AUGUST 1992

NOVELL® RESEARCH

Managing Memory in a DOS Workstation: Part 1

Edward A. Liebing
Senior Technical Editor
Systems Engineering Division

Ken Nefl
Principal Technical Writer
Systems Engineering Division

Many DOS workstation users are still bumping up against the 640KB memory barrier. This limitation becomes especially critical as more and more network drivers and memory-resident programs are loaded on the PC, taking up valuable application space in DOS's conventional memory. This AppNote is the first in a two-part series that explains what types of memory are available in a DOS workstation, how applications use memory, and how to use various memory managers to maximize the amount of memory available for running applications.

Contents

Introduction

Despite the recent upsurge in the use of advanced desktop operating systems such as DESQview, Windows, and OS/2, DOS is still prevalent in today's workplace. Many NetWare users still work within the confines of DOS's 640KB memory barrier. One of the most challenging aspects of maintaining a DOS workstation is managing this limited memory. After users load NetWare's LAN drivers, the NetWare shell, and their favorite terminate-and-stay-resident (TSR) programs, they often find that there's not enough memory left to run applications.

This AppNote is the first in a two-part series that covers some of the solutions to the problem of RAM cram in DOS machines used as network workstations. It explains the various types of memory that are available to DOS: conventional memory, high DOS memory, and the high memory area. It covers the difference between expanded and extended memory, and describes how applications work with conventional, expanded, and extended memory. This AppNote ends with a quick look at Novell's expanded and extended memory shells (EMSNETX.EXE and XMSNETX.EXE).

The second AppNote in the series will build on this foundation and explain how to use memory manager software to move NetWare drivers and TSRs out of conventional memory and into high memory. With these memory managers, users can free up as much as 631KB of the lower 640KB of memory for running DOS applications, and almost as much when running Windows.

Memory in DOS-Based Computers

DOS is designed to work within a 1,024KB (1MB) home field of memory. The top 384KB of this megabyte of memory is reserved for use by video display adapters, I/O device drivers, and other peripherals. That leaves 640KB of RAM in which to run programs.

The terminology used to identify these divisions of memory varies. The 640KB section is most often designated as conventional or low DOS memory. The 384KB of RAM above it is called "high DOS" memory, system memory, or upper memory. Microsoft uses the term "upper memory blocks" (UMBs) to describe contiguous ranges of memory within this area. (Some sources refer to the entire first 1,024KB of RAM as conventional memory, divided into low DOS and high DOS.)

Another small area of memory that DOS can access (on a machine with an 80286 or better microprocessor and more than 1MB of memory) is the 64KB area just above the 1,024KB boundary. This area is known as the "high memory area" or HMA.

In this AppNote, we'll refer to the first 640KB of RAM as conventional memory, the next 384KB of RAM as high DOS memory, and the 64KB above the 1MB boundary as the high memory area (HMA). Figure 1 shows these basic areas of memory.

Figure 1: Conventional memory, high DOS memory, and the high memory area (HMA).

A Word About Memory Addresses

One confusing aspect of any discussion of PC memory management is the memory addressing scheme imposed by the microprocessor. In a PC, every byte of memory has a physical address that the microprocessor uses to locate it. DOS applications access code and data in memory by using the addresses supplied by the microprocessor.

The original IBM PC was based on Intel's 8088 microprocessor, which has a 20-bit memory address bus and can therefore directly access a maximum of 220 or 1,048,576 bytes (1MB) of RAM. Rather than starting at location 0 and counting up to 1,048,576, Intel came up with a two-step segment:offset addressing scheme. The segment address specifies a 16-byte chunk (paragraph) of RAM; the offset address identifies a specific byte within that paragraph.

The CPU finds a particular byte in memory by using two storage locations, called registers, within the microprocessor. One register contains the starting segment value and the other contains the byte offset value. The maximum value that can be stored in each register is 65,535 (FFFF in hexadecimal notation). The CPU calculates a physical address by taking the address in the segment register, shifting it one character to the left, and adding the two registers together to obtain the physical address (see Figure 2).

Figure 2: Procedure for calculating an address in memory.

