AI Magazine Volume 2 Number 1 (1981) (© AAAI)

# Computing Facilities for AI:

## A Survey of Present and Near-Future Options

Scott Fahlman
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

At the recent AAAI conference at Stanford, it became apparent that many new AI research centers are being established around the country in industrial and governmental settings and in universities that have not paid much attention to AI in the past. At the same time, many of the established AI centers are in the process of converting from older facilities, primarily based on Decsystem-10 and Decsystem-20 machines, to a variety of newer options. At present, unfortunately, there is no simple answer to the question of what machines, operating systems, and languages a new or upgrading AI facility should use, and this situation has led to a great deal of confusion and anxiety on the part of those researchers and administrators who are faced with making this choice. In this article I will survey the major alternatives available at present and those that are clearly visible on the horizon, and I will try to indicate the advantages and disadvantages of each for AI work. This is mostly information that we have gathered at CMU in the course of planning for our own future computing needs, but the opinions expressed are my own.

Before going on, I should note this discussion will be limited to those machines and systems that are (or will be) in active use at one or more of the major established centers of AI research in the United States. This limitation is deliberate. in my opinion, it would be unwise for a new center to start from scratch with a machine or system that has not previously been used for serious AI research. To do so would be to take on a tool-building task that would delay the beginning of serious AI research for several years. Using an odd machine also tends to isolate the research group from the rest of the AI community. It seems a much wiser course to tie one's center in with others, so that tools and results can be shared

Of course, this does not mean that one cannot do AI research on practically any machine if there is no other choice. Good AI research has been done on machines from Univac, Burroughs, Honeywell, and even IBM, using unbelievably hostile operating systems, and in languages ranging from Basic to PL-I. If corporate policy or lack of independent funds forces some such choice on you, it is not the end of the world. It should be noted, however, that the lack of first-rate facilities is very likely to lead to a lack of first-rate people, and will have a serious impact on the productivity of the people you do manage to attract. In this article, then, I will concentrate on the choices that might be made by centers that have a free choice in the matter and the funds to obtain facilities that will be dedicated to AI use.

One other warning must be given: this material will become obsolete very quickly. If you should encounter this article a year after publication, much of it will be out of date. If the time is two years after publication, it will be totally worthless.

### Basic Computing Needs

What does an AI researcher want from his computing facility? What will make him most productive? Setting

aside, for now, the very specialized needs of those doing signal processing or robotics, the needs of the rest are relatively straightforward to state, if difficult to satisfy. In fact, the needs of the AI researcher are not very different from the needs of any other researcher in computer science, except that facility-related problems seem to become acute in AI a few years sooner than they are felt in other research areas The considerations are roughly as follows:

- □ AI programs tend to be very large because they contain, in one form or another, a lot of knowledge. It follows, then, that any machine used for AI must provide a large virtual address space in order to insulate the researcher from having to think about how to chop up his task into smaller tidbits and overlays. A 32-bit address space is comfortable for the forseeable future; a 24-bit (to the 32-bit word) address space is adequate for the next couple of years for most purposes; the 18-bit address space of the Dec-10/20 series is woefully inadequate and has seriously impeded the recent progress of AI. Don't even think about using anything smaller.
- □ Most AI programs burn a lot of cycles. If your machine is slow or is too heavily loaded, your high-powered researchers will be spending all of their time waiting for something to happen on their screens They will spend this time plotting against the management and reading the help-wanted ads To pack too many researchers onto a time-shared machine is a move of very dubious economic value.
- □ The operating system must be friendly. For AI, friendliness means flexibility Protection and quotas must not get in the way, the utilities must be screen-oriented rather than paper-oriented, documentation must be on-line and easy to use, and individual users must have easy access to multiple processes so that some of the waiting time can be spent editing or doing other useful work. Most important, the system should stay up: when you have invested an hour of CPU time and eight hours of your own time in a computation, a crash can be very irritating. Since most AI researchers are experienced programmers, a user interface that is easy for beginners to use is relatively less important than on a machine for general use, but remember that you may want to export your results to an environment with less sophisticated users.
- □ Though AI researchers spend more of their time computing than most people in computer science, they still spend most of their time editing programs, editing text, formatting documents, and communicating with one another via computer mail. First-rate facilities for all these activities must be provided. My own working definition of "first-rate", among systems available today, would be the combination of Emacs and Scribe. Tastes may vary in this, but it is clear that the use of a teletype-oriented editor or a primitive text-processing system can waste a great deal of your researchers' valuable time
- □ The programming done in AI is almost always of an experimental, evolutionary nature, and it is concerned mostly with symbol manipulation rather than arithmetic This argues very strongly for the use of Lisp over other currently-available languages An AI research center *must* provide a well-developed, well-maintained Lisp system; other languages are optional. The use of such languages as Sail for AI research seems to be declining rapidly now that good Lisps are widely available It may be, as some researchers argue, that Smalltalk-like languages are the wave of the future, but that wave is still well over the horizon for most of us.

