PCjr Ins and Outs

LeRoy Tabb, Jr.
Philadelphia Area Computing Society

A frequent question about the PC*jr* is "What is the difference between a PC*jr* and its full-grown big brother?"

The differences can be listed briefly. The PC*jr* does not have direct memory access (DMA); its video RAM is not located in memory block B0000 or B8000 hex; it cannot accept an 8087 Math Coprocessor; it has two slots for ROM cartridges; and it has only one disk drive and no expansion slots (but can be upgraded by using sidecars). Also, it has a larger allocation for video RAM and uses the TI SN76469N sound generator chip. Got it? If not, join the crowd, but I'll try to shed some light on all of this.

The PC*jr* does not have DMA. This means that, unlike its big brothers, the PC*jr* does not have a DMA controller (the 8237A) to allow the diskette drive to interact directly with memory. Therefore, diskette input/output must be handled by the 8088 CPU through a process called bit nibbling. The most apparent result of this difference is that you can't type ahead while the disk drive is being addressed. The most important result is probably that the CPU has to suspend running our programs while it writes to or reads from the diskette drive, so our PC*jr* is a bit slower than the rest of the family. The fact that the PC*jr* doesn't use DMA may also be a point of potential incompatibility with some software written exclusively for the PC (but that's a rare problem).

In my mind, the most important difference between the PC*jr* and the PC is the location of video memory. Before we go on, however, let's look at how the PC*jr* and the PC use their memory. To understand this, let's digress one step further and begin by talking about the brain and heart of the IBM PC, XT and PC*jr*—the 8088 microprocessor.

The 8088 is a 16-bit processor, and through some trickery called segmented addressing, is capable of addressing up to 1,024K bytes, or one megabyte, of memory. But not all of that memory is available for our use. To see exactly how IBM uses that memory, let's pull out a map. (See Figure 1.)

Figure 1 is not a traditional map, but a map of how the 8088's memory capacity is used by IBM computers. Each line in the memory map (00000 through F0000) represents the starting hexadecimal address for a 64K block of memory. Each block of memory has a specific application. Blocks 00000 thru

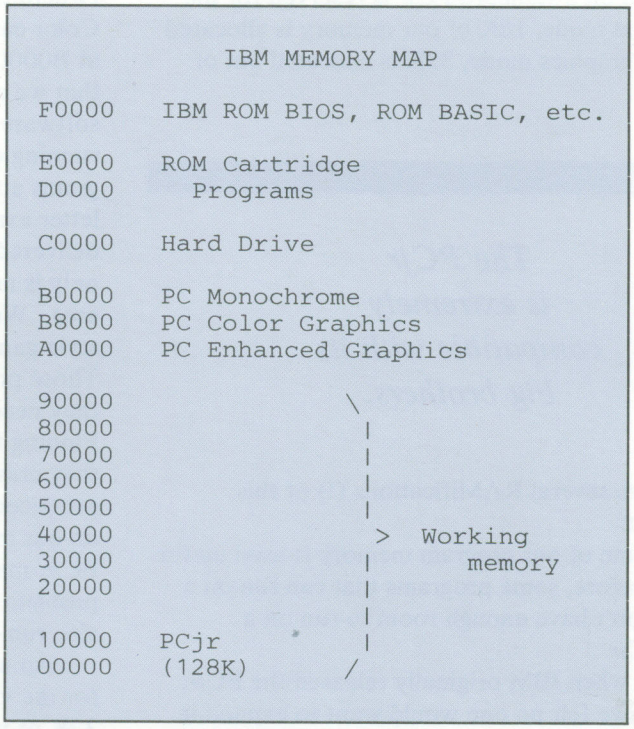


Figure 1. IBM Memory Map

90000 are used for working memory. That's what we mean when we speak of a computer's memory. These blocks represent the memory available for our programs; they add up to the 640K maximum amount of memory in the PC, XT, and PC*jr*. Above that, block A0000 is used for IBM's enhanced video options, the Enhanced Graphics Adapter and the Professional Graphics Adapter. Continuing to climb the map, we find block B0000 which, as I mentioned earlier, is used for the video on the PC and XT. Our PC*jr* does not use that block of memory. Still climbing, block C0000 is used for the hard disk controller on the XT.