The notation for representing memory addresses can be perplexing. Sometimes you'll see both segment and offset values separated by a colon, as in FFFF:000F for the sixteenth byte in memory segment FFFF. This can also be represented as the effective address 0FFFFFh (the h after a number means hexadecimal). When speaking just of 16-byte paragraph ranges, the offset value is often omitted.

The 1,024KB of DOS memory is divided into 16 segments of 64KB each (16 þ 64KB = 1,024KB). Conventional memory consists of the ten segments ranging from 0000h to 9FFFh (bytes 0 to 655,167). High DOS memory consists of the six segments ranging from A000h to FFFFh.

Conventional Memory

Conventional memory is the memory available to DOS for running programs and applications, as well as for temporarily storing the data created by these applications. Once the data is created or manipulated in memory, it can then be saved to a network disk or to a local storage device, such as a floppy diskette or a hard disk drive. Most DOS users become all too familiar with this 640KB memory area, especially when they start getting Program too big to fit in memory errors.

Why is conventional memory limited to 640KB? The main two reasons are the design of the original IBM PC and the high cost of memory at the time it was introduced. In designing DOS for the 8088-based PC, Microsoft and IBM looked at the older CP/M systems, which typically ran just fine with 64KB of RAM, and decided that ten times that amount would be more memory than could ever be used. Besides, according to one IBM presenter, the maximum memory configuration offered for the original IBM PC line was 128KB, and that configuration carried a hefty price tag of over $10,000. So the initial designers of DOS set the upper limit for running applications at 640KB and reserved the remaining 384KB of memory for system hardware use.

Besides being used to load applications and their data, conventional memory is used for loading DOS itself and by terminate-and-stay-resident programs (see Figure 3).

Figure 3: Conventional memory is used for loading DOS, TSRs, and applications.

TSRs are programs that load themselves into memory and stay resident even when other applications are running. One example of a TSR is DOSEDIT.COM, which saves in memory the commands the user types

at the DOS prompt. The user can then recall previously typed commands and modify them, if necessary, to avoid a lot of tedious retyping of complex commands.

NetWare's communication drivers (IPX.COM, or LSL.COM, network board driver, and IPXODI.COM) and the NetWare shell (NETX.COM) are also memory resident programs. Each of these programs requires a certain amount of conventional memory to keep itself resident. Loading TSRs along with the NetWare drivers can take a large bite out of available conventional memory, thereby constraining the largest DOS application the PC can run.

The main objective of PC memory management is to maximize the amount of conventional memory that is available, so users can run memory-hungry applications.

High DOS Memory

The high DOS memory area is used by video display adapters for screen refresh buffers, and by the system ROM BIOS (Basic Input/Output System) and various hardware peripherals to talk to DOS. Many network adapters also use RAM in this area of memory.

High DOS memory is divided into six 64KB segments. Memory within these segments can be doled out in multiples of 4KB: for example, from B000 to B0FF is 4KB of memory, and from B000 to B1FF is 8KB. (The size of these memory segment pieces will become more important when we talk about memory managers in Part 2.)

Portions of these upper memory segments are used for different system purposes. Figure 4 shows an overall map of the high DOS memory area and the system drivers that use it.

Figure 4: The high DOS memory area is used by video display adapters, I/O device drivers, and system ROM BIOS.

Memory managers can make good use of the area from C000 to DFFF because it usually offers the most free upper memory. However, the amount of unused memory is entirely dependent on your computer's configuration.

The following sections explain some of these high DOS memory users in more detail.

Video Display Adapters. Figure 5 shows the sections of high DOS memory that various types of video display adapters use in text and graphics modes. This information is useful when trying to figure out which upper memory blocks a particular type of video monitor uses.

Figure 5: High DOS memory segments used by common video display adapters.

Note that CGA, EGA, and VGA adapters running in low resolution graphics mode use the same RAM addresses as they do when running in text mode. So if a user has an EGA or VGA monitor but uses only applications that support low resolution graphics, the memory section from A000 to AFFF won't be used. If applications run in EGA/VGA (high resolution) graphics mode, A000 to AFFF are included as part of the adapter's memory configuration.