As I said, these requirements are simple to state, but hard and expensive to realize. To equip an AI center in a way that will help you to attract the best people, you should probably plan to spend something like $50K-$70K per researcher in computing equipment. (For serious work in robotics, plan to spend a lot more ) An established center can get by with less, but if you are trying to start a new center it will be very hard to recruit people to work with inferior facilities There are many examples of well-intentioned efforts that never reached critical mass in people because the computing environment was wrong.

In providing these facilities, there are two basic approaches: time-sharing and the use of powerful personal machines connected by a high-bandwidth network. Time-sharing has been the mainstay of the field for over a decade and, as I write this, is still the only option available from commercial sources. The personal computing option is expected by most leaders in the field to be the dominant force of the next decade, starting very soon, but it will be a year or two before it is a practical option for users who do not want to do a lot of the development themselves That is one of the reasons why the field is currently so unsettled: users must acquire enough time-sharing capacity to meet their present needs, but they want to save enough equipment money to allow for a quick move into the personal-computing world as soon as this becomes practical. We will consider the options in both of these worlds in the following sections.

## Time-Sharing Options

Two families of time-sharing machines are used by the

vast majority of AI researchers: the Decsystem-20 (and its predecessor, the Decsystem-10), and the Vax, both products of the Digital Equipment Corporation. As I said earlier, it is possible to do AI on other machines, but anyone with a choice in the matter should probably stick with these two hardware families.

## The Decsystem-20 Family

The Decsystem-20, available in a range of sizes, provides a mature programming environment and by far the largest selection of useful software. Unfortunately, an 18-bit address space is woven deeply into the instruction set of the Dec-20 family. When this family began with the old PDP-6, this must have seemed like an immense address space but, as we noted earlier, it is just too small to meet the needs of AI researchers in the 1980's. Some of the newer models of the Dec-20 extend the address space beyond 18 bits, but the additional address space is awkward to use and very little of the existing software can take advantage of it

Despite the inadequate address space, there are some situations in which the Dec-20 may be the option of choice. In a crash project that will not be needing more than the available address space, the user amenities on the Dec-20 make it very attractive. In a situation where the research will consist mainly of using and extending existing AI systems, and where these systems run on the Dec-20, it is obviously the machine to use. In a large center, it may be advantageous to have users edit, process text, and do most of their program development on a large Dec-20 system, while big Lisp jobs are sent to one or more Vax machines for service. This option requires a very good local network if it is to be successful.

If you do use the Dec-20 hardware, you have a choice of two operating systems from DEC: Tops-10 and Tops-20 Tops-10 is an outmoded system that is totally unsuited to the needs of AI. Tops-20, based on the Tenex system developed at Bolt, Beranek, and Newman, is clearly superior since it provides demand paging, tree structured directories, multiple processes per user, flexible terminal handling, and the friendliest user-interface of any system that I have seen. (Some of these features have been tacked onto Tops-10 as afterthoughts, but in very clumsy forms.) All Tops-10 software runs on Tops-20, but the converse is definitely not true. Tops-20 users can run Interlisp, Maclisp, Emacs, Scribe, Tex, and most of the major AI programs that have been developed in the past decade Some sites still run the older Tenex system; this is almost equivalent to Tops-20 in its features and available software, but users are on their own for maintenance.

