AMS DEVELOPMENT SYSTEM INTRODUCTION

Enclosed please find the completed AMS Development Kit. This package, which is included with all AMS cards is fairware, and may be freely distributed by user groups and other organizations.

This package consists of the following items:

AMS System "Boot", version 1.2
AMS Macro Assembler, version 1.1
AMS Object Library, version 1.1
AMS Object Linker, version 1.0

Users of the development system are encouraged to forward a donation to the author, R.A. Green, if they find it useful. Additionally, users are encouraged to write or call with any problems found, so that they can be corrected.

While the amount of labor and time put into this project should impress anyone (our development team, including Art Green, Joe Delekto, Jim Krych, Tony Lewis and myself have been working on this project for over 2 years now), what does this all mean to the TI community? This is a fair question.

Memory management, the design of memory systems, and programming for memory systems is a pretty esoteric discipline. The people involved on this team are the most knowledgeable people in the community on these issues - in fact, I would go so far to say only a 20-25 people in the community can really appreciate their accomplishment, and perhaps only 5-10 could begin to duplicate it.

While the subject is pretty mind-numbing stuff to the average person, if it was solely an exercise to demonstrate technical skills, it would be fair for the community to just congratulate these guys and go back to sleep. However, while the complexity seems unreal, the reason why this was done, and the ultimate results are very important to the future of our community.

Why did we do this? Drastic times call for drastic measures. The TI community is dying because the 99/4A, as-is, can't be pushed much further. More capable software can't be written because there is no memory to put it in. New devices like MIDI interfaces, sound digitizers and multimedia are impossible not because of a lack of speed, but because there is no place to put their data. Graphical user interfaces and desktop publishing, as well as traditional programs like spreadsheets, word processor and databases are hindered more by a lack of memory than anything else. Asgard realized this years ago when some of our more ambitious projects screeched to a halt - because we realized that while we could sketch out what we wanted on paper, there wasn't enough room in the 4A to run it.

While the Geneve was built to address this problem (among several others), we also came to believe that it's a dead-end. Myarc used too many proprietary parts to cost-effectively produce the Geneve. Even if all the system software worked perfectly (which it may never do because of bugs in the hardware), the machine is simply too expensive to make. For this reason, we decided to focus our efforts on the 4A.

We then examined other memory devices available to see if they could be made to work as an Extended Memory Device (EMD). Virtually all of them were totally inadequate for the job. Only OPA's RAMBO came close, and it suffered from a wide range of problems - the major ones being unreliability caused by the fact that the memory card was used for other things (as a RAM-disk), a difficult to use programmers interface, and very slow paging software and hardware. The fact that the designer of the device agreed with our assessment was no comfort.

The only way out of the problem was to go back to the drawing board and start from scratch.

On the hardware end we decided that the only way to make a fast, reliable memory card for programs and data was to make a device dedicated to the purpose. This meant that the card was only usable as an EMD and nothing else. This approach also offered the benefits of dramatically reducing the basic complexity and cost of the hardware design to not much more than that of a 32K card, and of reducing the chance of compatibility problems that has plagued other memory cards. On this basis, we ruled out trying to adapt other cards already available.

After thinking through the hardware, we began exploring what we needed to do on the software end to make it work. We began exploring "segmented memory systems" (as this kind of thing is called) used in the rest of the computer world. After much work, we decided that the approach used by Intel with the 8088 (and TI in their 99/8 and 9900-based minicomputers) was the most applicable - the concept of overlays. Considering how much mileage the PC has gotten out of this idea, with inferior hardware, it seemed like a good idea to us.

After deciding on an approach, we began designing the hardware and the software. Early on we had to make a decision as to whether or not to put the core controlling software for our device as a DSR on the card or not. After considerable discussion, we decided that a DSR would be a mistake. Why?

1. On a basic level, because the goal of our card was to use memory as memory, not as a device like a RAM-disk or an RS232 card.