The next two blocks (D0000 and E0000) are of particular interest to PC*jr* users, because they are addresses into which we plug our ROM cartridges. This area is unused by the rest of the IBM family. Finally, we come to the top of memory, block F0000, where all of the built-in ROM resides. That includes ROM BASIC (remember, Cartridge BASIC would be in either block D0000 or E0000); the built-in diagnostics; the ROM BIOS; and the ever-popular Keyboard Adventure! This top part of memory holds a lot of interesting things, but they will have to wait to be explained in a future article.

We've seen that block B0000 is not used by the PC*jr*. Well, if our video isn't stored there, where *is* it stored? As you might guess, the PC*jr* allocates a part of our normal RAM memory for video. The highest

part of RAM on a standard PC*jr* is reserved for the video. In text mode, 16K of our memory is allocated to video; in graphics mode, 32K is allocated out of RAM.

The PCjr is extremely compatible with its big brothers.

There are several RAMifications (!) of this arrangement:

First, some of our program memory is used up for video. Therefore, some programs that can run on a 128K PC don't have enough room to run on a standard PC*jr*.

Second, when IBM originally released the PC*jr*, they must have felt no one would want to expand it beyond 128K. So they placed the video RAM at the top of the 10000 block of memory. That caused a separation between the first 128K of RAM and the rest of the working memory (20000-90000), and that meant our programs wouldn't know where to find additional memory, even if we did find a way to install it. Well, as we all know, ways were found to install additional RAM and to get around the gap in our working memory. Getting around that gap is the primary purpose of the CONFIG.SYS files used with IBM's expanded memory sidecar.

The third problem associated with the location of the PC*jr*'s video RAM is that it is physically located at an address different from the one expected by PC software. (This is, by far, the biggest source of incompatibility between the PC*jr* and the PC.) In other words, if a program goes to an address in block B0000 on our PC*jr*, it would find no one home. It would be like trying to send a letter to an address that doesn't exist. Fortunately, the post office can be instructed to forward such mail to an alternate address. In the case of the PC*jr*, that post office is called a Video Gate Array (VGA). The VGA traps any instructions aimed at the B8000 memory area (where the PC Color/Graphics card would be; the Color card is at B8000H and the Monochrome card is at B0000H) and redirects them to the area of memory that we are actually using. This all works fine, and software that plays by the rules will have no problem running on our PC*jr*s. The problems come when programs attempt to circumvent the system. Using the letter example, suppose you are in a hurry to have it delivered, so you deliver it personally. Surprise! Not only is no one home, there isn't even a house in the area. Well, some software programs, typically arcade-style games that rely on speed, behave this way. Those programs make direct calls to the addresses they need, thereby bypassing our VGA postman. Finding themselves somewhere in the netherworld, they crash. Almost all the arcade-style games produced before the relaese of the PC*jr* will not run on our machine. Fortunately, after the release of the PC*jr*, many software developers recognized the problem and wrote versions of software that would also run on the PC*jr*.

An advantage of the PC*jr*'s use of working RAM for the video memory is that it allows us to allocate 32K of video RAM rather that the 16K available on the PC. This gives the PC*jr* a better color graphics capability than the standard PC.

Another of PC*jr*'s advantages is that it uses the TI SN76469N chip for generating sound. This allows the PC*jr* to play music in three-part harmony. The PC*jr* also can be made to synthesize human speech!

Don't panic after reading all of this! In spite of its differences, the PC*jr* is extremely compatible with its big brothers. After I expanded my PC*jr* to 640K, I found minimal compatibility problems. As previously mentioned, incompatibilities came from older arcade-style games (the newer ones are better anyway) and from large-scale business-oriented software that insists on using lots of memory and two disk drives. If you add a second drive, you'll reduce these problems too.

I've touched on a few of the most important functional differences between the PC*jr* and its bigger brothers. There are other differences (the keyboard, for instance). If you're interested in doing more reading on your own, I can suggest three books by Peter Norton: *Discovering the IBM PCjr Home Computer*, *Exploring the IBM PCjr Home Computer*, and *Programmer's Guide to the IBM PC*.

PCjr Program Compatibility

Steve Mark
Atlanta IBM Employees PC Club

It has often been my experience that programs that are not supposed to function properly on the PC*jr* actually run just fine. The purpose of this article is to share, with other PC*jr* owners, the factors I have found that do and do not affect PC*jr* compatibility.

I will try to keep this discussion very non-technical. I don't know much about interrupt levels, assembly language, or soldering irons, and have no burning desire to learn now. I promise to get no deeper than an occasional reference to the CONFIG.SYS file and some Internal Modem command codes.