While DEC states that they are not trying to phase out the Dec-20 in favor of the Vax, their pricing structure, especially for main memory, tends to make the Dec-20 family relatively unattractive. An option that some users may want to consider is a line of machines from Foonly, Incorporated These machines execute the Dec-20 instruction set and can therefore run most of the same software They tend to be substantially less expensive than the comparable machines from DEC, but they must run Tenex rather than Tops-20, and maintenance may be a problem in some areas. If you have the staff to do some of your own hardware and software maintenance, Foonly seems like a good option; if not, you should carefully explore the maintenance issues before buying.

In the Tops-20 world there are two major Lisp systems in use, both with fanatical adherents. Interlisp, developed at BBN and Xerox PARC, and Maclisp, developed at MIT. Interlisp contains a large number of built-in facilities to provide the user with a total programming environment -- arguably the best programming environment ever provided for any computer language. Everything from a built-in program editor to an indexing facility to a spelling corrector is provided as part of the system All of this is documented and centrally maintained. Maclisp proponents point out that this wealth of features in Interlisp can often be more confusing than helpful and that little address space is left over for the users' programs.

Maclisp is a much leaner (some would say more primitive) system, in which efficiency has received the primary emphasis Maclisp's compiler is able to produce very efficient fast-loading code, especially for arithmetic, which has traditionally been a weak area for Lisp Many of Interlisp's more complex features are available in Maclisp as optional, user-loadable packages, but these are not considered part of the Maclisp system itself. Maclisp code is normally edited externally in the Emacs editor, which knows about Lisp syntax and pretty-printing; a special linkage between the two systems makes it easy to alter individual function definitions in the middle of a run. This external editing style has some advantages in dealing with comments and macros in the Lisp source; it can be awkward to handle these in an internal S-expression editor. It is also argued that it is easier for users to edit Lisp with the same editor that they use for everything else. Since several Tops-20 systems are now in use at MIT, I would expect future maintenance of Tops-20 Maclisp to be on a par with Interlisp maintenance. Documentation for Maclisp has been scandalously poor in the past, but the situation seems to be improving

I have an opinion in the Maclisp vs. Interlisp debate, but I will not express it here. Both systems are very good programming environments. The dialect you use will be determined by the tastes of your people (a strong function of where they were educated) and by the language used by any collaborators you may have at other sites. It is relatively easy to transport most code between the two systems; it is harder to move users from one system to the other, since the user environments are very different.

## The Vax Family

Digital Equipment's Vax family of computers is a newer design than the Dec-10/20 series, and its 32-bit virtual addressing solves the space problem for the forseeable future. It would appear that the Vax is destined to be the next major time-sharing machine for most of the computer science research community, including AI. If personal computing develops as quickly many of us believe it will, the Vax may well be the last time-sharing system that is in common use in the research community At present, however, the Vax world is lacking many of the software amenities that are available on the Dec-20. The software situation on the Vax is being improved rapidly as Vaxes come into common use at major research sites. At some point, perhaps a year or so in the future, the Vax will become a nicer machine to work on than the Dec-20. Given comparable software, the Vax's large address space will certainly make it a superior machine for AI

There are two major operating systems available for the Vax· the VMS system, supplied by DEC, and the Unix system, supplied by Bell Labs but extensively modified by members of the ARPA-sponsored research community A third option, the Eunice package from SRI, is an emulator which disguises VMS to look like Unix to users and to programs

DEC's VMS system is a curious mixture of strengths and weaknesses Some of the system's internal mechanisms for paging, process scheduling and switching, and buffered disk I/O are very good, and are to some extent tunable to meet the needs of a particular installation. Unfortunately, the face that this system presents to users is an unpleasant one, obviously meant to appeal to the Fortran/Cobol market and not to those users who want a modern, flexible environment for editing and program development Users can run only one job at a time, must contend with a very clumsy system of quotas and restrictions, must do all their I/O through a complex Cobol-ish record management system, and must deal with terminal drivers and system utilities that are strongly oriented toward the use of old-fashioned line editors. VMS is a large and complex system written mostly in assembler, and the sources for the system are expensive to obtain. This means that it is hard for users to modify the system except in the ways that DEC anticipated. Many of the VMS system's problems require only minor fixes, but users will have trouble making such fixes and DEC moves very slowly on such matters