2. We found that DSR based software executes much slower than RAM-based software - which is important considering that the core routines would be used constantly. Trying some scheme to load them from a DSR into RAM before executing would increase the hardware complexity and cost, and the chance of a compatibility problem, dramatically.

3. Along the same lines, any DSR would increase the chance for compatibility problems - the last thing we wanted to do was to waste time debugging problems with Myarc or Corcomp products caused by a DSR bug.

4. Lastly, DSRs are fixed. If you find a bug in it, the only way to correct it is to replace it. Consider all the trials and tribulations Myarc and Myarc users went through with periodic EPROM upgrades of the HFDC and the Geneve. Also, if software

is written to work around a DSR bug, it may not work with an upgraded DSR.

In contrast, if the software is in RAM, the only thing we would have to do to correct a bug is issue an upgrade. Old programs written for earlier versions of the operating system software would continue to work fine, and new programs could take advantage of new features without worrying about hardware compatibility problems. This is the same reason that Microsoft and Apple load their operating systems from disk, and not from ROM chips.

Finally, on a design level, we felt that we couldn't make the system as easy to program for by trying to shoe-horn everything we needed into a little DSR. The AMS development software takes up 5 DS/SD disks - much too much to squeeze into even the largest DSR.

After deciding on approaches for the hardware and software, the only thing left to be done was to implement it. Of course, implementing it DID take 18 months. While the difficulty in designing and building the hardware was nothing to sneeze at, for the most part the real innovation was in the software.

On the software end, the most critical piece was the Linker. This program does all the work of taking pieces of a program and combining them together into a program that can be loaded into, and take advantage of the memory card. The Linker embeds controlling code into each program and its subroutines that makes sure everything is in the right place at the right time. The real innovation is that it is able to do this without the programmer having to explicitly program for it. The only thing the programmer has to do is make sure his program obeys a number of rules that 99/4A programs already generally do anyway. Once done, the Linker will generate a program theoretically up to 16Mb in size.

Another innovation is the Loader, which is built into the System Boot program. The loader takes programs created by the Linker and loads them into the memory card in the pages of memory designated by the Linker. Our particular version also functions as a "memory manager" - keeping track of memory usage, and allowing several different programs to be loaded at once (as you would a TSR or Device Driver on a PC).

Another critical piece is the Object Library. This file contains a range of subroutines that allow a program to ask for pages of memory and move data around in them. A programmer using these routines in his programs can take advantage of the memory to hold data without explicitly writing code to do so into his/her programs. Further, a Library Manager is included with the package that allows a programmer to customize the library.

Last, but certainly not least, another major component of the package is the Macro Assembler. The most capable assembler available for the 99/4A or the Geneve, this macro assembler further simplifies the programmers life by including common assembly and memory functions in macros. The assembler itself also takes advantage of the memory - giving you much more space for labels (variables), and other such things than you would normally have in an assembly program.

What does all this wonderful technology do that benefits non-programmers? Simply enough, it enables programmers to write programs for the 99/4A that are a match for the capabilities of PC and Mac programs, with no more effort than you would need to do so on a PC. While it doesn't provide an immediate benefit to a non-programmer, in the long run, non-programmers will benefit from what this device lets programmers do - and that they could never do before.

That is why this is so important to the future of the 99/4A. In fact, without a device like this, the 4A probably has no future. With it, the 4A has enough memory to keep programmers occupied and interested for years, and just as importantly, programmers have the tools to use that memory.

Asgard Peripherals is proud of the accomplishment of these talented people, and that we had the opportunity to facilitate and support this kind of effort. We also want to make sure that the product of their labor is not ignored. What we are doing is basic to so many other things we want to do, and to the computer in general. That is why we are offering to license the design of our basic AMS card to anyone and everyone at practically no cost. That is also why all the development software is available free of charge.

Chris Bobbitt
May 12, 1993

Note 1: May be reproduced by anyone if unaltered.
Note 2: While AMS was not designed for Geneve compatibility, we haven't forgotten 9640 owners. Watch this space for future announcements.