Many VGA and Super VGA adapters come with 512KB to 1,024KB of memory on the adapter itself. This on-board memory offloads the high DOS memory used and speeds up image refresh to the screen.

Network Adapters. RAM in segments C000 to DFFF is available for use by network adapter drivers. (This portion of memory is also used for page frames with an Expanded Memory System (EMS) driver, as we'll explain later.) Both Arcnet and Token-Ring adapters use some memory from these block segments.

Arcnet adapters use 16KB of memory. The RAM address settings available for Novell's RX-Net II driver include:

C000 -C3FF (setting 3)
C400 -C7FF
C800 -CBFF
CC00 -CFFF (setting 1)
D000 -D3FF (setting 0 - default)
D400 -D7FF
D800 -DBFF
DC00 -DFFF (setting 2)
E000 -E3FF

To figure out which range of addresses a particular RX-Net II board is using, you need to know what the board's configuration setting is. For a dedicated IPX workstation, RX-Net defaults to D000 as configuration setting 0, CC00 as setting 1, DC00 as setting 2, and C000 as setting 3. (The other RAM addresses are available to DOS ODI and OS/2 workstations, and for boards that can be configured through Novell's JUMPERS utility. See Novell's Installation Supplements for more information.)

Token-Ring adapters use a 4KB block for ROM and a 16KB block for RAM. The configuration options for the IBM Token-Ring PC Adapter/A microchannel adapter include:

	ROM Addresses	RAM Addresses
Primary	CC00 - CDFF	D800 - DBFF
Alternate	CE00 - CFFF	D400 - D7FF
Alternate	C400 - C5FF	C000 - C3FF
Alternate	D000 - D1FF	D800 - DBFF
Primary	D200 - D3FF	DC00 - DFFF

The second alternate listing is nice because it places both RAM and ROM addresses between C000 and C5FF. However, some VGA or Super VGA monitors might already be using C400 to C7FF. You can end up with memory problems if you have two hardware peripherals trying to share the same memory segment. Check the Token-Ring adapter manufacturer's documentation for the exact ROM and RAM addresses used by other board settings.

Token-Ring adapters can fragment addressable upper memory, which can be a problem if you have applications that run in expanded memory and need a large EMS page frame (see the explanation of expanded memory later in this AppNote).

System ROM BIOS. Normally, the E000 through F000 range is reserved for the system ROM BIOS. BIOS is a series of software instructions (routines) that allow DOS to access the keyboard, monitor, attached printers, floppy and hard disk drives, CD-ROMs, and so forth. BIOS resides in ROM (Read Only Memory), which is why it is called ROM BIOS.

Early IBM computers used the E000 area for their ROM BIOS and reserved F600 to FFFF for ROM BASIC to run BASIC programs. Later, the ROM BIOS region was moved up to F000. In IBM PS/2s, the range from E000 through FFFF is reserved for ROM BIOS and VGA ROM.

Regardless of the system you have, chances are there are many holes in the high DOS memory area that can be made available to memory managers. This available memory can amount to up to 266KB of additional memory for placing TSRs out of conventional memory.

High Memory Area (HMA)

The high memory area (HMA) is the first 64KB of extended memory, located just above the 1MB address boundary (1024KB to 1088KB). Many 80286- and 80386-based computers contain more than 1MB of memoryþoften 2MB, 4MB, 8MB, or more. By default, this memory is configured as extended memory. DOS

can't normally use extended memory. But with an XMS (eXtended Memory Specification) memory manager, the HMA can be used by DOS applications without having to switch the microprocessor into protected mode (a more advanced mode of memory addressing used in 80286 and later microprocessors).

Technically, the size of the HMA is 65,520 bytes (64KB minus 16 bytes). DOS can address this piece of extended memory in real mode. MS-DOS's XMS-compatible memory manager called HIMEM.SYS makes this area available to DOS programs.

HMA access is possible because of the segment:offset addressing scheme we discussed earlier. The last memory address in high DOS memory is the sixteenth byte in segment FFFF, or FFFF:000F. But the offset register is capable of holding values up to FFFFh. This makes it possible to reference addresses in the range FFFF:0010 through FFFF:FFFF.