Unix also has its problems. The system was developed on the PDP-11 many years ago, and was moved to the Vax without much modification by Bell Labs. The system is full of concessions to the PDP-11's tiny address space and to the printing-terminal mentality that seems to have permeated Bell Labs until very recently. However, since it is very simple and is implemented in C, the Unix system is relatively easily modified to meet the needs of any given site or the opportunities presented by a new machine. It was largely because of this flexibility that Vax/Unix was chosen over VMS for use in the ARPA-sponsored VLSI and Image Understanding projects. This choice, in turn, has influenced many other research efforts to use Unix as well, and to try to coordinate the changes and improvements that they make

A group at the University of California at Berkeley has taken the lead in this effort by adding demand paging and many other useful features to Vax/Unix. This group is responsible for VI, probably the best screen editor that is currently available on the Vax, though it lacks the flexibility of Emacs. They are also responsible for Franz Lisp, a nearly-compatible version of Maclisp that is written in C. At present, this is the only serious Lisp that runs on the Vax. Franz Lisp is still experiencing some growing pains and bugs, but its users at CMU seem to find it livable in its present state. It is slower than Maclisp on a comparable Dec-20, but it does make use of the Vax's large address space.

As I said, the Vax/Unix software world is improving rapidly Franz Lisp and the Berkeley pager are being worked on to improve their efficiency. A group at USC/ISI is working on an Interlisp system for Vax/Unix, and expects to have a version available by the end of 1981. Scribe has just been moved to Vax/Unix, and a group at CMU has implemented an inter-process communication protocol that solves some long-standing difficulties with Unix pipes. In my opinion, the major items still missing from Vax/Unix are an editor with the power and flexibility of Emacs, a tree-structured information system to replace the present clumsy online manual, and a more intelligible interface to the operating system and the assorted utilities. (For some reason there is a tradition on Unix that programs should have two-letter names and meaningless single-character option switches. This was ugly but tolerable on a simple minicomputer system; it is quickly becoming intolerable in a diverse, software-rich research environment.)

Meanwhile, back in the Vax/VMS world, a group at MIT is working on NIL, a reimplementation of Maclisp with many added features. An emulator for NIL is now running on top of Maclisp, and the Vax/VMS version is nearing completion. The system may be ready for outside users in six months or so according to its developers. A version of EMACS is being written in NIL, and should be ready whenever NIL itself is. The NIL project has been plagued by delays, but when NIL is done it will offer more features and will probably be faster than Franz Lisp. Since NIL is a superset of Maclisp, it should be easy to move code from Maclisp or Franz Lisp into NIL.

The Eunice system from SRI is an attempt to get the best of both worlds by emulating the Berkeley-modified Unix system on VMS. I have not yet had a chance to observe Eunice first-hand, but it is said by its developers

to run all Unix software without significant modification, and to do so at higher speed than on true Unix because of VMS's superior paging and file I/O. VMS and Eunice users can coexist on the same machine. Eunice is running now and has already been used to make Franz Lisp available to the VMS world. If all of these claims are true (and I have no reason to doubt them) then Eunice appears to be the best system for most users of the Vax. it is faster than real Unix and gives its users access to software developed for either of the other two systems. Unix retains some advantages in the area of user modifiability and simplicity, for users who need that

In summary, I would say that Eunice or Berkeley Unix on the Vax looks like the right combination for most new AI centers to use. The Dec-20 is is a more comfortable system to use at present, but its address-space problem is fatal in the long run, and the Vax software situation is improving rapidly.