I also promise that everything in this article is based on my *personal* experience unless otherwise noted. There will be no "I think I heard somewhere that . . . "

My present system has 640K with two diskette drives. I started in September 1985 with a 128K, single drive PC*jr*. A month later, I bought a PC*jr* Internal Modem. Then came a 128K Microsoft Junior Booster, a Quadram Expansion Chassis that included a second drive, 384K, and a parallel port. A Hayes 1200 bps external modem and an IBM Proprinter finished the configuration (and my bank account).

Most reasons that programs supposedly will not run on the PC*jr* can be categorized as follows:

1. Configuration
2. Communications issues
   - The PC*jr* Internal Modem
   - Comm port addressing
   - DMA
3. DMA
4. The Video Buffer
5. It actually won't work on a PC*jr*.

This article discusses each of the above considerations and has a section that contains some simple techniques I have used to get the best performance from my PC*jr*.

Configuration

One of the earliest sources of misinformation regarding PC*jr* compatibility was the original maximum configuration. Many programs were said not to run on the PC*jr* simply because the programs required two diskette drives and/or more than 128K. To confuse matters more, many dealers and software vendors seemed unaware that several accessory manufacturers made it possible to add a second diskette drive and/or a fixed disk to the PC*jr*.

Some software manufacturers, however, came out with PC*jr* versions of their packages. These versions usually included one or two cartridges and were designed to run on a 128K, single-drive machine. The use of the ROM cartridges resulted in reasonably good performance, but prevented the PC*jr* version of these programs from running on anything other than a PC*jr*. The trade-off was that the user gained performance, but lost upward compatibility.

Because there were special PC*jr* versions of these programs, and because these versions would not run on other PCs, people believed that the PC version would not run on the PC*jr*. Wrong! In most cases, the "regular" version will run just fine on PC*jr*s that meet the package's configuration requirements.

Vendors created versions of their software (e.g., IBM's Planning Assistant) that utilized an overlay structure to fit into 128K, although with substantial performance degradation. Many PC*jr* users who have upgraded their machines to 256K and more are still suffering unnecessarily from the poor performance of the PC*jr* version of their software because they do not realize that the original constraint was memory, not the "*jr*" on their machines' nameplates.

The PC*jr* has one unique memory consideration. The Color/Graphics Adapter on a PC contains a 16K video buffer. The PC*jr*'s display adapter does not include memory for this buffer. Therefore, when you boot the PC*jr*, DOS allocates 16K of the system's memory for a video buffer. (The positioning of this buffer is the reason you need drivers such as PCJRMEM.COM in order to recognize memory beyond 128K, but that is beyond the scope of this article.) The result is that a program that has less than 16K to spare when running on a PC will not run on the same size PC*jr*. Remember that the words "requires a 256K system" usually mean "will not fit in a 128K system."

Communications

The PC*jr* Internal Modem

Where do I start? This little jewel has caused more confusion and consternation than all the tax simplification measures that the U.S. Congress could ever dream of!

Most of the problems in getting communications programs to run on the PC*jr* are really problems getting the programs to send the proper commands to the PC*jr* Internal Modem. The modem does *not* accept the Hayes (AT) command set. Most popular communication programs (QMODEM, PC-TALK, CROSSTALK, etc.)

are set up to issue the Hayes command set, with some method of specifying a different command set if necessary. In browsing through bulletin board systems around the country, it appears to me that many people have problems using non-Hayes compatible modems regardless of whether the attached PC is a PC*jr*. The PC*jr*'s internal modem just seems to get more criticism because it is concentrated in one environment.

Some programs (including the IBM PC Videotex Connection and the terminal program on the PC*jr* Sampler diskette) support the Internal Modem's command set and will dial, communicate, and hang up correctly. However, they often will not properly do things such as changing data format from 8-N-1 to 7-E-1. Other programs such as QMODEM allow the user to specify the control sequence for dialing, etc., but also will not issue the commands to change format. The result is that the modem will not dial, or else the received data looks like garbage.

It is usually very simple (at least it is with QMODEM and PC-TALK) to manually issue modem commands from the "terminal" screen of your program. Therefore, the best advice I can give you is to keep the PC*jr*'s command reference handy. There is one in the small supplement to the *PCjr Guide to Operations* that comes with the modem. There is a better one in the *PCjr Technical Reference*. It's not really as onerous as it sounds, though. The only time you should need to enter commands manually is when changing format or entering transparency mode.

