd = Ground
Pin 67 = Presence detect bit 0
Pin 68 = Presence detect bit 1
Pin 69 = Presence detect bit 2
Pin 70 = Presence detect bit 3

RAM Chip Speed

Memory-chip speed is reported in nanoseconds (ns). (One *nanosecond* is the time that light takes to travel 11.72 inches.) 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. Faster RAM chips may improve your system performance also when the motherboard was designed for faster chips than the manufacturer installed. For example, if your motherboard was designed for 70ns chips, but the manufacturer installed cheaper 80ns RAM, you may get a slight performance boost from removing all the old RAM chips and installing faster ones. However, do not mix chip speeds by replacing only some 80ns chips with 70ns ones.

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.

Testing Memory

The best way to test memory is to install and use it, with your PC system acting as the testing tool. Numerous diagnostic programs are available for testing memory. Many advanced diagnostic programs are discussed in Chapter 20 "Software and Hardware Diagnostic Tools." Many of these programs, such as the Norton Utilities NDIAGS program, are very inexpensive and yet offer very complete memory diagnostic capabilities. One word of advice is that all memory testing should be done on a system booted from a plain DOS disk with no memory managers or other resident programs loaded.

Parity Checking

As mentioned previously, the memory chips handle eight bits per character of data plus an extra bit called the *parity bit*. Memory-control circuitry uses the parity bit to cross-check the integrity of each byte of data. When the circuitry detects an error, the computer stops and displays a message informing you of the malfunction. Parity checking is the first line of defense for memory and other system errors. Note that many newer systems are coming with non-parity SIMMs to save money. This eliminates parity checking, and increases the likelihood that errors go undetected.

Power-On Self Test

The Power-On Self Test (POST), which is in ROM, can be an effective test for problem memory. When you turn on your system, the POST checks the major hardware components including memory. If the POST detects a bad memory chip, it displays a warning. A more sophisticated disk-based program, however, usually does a better job.

Advanced Diagnostic Tests

A number of diagnostic programs can be used to test RAM chips and other system components. For example, Norton Utilities includes a utility called NDIAGS that tests RAM chips for defects. Other utility packages that can be used to test RAM chips are discussed in Chapter 20 "Software and Hardware Diagnostic Tools." For IBM computers, the Advanced Diagnostics Disk contains utilities that can be used to test your system RAM.

Such programs should be used any time you receive a memory parity error message, or a memory error in the POST (Power-On Self Test). Even if you do not receive error messages, if a properly running system suddenly begins locking up or if strange characters appear on-screen, you should run a good diagnostic program. Software diagnostics can help you spot trouble before a hardware problem destroys data.

Summary

This chapter discussed memory from both a physical and a logical point of view. The types of chips and SIMMs that physically comprise the memory in a PC system were discussed, and the logical arrangement of this memory was examined. The terms used to describe the different regions and the purpose for each region were covered. The chapter also looked at ways of reorganizing the system memory and taking advantage of unused areas.