## Personal-Computing Options

Time-sharing is based on the assumption that computers powerful enough to be useful for research, especially AI research, are so expensive that they must be shared among many users. Advancing technology is rendering this assumption obsolete Every year the price of computers and memories comes down and personnel costs go up. The task of the past two decades was to find ways to use every precious computer cycle for productive work, the task of the 80's is to find ways to improve the researcher's productivity, even if some computer cycles are thrown away in the process This change in the relative costs of machines and people provides the impetus for the move to powerful personal computers for AI research. The compromises inherent in time-sharing are just too wasteful of scarce human resources

Two major research centers, Xerox PARC and the MIT Artificial Intelligence Laboratory, have taken the lead in exploring this new world using machines that they have built themselves, the Dorado and the Lisp Machine, respectively. From these two efforts, and from other efforts that are starting elsewhere, a consistent picture of the next generation of AI machines is emerging. Their features include the following:

- Each user has a machine whose speed is at least comparable to that of KA-10 processor, the workhorse time-sharing machine of an earlier era
- Each machine has a high-resolution raster-scanned display, a keyboard, and a pointing device such as a "mouse". A color display and audio I/O are optional.
- Each machine has something like a megabyte of main memory and 100 megabytes of local disk storage, which is used as swapping space to provide a large virtual memory.
- The machines provide a large user-writable microstore, which is used to provide support for graphics, high-level languages, and sometimes to accelerate the user's critical inner loops.
- All of the machines are connected to one another by a high-bandwidth network -- an Ethernet or something comparable. This network also connects the individual machines to the printers, file-storage machines, and other shared resources.

The last component, the high-speed network, is an essential component of this technology. An important feature of the computing environments that were developed on time-sharing systems was the easy communication and the sharing of information and files among users. The high-speed network, along with appropriate software, allows us to bring this same ease of communication and sharing to the personal-computer world. It also allows for the sharing of the items that are still too expensive to replicate. printers, file systems, perhaps even a Cray-1 for the occasional job that wants to crunch lots of numbers

By moving our research to personal computers, we obtain a number of advantages

- Large, cycle-intensive programs can be run efficiently. In a time-sharing system, it is possible to meet the demands of the big AI programs or of a lot of users doing interactive editing, but it is very hard to satisfy both groups On a personal machine, the system can be tuned to the task at hand.
- An interactive user interface of very high quality can be provided. This is due to the high-resolution display on each machine and to the instant availability of enough processing power to update that display when necessary. It is hard to provide such timely bursts of processing on a time-shared machine.
- The user has access to the full power of his machine, no matter what other users are doing No more waiting until 4 a.m. to get cycles.
- Reliability is an inherent part of the personal machine environment. A failure in any one machine cannot bring down the others. The critical demonstration or production run can simply be moved to another machine.
- Some AI work requires the use of special, experimental processing hardware for such things as image-processing, knowledge-base searching, control of manipulators, and

complex graphics. Such devices can easily be added to a personal machine, and can get the instant service that they may require. It can be very awkward, both technically and politically, to add experimental devices to a heavily used time-shared machine.

- ☐ The computing environment can be extended in smaller increments than is possible on time-shared systems. If a few new researchers join the group, a few new machines can be added.
- ☐ Any AI application systems that are developed on personal machines can be exported easily, simply by having the customers buy the same machines that they were used to develop the program.

Now for the bad news: as of today, you cannot go out and buy a personal machine of the typed described above. To date, the only research groups who have been able to use such machines are those who have built machines for themselves. Such machines will be marketed soon, but none have yet been delivered. So, despite all of the advantages noted above, you will have to live with time-sharing for another year or two. Still, it would be a mistake not to keep a close eye on the development of personal machines for AI, so that you can jump in when the time is right. In the remainder of this section, I will describe what I feel are the efforts that should be watched most closely.

Xerox PARC began the new era of personal computing with the development of the ALTO, a nice machine for some uses but too small for use in AI except as a graphics terminal. Xerox has recently developed a much more powerful machine called the Dorado. This machine is implemented in ECL and runs both Interlisp and Smalltalk. A slower and less expensive machine, the Dolphin, runs much of the same code as the Dorado and is coming into widespread use within Xerox. This machine is near the lower boundary of usefulness for AI—reasonable for many applications, but not for the large, cycle-intensive ones. Xerox seems to have no interest in producing the Dorado for outside sale, and is still trying to decide whether to sell any Dolphins to outsiders. For the near future, then, it appears that Xerox's contribution to AI and personal computing will be mostly in the form of ideas, not hardware.