Address references in this range generate a carry bit when the 16-bit offset value (0FFFFh) is added to the 20-bit shifted segment value (FFFF0h). The 8088 has no address carry bit, so the processor simply wraps around to address 0000:0000 after FFFF:000F. On an 80286 or later microprocessor, there is an address carry bit called A20. If the system activates this bit while in 8088 (real) mode, the wraparound does not occur, and the high memory area becomes available.

To access HMA, computers must enable Address Line 20 (A20). AT- and MCA-class computers remain compatible with certain 8086/8088 programs by disabling A20. Disabling A20 allows for address wrap-around at the 1MB mark for those programs that employ address wrap-around, as explained earlier.

Programs that are placed in HMA must be well behaved enough to disable the A20 line when they are not in use and enable A20 when they are. A number of applications (generally CP/M programs converted for use with DOS) rely on the 1MB memory wrap routine and will overwrite HMA if the A20 line is enabled at the time. Otherwise, you won't be able to run those programs along with HMA, or vice versa.

Only one program at a time can control A20, so only one program can run in HMA. For this reason, any program or utility placed in HMA must use as much of the 64KB space as possible. Programs or utilities that don't use 90% or more of this space should be allowed to be placed into conventional memory so that another program can access HMA more efficiently. This gives you greater control over what you place in HMA.

Note: MS-DOS v5.0 does not disable A20 when loaded high.

Expanded Memory

Expanded memory is memory that initially resided on a separate memory expansion card. It is essentially invisible to the microprocessor because it is not part of the PC's normal memory addressing range. To use this memory, the computer needed an Expanded Memory Specification (EMS) memory board and device driver to physically map blocks of expanded memory into pages located in the high DOS memory area. EMS defines various system-level calls through which applications can allocate, use, and release blocks of expanded memory.

LIM/EMS 3.2

The need for expanded memory arose when large spreadsheets and database programs began to stretch the 640KB limit of DOS on an 8088-based PC. To provide additional memory to spreadsheets, Lotus, Intel, and Microsoft collaborated to define the LIM Expanded Memory Specification board. The first go-round was LIM version 3.2, released in 1985. This quickly became a widely recognized standard, and several application developers (including Lotus, Microsoft, Borland, and Autodesk) wrote EMS-compatible versions of their major applications.

To access memory on the LIM board, the Expanded Memory Manager (EMM), which is the device driver portion of the EMS system, creates a 64KB window or page frame in high DOS memory. The 64KB page frame is divided into four 16KB pages of memory that must be contiguous within upper memory (see Figure

6).

Figure 6: Expanded memory originally required a contiguous 64KB page frame in high DOS memory.

The EMS page frame maps to the expanded memory on the LIM board. The LIM board uses 16KB physical page frames in upper memory to access its expanded memory banks. Each page frame can access multiple physical pages by first mapping the address of the frame to its expanded memory location and then swapping in the pages (in 16KB pages) as needed.

LIM 3.2 allowed access to up to 8MB of expanded memory. However, the expanded memory could only be used for data storage, not for running applications.

LIM/EMS 4.0

In 1987, the EMS standard was revised to version 4.0. This new specification incorporated some improvements that had been added to LIM 3.2 by AST, Quadram, and Ashton-Tate in their Enhanced Expanded Memory Specification (EEMS). Under LIM 4.0, the number of 16KB blocks in the page frame was increased from 4 to 64. In addition, the blocks in the page frame itself no longer had to be contiguous. LIM 4.0 also allows for backfilling, the process of filling in conventional memory starting at 640KB on down to 256KB and advising DOS of its presence. The new version also allows up to 32MB of expanded memory.

LIM/EMS 4.0 also provides multitasking capabilities to let more than one application run at the same time. Two examples of programs that take advantage of this are Windows 3.1 and DESQview 386.

Caveats for Using Expanded Memory

For all its benefits, LIM/EMS is not an ideal solution to the RAM cram problem. Even with LIM 4.0, many EMS applications won't use expanded memory unless they can find 64KB of contiguous memory. Finding a 64KB block of upper memory can be tedious, and you may have to experiment with network adapter and video adapter address settings until you end up with an available 64KB memory block. (We'll get into this more in Part 2.)