Now comes the fun part. If all you want to do is exchange messages and download files, you can skip this discussion. However, if you want to endear yourself to your favorite SYSOP and be a good BBS citizen, you will occasionally want to upload a file or two. The PC*jr*'s internal modem makes that a real adventure. To understand why, you need to realize that the modem attempts to execute commands even if it finds them in the middle of a data stream that you are sending to another computer. If you try to upload a binary file (like a .COM or .EXE file), sooner or later the data will include a bit pattern that the modem thinks is its command character (Ctrl-N). It will not only fail to send that character down the line, but will try to execute the "command" it thinks comes behind it.

Without going any deeper into what happens when the modem receives what it thinks is an invalid command, or delving into how to put the modem into transparency mode, it should be obvious that the problem we are discussing is a function of the modem being used and not the fact that the system is a PC*jr*. Try all the tricks you can find documented on many bulletin board systems until you find one that works.

Communications Port Addressing

This is one problem you *won't* have if you use the internal modem. Some people, cannot get their communications programs to recognize external modems. If you have this problem, then you may be facing the COM1 versus COM2 addressing mystery.

When the PC*jr* Internal Modem is installed, it is COM1, and the external serial port on the back of the system is COM2. When there is no modem in the internal slot, the external port becomes COM1. So far, so good. The problem is that, regardless of whether an internal modem is installed, the base address and interrupt level of the external port remain the ones normally used for COM2.

If your communications program is well-behaved and does not try to bypass the system BIOS, none of this should cause any problem. Just tell the program that your external modem is COM1. QMODEM*jr* (version 1.07) works fine this way. Other programs seem to use the base address and interrupt level associated with whatever comm port you have specified for your modem. To use these programs (e.g., QMODEM 2.0 and 2.2), just tell them your modem is at COM2. That is how these three programs worked on my system after I removed the internal modem.

But I have heard of different people who got different results, supposedly using the same programs! In fact, on several bulletin board systems there are routines available that claim to swap COM1 and COM2 so you can use an external modem. The SYSOP of the IBM*jr* forum on CompuServe tells me the problem appears to be a function of the compiler used to compile the program and/or any external port drivers that the compiler uses.

If you think you have this problem, you have two choices. The first is to find an internal modem and plug it into your machine. This will straighten out the addresses so you can use your external modem as COM2. The second is to get one of the programs that logically swaps the addresses back to where they belong. These programs have names like SWAPCOM or COMSWAP, and can be found on many PC*jr*-oriented bulletin board systems.

There has been an interesting development on this issue. The QINSTALL program for QMODEM 2.2 lets the user specify the base address and inter-

rupt level for each comm port. The defaults are provided by the program, so all you have to do is switch the specifications for COM1 and COM2. Because I was unable to recreate the problem on my system, I could not test the effectiveness of this new feature. In theory, however, it should solve the problem.

Direct Memory Access

Direct Memory Access (DMA) is a standard feature on all IBM PCs except the PC*jr*. Simply stated, DMA allows the processor to overlap disk I/O operations with other work. Some manufacturers offer a means of adding DMA to the PC*jr*.

With regard to communications, DMA allows the system, when downloading a file, to continue receiving new data while the data just received is being written to diskette. Without DMA, a few incoming characters will be dropped on the floor each time the system writes out a buffer full of data. When your download completes, you will have, at best, a text file with some missing letters, and, at worst, an unexecutable program. Uploading is not affected, because the system will not try to send data while it is reading from the diskette.

If you have very little extra memory beyond that required to run your communication program, then the lack of DMA is indeed a problem. But if you have extra headroom, the solution is very simple: do your downloading to a RAM disk.

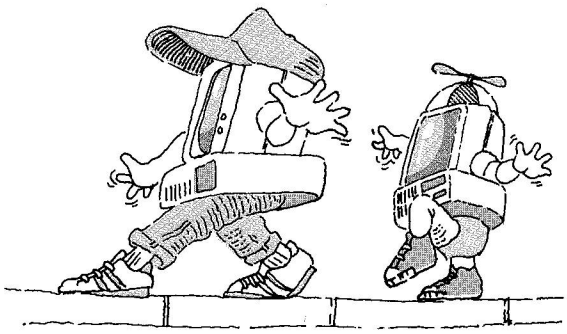
The documentation for QMODEM 2.2 says, "A PC*jr* must contain a DMA chip to successfully use the transfer protocols." I say this is not true. I normally run QMODEM with a 200K RAM disk. If I plan to download more than 200K of files, I simply install a larger RAM disk. I have successfully up/downloaded files to bulletin board systems, CompuServe, and point-to-point. After the transmission is completed, copy your new files to a real diskette immediately to prevent losing them in a power outage.