The only other personal machine that has seen active service in AI is the Lisp Machine (sometimes called the CADR, since it is the second iteration of the design) from the MIT AI Lab. These machines have been in active use for about two years at MIT; to date, about a dozen have been built. The Lisp Machine is implemented in TTL, and it runs Lisp programs at a speed that is somewhere between that of a dedicated KA-10 and a KL-10. The Lisp system used on this machine is based on Maclisp, but it has many advanced features that depend critically on the Lisp Machine's micro-codability. In fact, aside from about 9000 words of custom microcode, all of the code on the Lisp Machine is written in Lisp. The software includes a complete Lisp system with debugging aids and a compiler, graphics support, an Emacs-like editor, support for Smalltalk-like object-oriented programming, and a micro-compiler for turning some of the user's time-critical functions into Lisp Machine microcode.

After a number of false starts, there is now a company that is preparing to build and sell the Lisp Machine commercially. In fact, there are two such companies, reflecting a schism within the group at MIT. One company, Symbolics Incorporated, has signed up most of the Lisp Machine crew at MIT as employees or consultants and has re-engineered the MIT design for easier construction and maintenance. They expect to ship their first machines in the summer of 1981 for about $150K to $80K, is planned for introduction sometime around the summer of 1982; it is this machine that Symbolics hopes to sell in large quantities. Symbolics has a license to market all of the current MIT software, and plans to augment this software considerably in the coming months. The company appears to be quite well financed, with considerable business and manufacturing expertise, and the chances for their survival appear to be high.

The second company, Lisp Machines Incorporated, is primarily the creation of Richard Greenblatt, one of the key members of the original Lisp Machine group at MIT. LMI plans to offer the Lisp Machine and its software, exactly as it exists in the current MIT version, for about $80K per machine. This leaves LMI with a considerably smaller margin of profit than is traditional in the computer industry; they plan to compensate for this by requiring partial payment in advance and by selling mostly to "sophisticated" users who can handle some of the hardware maintenance themselves. LMI has received some firm orders and hopes to ship their first machine in February, 1981.

Three Rivers Computer Corporation has recently begun to ship their PERQ machines. These machines, priced around $30K-$35K, are considerably smaller than those discussed above, which makes them attractive as editing and office machines but not adequate for serious AI research. To be more specific, the PERQ at present offers only a 256K byte main memory, of which nearly 100K is dedicated to the display. The largest available local disk is is 24M bytes. The microstore is only 4K instructions, compared with 16K of more efficient microstore on the Lisp Machine. There is no hardware page map on the PERQ; memory mapping must be done in microcode. Three Rivers had plans to correct all of these deficiencies sometime in the future, but at present they are very busy trying to produce enough of the current machines to meet the demand for them. Whenever these extensions arrive, the PERQ will become a more interesting option for AI applications.

A group at Bolt, Beranek, and Newman has developed a personal machine called the Jericho. This machine is somewhat less powerful than the Lisp Machine, but it is more

powerful than the current PERQ: it can take any amount of main memory from .5 Megabytes to 2.5 Megabytes, comes with a 200 Megabyte disk, and offers a variety of display options, both monochromatic and color. The machine contains a hardware page map, but this is a simple, single-level map and provides only a 22 bit virtual address space (to the 32-bit word) in the current model. These machines are in use within BBN running PASCAL, and an Interlisp implementation is running but not yet polished. BBN will probably market these machines outside the company within the coming year, but this decision is still up in the air, as is the price.

The Spice project at CMU is an attempt to develop an integrated personal computing environment that will serve the needs of our entire computer science effort, including the traditional areas of CS as well as AI and robotics. One of the novel features of this work is our determination to use only commercially available hardware, and to make our software system portable to any machine that fits our general vision of what personal computers should be. In this way we hope to be able to take advantage of whatever hardware is most attractive at any given time, and to mix different hardware options in an integrated system that presents a consistent environment to the user. Another novel feature is that Spice supports multiple languages and multiple processes on each machine, and ties processes on the same or different machines together with a very flexible set of inter-process communication protocols. When it is complete, Spice will be a relatively portable software system containing a complete Lisp environment (similar to but simpler than Lisp Machine Lisp), a complete Ada programming environment, editors and text processing systems, a multi-media message system, software for a central file system, an extensive user-interface package, and many other features.