With 8086/8088 and most 80286 computers, you need both a LIM memory board and an Expanded Memory Manager (EMM) to use expanded memory applications. With 80386 and 80486 computers, memory managers such as DOS 5.0's EMM386.SYS can convert the computer's extended memory into expanded memory. EMM386 allows you to run expanded memory programs without the need for a LIM board.

If you don't have an 80386/486 computer, install only LIM boards that meet the LIM EMS specification you need (3.2 or 4.0); also, be sure the program you want to run works with the LIM specification you choose. The application's documentation should tell you which LIM specification it supports.

Since other devices such as video adapters and network interface cards also use this area in high memory, keep track of which boards you have installed and the memory address areas they occupy. If you experience an address conflict with any installed boards, those boards will not work and will often hang the computer when two programs access the same memory address at the same time.

Another potential problem occurs when using an expanded memory manager such as EMM386. Some network interface cards don't reserve their address space until they are being used. The Expanded Memory Manager may not know the area in upper memory has already been reserved, so it reserves that area too. Network interface cards will overwrite other boards and software using conflicting addresses, which can also cause the application to hang.

Programs like System Sleuth from Dariana Inc. (Cypress, CA) or the MSD diagnostics program that ships with Windows 3.1 can show you if you have any high memory conflicts. Also, the DR-DOS v6.0 MEM /A and MS-DOS v5.0 MEM /P commands can show you a lot about how memory is mapped. However, these

commands don't show you if there are memory conflicts.

Extended Memory

Extended memory is directly accessible memory that begins at the 1,024KB boundary and extends up to 16MB for 80286 computers, and up to 4GB for 80386 computers (see Figure 7).

Figure 7: Extended memory is memory above the 1MB boundary.

Since only 80286 and later microprocessors can directly access extended memory, that is what most 80286, 80386, and 80486 computers come with. Extended memory must be contiguous-there can be no holes in the memory extension. Many computers nowadays come with 640KB of conventional memory and either 384KB or 1,408KB of extended memory, giving them a total of 1MB or 2MB of RAM.

While DOS applications can only address 640KB, other programs (such as Windows v3.x and DESQview 286 and 386) extend DOS itself to take advantage of a workstation's extended memory. These programs use an eXtended Memory Manager (XMM) that conforms to the Lotus/Intel/Microsoft and AST extended memory specification (XMS) v2.0. A number of expanded memory managers (including the EMM386.SYS driver that comes with DR-DOS 6.0) also conform to XMS specifications, which allows you to select extended memory, expanded memory, or both, when calling on a memory manager program.

Adding Extended Memory

When adding memory to a workstation, be sure the memory board begins its addressing at the point where the workstation ends its memory (not including DOS's high memory). For example, whether the workstation has 512KB, 640KB, or 1,024KB (1MB) of RAM, be sure the memory board begins at the proper memory address.

Depending on their hardware scheme, some computers include high memory in their computations and begin their extended memory at 1,408KB (1,024 + 384). For example, Novell 286A and 386A machines came with 1MB of memory and begin their extended memory expansion at 1,408. When buying additional memory, first determine where the computer begins extended memory addressing and make sure the memory board you want to purchase can begin at that address.

Some memory boards are hard-coded to begin at a certain memory address; if that memory address does not match the workstation, you cannot use it. Most memory boards nowadays have configurable memory that you can set through hardware jumpers or through software.

Uses for Extended Memory

When you install an XMS driver, applications can allocate extended memory by making calls through the XMS driver. Extended memory becomes a large memory pool and uses a dynamic allocation scheme for data. This allows the extended memory manager to release and reallocate memory as applications warrant.

Extended memory managers can address extended memory two ways: from the top down and from the bottom up. Those that take the top-down approach normally allocate a contiguous block that starts at the top of extended memory and work down from there. Those that use the bottom up-approach usually can't change memory allocation once the size is designated. The allocation will stay that size until the computer is rebooted.