I know that several PC*jr* users have experienced system hangs that require a reboot to clear. This would make one quite nervous about trusting a RAM disk to receive a lengthy file. Let me try to set you at ease. There is a diskette containing patches to DOS 2.10. These patches fix a bug in the way DOS handles (or fails to handle) certain keyboard interrupts that are unique to the PC*jr*. I installed the patches (a very simple process) over six months ago and have not experienced any lock-ups since that time. You should be able to get the patch diskette from your dealer or another PC*jr* user. Ask around, it's worth the search.

DMA in General

The lack of DMA affects the performance of programs that use diskette drive(s). However, except for the communications concerns covered above, this shortcoming rarely affects whether a program will indeed run on a PC*jr*.

Some programs will simply not run correctly, or at all, without DMA. I also understand that some copy protection schemes use the DMA processor. If true, these programs obviously will not run on a standard PC*jr*. (I do not have a DMA chip in my PC*jr* and so have had no first-hand experience with these programs.)



If the software you want to run requires DMA, all it takes to run it on your PC*jr* is money. At least one manufacturer offers a DMA chip for the PC*jr*. The hitch is that the chips usually come bundled in expansion units which also include second drives, parallel ports, and other goodies. That's fine if you are just starting to expand your PC*jr*, but if you don't want (or already have) a second drive, it's an expensive DMA chip.

The Video Buffer

Here again, I am at (and sometimes beyond) the limits of my personal knowledge, but I'll give it a try anyway.

The video buffer is where the hardware gets the information it needs to put characters and pictures on the screen. Your program has some options as to how (or if) it will put information into the buffer. If the program uses DOS or normal BIOS to write to the buffer, then there should be no problem. Fortunately, most programs are well-behaved in this regard. If your program tries to address the buffer directly, or (worse) tries to write directly to the display, bypassing the buffer, then you've got a problem.

There are two differences between the way the video buffer is handled on the PC and on the PC*jr*. First, as I mentioned earlier, the video buffer is in main memory on the PC*jr*, rather than on the display adapter as is the

case on other PCs. This means it is at a different physical address. If your program tries to write directly to the address where it thinks the buffer is, it won't work. This is especially true if your PC*jr* has been expanded beyond 128K, because the memory management software (PCJRMEM.COM or equivalent) moves the video buffer so DOS can find the expansion memory. I have not run across many programs that fall into this category, but I suppose that is small comfort to those of you whose favorite game does. I do not know of any method of getting such programs to run on the PC*jr*.

The second problem is another that just takes money to solve. In addition to being in a different place, the PC*jr*'s video buffer is mapped differently than the PC's. This means that a position in the PC*jr*'s buffer corresponds to a different place on the screen than the same position in the buffer on the PC's Color/Graphics Adapter. A program that tries to manipulate data directly within the buffer will run, but the screen will look very strange. I ran across a game once whose title screen was garbage. I took a guess, pressed "any key," and the rest of the game ran fine.

As I said, all it takes to solve this problem is money. The mysterious little PC/PC*jr* switch on the back of second drive units is there for precisely this purpose. When you power up your machine with the switch in the PC position, and execute the PCVIDEO routine that comes with the hardware, the mapping of the video buffer is changed to match the PC's. After that, my game had a real title screen. If you really want to, you can even run a PC*jr* using DOS 1.10 this way.

Some Programs Will Not Work on a PCjr

The vast majority of software written for the IBM PC will run on an adequately configured PC*jr*. In addition, we have discussed several ways to make programs that supposedly will not run on the PC*jr* work as well.

The simplest way to speed up your PCjr is to add memory.

There are programs, though, that cannot be made to execute correctly on our PC*jr* machines. Most of these programs write directly to the hardware interface for the diskette drive or display. They do this to optimize performance, or to bypass some limitation of the interface supplied by the system BIOS. Some games and graphics programs write directly to the display to do fancy manipulation of the screen images and improve performance. Unless there is a PC*jr* version of these programs, you're probably out of luck.

A few programs rely on timing to run correctly. And finally, there are those programs that require some piece of hardware (like a math coprocessor) that cannot be attached to the PC*jr*.

All told, these problems represent a very tiny portion of the mountains of software available for the PC*jr*.