RAM Disks. One example of extended memory usage is a RAM disk. Computers with extended memory can dedicate some of that memory for a virtual disk that simulates a hard disk drive. Because memory access is much faster than hard disk access, a RAM disk is much faster than hard disk drives.

A typical RAM disk begins at the 1MB address (unless you are using HMA, then it's a little higher) and grows

upward. If you make no designation otherwise, a RAM disk can take all available extended memory. You can then use the RAM disk for application storage to improve the application's data access and retrieval time.

For example, you can place your most frequently-used programs and utilities, as well as COMMAND.COM, in a RAM disk. If you do place COMMAND.COM in the RAM disk, be sure to set up your COMSPEC command to include the directory path to the RAM disk drive letter. Also, place the RAM disk drive letter at the front of the PATH statement so you can access the applications and utilities from the RAM drive before the computer searches through the physical locations.

Since a RAM disk drive is memory, its contents are susceptible to power glitches in any form. For this reason, it's best to place only applications and utilities in the RAM drive and leave data files on local or network directories. This way, you're not saving updated files to memory but to a physical location.

In DR-DOS 6.0, you create a RAM disk by placing a DEVICE command in the CONFIG.SYS file, similar to the following:

```
DEVICE = C:\DRDOS\VDISK.SYS 1024 /E
```

In this example, the RAM disk is 1MB (1,024 bytes) in size and is using extended memory (/E).

RAM disks take the next available drive letter after the disk drive specifications the computer has. For example, if you have a local C drive, the RAM disk takes drive letter D; if you have a local D drive, the RAM disk takes drive letter E, and so on.

DOS Extender Applications. Some applications are linked and compiled with a mini-protected mode, which allows them to take over all extended memory for their use. Besides using extended memory for data, DOS extender applications can execute themselves in extended memory as well.

An example of a DOS extender application is Lotus 1-2-3 v3.0 and greater. These versions of Lotus 1-2-3 run on DOS, yet take control of all extended memory for the program's use.

Expanded and Extended Memory Shells

If expanded or extended memory is installed on a DOS workstation, one way to conserve precious conventional memory is to use Novell's expanded or extended memory shells. These shells store most of the shell code and data in expanded or extended memory, thereby reducing the amount of conventional memory NetWare requires.

The expanded memory shell (EMSNETX.EXE) loads most of the shell code into expanded memory. Version 3.26 of this shell leaves only about 9KB in conventional memory to handle interrupts and some data. Using this shell saves approximately 33KB of conventional memory space (see Figure 8).

To use EMSNETX.EXE, the user must first load a LIM/EMS 4.0-compatible expanded memory manager. (One is usually included with the expanded memory board.) If you use MS-DOS 5.0's EMM386 memory manager, for example, you need to add the /RAM switch to the EMS386 line in the CONFIG.SYS file.

Figure 8: Novell's expanded memory shell (EMSNETX.EXE) loads most of its code into expanded memory.

NetWare's extended memory shell (XMSNETX.EXE) moves most of the shell out of conventional memory and puts it into the high memory area just above the 1MB boundary. With version 3.26, about 8.5KB of the shell must remain in conventional memory to handle interrupts and some data, but using this shell frees up 34KB of conventional memory (see Figure 9).

Figure 9: Novell's extended memory shell (XMSNETX.EXE) loads most of its code into extended memory.

To use the extended memory shell, an extended memory manager (such as Microsoft's HIMEM.SYS) must be loaded at the workstation. The shell is written to support XMS v2.0 or compatible memory managers.

These enhanced shells are designed to help with memory usage problems. While they use significantly less conventional memory, both perform slower than the standard NETX.COM shell. The performance hit will vary from machine to machine. On the average, a 5-10% degradation can be expected. Users will have to weigh the benefits of increased conventional memory against any performance degradation that they may experience.

Conclusion

This AppNote has introduced DOS workstation memory theory. In Part 2, we'll take a look at how to use memory managers to make more conventional memory available for applications. Specifically, we'll look at DOS v5.0 and DR DOS v6.0, as well as the memory management features included in Windows v3.1. We'll also look at Qualitas' 386Max, Quarterdeck's QEMM 386, and V Communications' Memory Commander.