Performance Tips

The simplest, most effective thing you can do to speed up your PC*jr* is to add memory. There are three ways you can use additional memory to achieve a significant (up to 50 percent in some (cases) increase in performance.

The first was discussed earlier. Some programs such as Writing Assistant and Planning Assistant come in two versions. One of these versions is structured to run in a 128K machine. It generally makes heavy use of overlays and goes out to diskette each time you invoke a new function. It works in 128K, but I hope you like the sound of your diskette drive grinding away. The other version, a resident version, loads completely into memory and only needs to access the diskette for data files. An added advantage is that after the program is loaded, you can usually remove the program diskette and insert a separate data diskette. This gives you much more room for files on a single-drive system.

The second performance improvement you will gain from adding memory takes us back to our old friend, the video buffer. In order to keep the current image on the screen, the image needs to be refreshed every few microseconds. If you remember, the PC*jr*'s video buffer is in main memory. Because of this, the cycles it takes to do this video refresh are taken from other jobs that are executing in main memory. (I'm not sure, but I believe one out of every three cycles is used for this purpose.) That is why the PC*jr* seems to process slower than the PC even though it has the same clock and 8088 processor. The good news is that this impacts only the first 128K of memory. Programs that are loaded into expansion memory are not affected.

There are two ways to force all of your programs to load into expansion memory. The simplest is to use the "/C" parameter with PCJRMEM.COM in your CONFIG.SYS file. This causes DOS to fill the system's main memory with I/O buffers, and force all user programs to be loaded above 128K. It also allows the use of the PC*jr*'s enhanced video modes, but that's another story.

The other way to fill up the first 128K is to define a RAM disk of at least 90K. This combined with DOS and the video buffer will fill the main memory, and your programs will be loaded into the expansion memory.

Does it really help? I have a program called THATSALL.EXE that plays the well-known cartoon theme and writes "That's all Folks" across the screen. It takes about 26 seconds to run in main memory, but in expansion memory, it takes about 17 seconds, the same as on a PC or XT. That's a 35 percent improvement.

The third way to take advantage of additional memory takes us back to my old favorite, the RAM disk. Let's go back to those programs that had to overlay themselves to fit into 128K. Sometimes the full version is a separate product that will cost you either an upgrade fee or the full price for the product. If you don't need the additional features that may be available in the full product, you can still get near-resident performance from the PC*jr* version.

Just load the program and its required modules onto your RAM disk, and execute it from there. It will still go through its overlaying process, but will do so at the speed of memory rather than the diskette drive. With a little experimenting, it's not hard to determine which files must be copied to the RAM disk to make this trick work. (Note: this technique does not apply to overlaid programs.) I use this technique to make the Personal Computer Picture Graphics program run almost as fast on my PC*jr* at home as it does on the XT with fixed disk at work.

The PCjr is a much more useful machine than it is given credit for.

Some programs, including the Assistant Series, allow you to specify a work drive for the program to use when sorting and doing various other tasks. By using your RAM disk for this purpose, many jobs will go a lot faster.

Are you tired of having to swap back to your DOS diskette every time you want to use an external command such as DISKCOPY, CHKDSK, PRINT, FORMAT, etc.? Why not just copy those programs that you use often onto your RAM disk when you boot the system? Then they will be right there on drive C for you to use whenever you need them. It's almost like having a small fixed disk.

The one caution to remember about a RAM disk is that when you reboot, or if a power fluctuation causes the system to do a power on/reset, the contents of your RAM disk are lost. Therefore, you should not put any non-recoverable data on a RAM disk. I'm a little gutsy, so when I use Filing Assistant, I copy my file to the RAM disk, update it there, and then copy it back. *(Editor's note: Don't be gutsy in South Florida, where lightning often causes power outages.)* I figure that, at worst, I may have to reenter the session's updates if something happens before I can copy the file back to diskette. In the eleven months I have been operating this way, I have never had a problem. Not only does it seem as if I were using a fixed disk, but I've also saved a lot of wear and tear on my diskette drives. If you are using Reporting Assistant, the effect is truly amazing.

Summary

I hope this has been of some help to you. I have tried not to go too far away from the intended subject. For example, there has been no discussion of the PC*jr*'s advantages over the PC (video, music, and a smaller footprint), or the fact that the PC*jr* comes with things that you have to add to a PC (like a display adapter, serial port, game port, etc.). My objective has been simply to show that the PC*jr* is a much more useful machine than it is generally given credit for.