Our initial implementation of Spice will be on the PERQ, despite its present limitations for AI work. By the end of 1981, we hope to have a usable first version of Spice running, with an essentially complete (if rather slow) Lisp environment; development and improvement of Spice will continue for several years beyond that date. We plan to move Spice to a more powerful machine, more suitable for AI work, as soon as a few copies of such a machine are available to us. Some companies have expressed a desire to follow our work on Spice very closely; an industrial affiliates program is being set up facilitate the sharing of information with these firms.

A few words are perhaps in order about the specialized needs of workers in robotics and vision Even more than most AI researchers, these people need the real-time response, the microcodability, and the good graphics interface that is provided by the personal machines. Until such systems can be obtained, the only good solution is to dedicate an entire Vax to the people who are doing this work, and to attach some sort of frame-buffer display to the Vax for work in vision In the past, some of this work was done on dedicated PDP-11 systems or on local PDP-11's tied to a larger time-shared processor. Such solutions are extremely awkward in practice. In addition, of course, robotics research requires a first-rate machine shop and an electronics shop, with people who know how to use these facilities properly.

One final consideration raised by the introduction of personal machines is the building that you put them in For any AI research, it is important to have a building that is available (lit and heated or cooled) 24 hours per day, 7 days per week -- even when the contention for cycles is eliminated, many AI people will be nocturnal. But with personal machines on the way, it is important to have a building with adequate wiring and cooling throughout, and not just in a central machine room. It is not yet clear whether the personal machines will work best in offices, in clusters associated with a group of offices, or in a machine room with only the displays distributed around the building, but it would be unwise to lock out any of these options at this point. Rumor has it that IBM is piping Freon for cooling into every office in their new buildings -- can liquid helium pipes be far behind?

## For More Information...

This brief survey has necessarily been superficial. Except in a few cases, I have not even tried to indicate prices, configurations, warranty and service information, or waiting times for delivery In addition, users who are counting on items that are not available now will want to contact the organizations building these items for updates on the progress of the item in question The following list should help you to find the right person to talk to, or at least someone who can tell you who the right person is For readers on the Arpanet, I have also included netmail addresses where these exist. ■

**For Vax/Unix:**
Professor Robert Fabry
Department of Electrical Engineering
and Computer Science
University of California
Berkeley, California 94720
(FABRY @ BERKELEY)

**For Foonly machines:**
Foonly, Incorporated
999 Independence Ave.
Mountain View, Ca 94043
(415) 969-7815

**For Franz Lisp:**
Professor Richard Fateman
Department of Electrical Engineering
and Computer Science
University of California
Berkeley, California 94720
(FATEMAN @ BERKELEY)

**For Decsystem-20, Tops-20, Vax, and VMS:**
Consult your friendly local DEC salesperson.
Interlisp is available through DECUS.

**For Interlisp on VAX:**
Mel Pirtle
University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90291
(PIRTLE @ ISIB)

**For Maclisp and NIL:**
Jon L. White
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, Mass. 02139
(JONL @ MIT-MC)

**For Eunice:**
David Kashtan (technical questions)
Chuck Untulis (administrative questions)
SRI International
Computer Resources
333 Ravenswood Ave
Menlo Park, California 94025
(KASHTAN @ SRI-KL, UNTULIS @ SRI-KL)

**For Lisp Machines:**

Russell Nofsker
Symbolics, Incorporated
605 Hightree Road
Santa Monica, Ca. 90402
(213) 459-6040

Steve Wyle
Lisp Machines, Incorporated
163 N Mansfield Ave
Los Angeles, Ca 90036
(213) 938-8888

**For PERQs:**
Three Rivers Computer Corporation
720 Gross St.
Pittsburgh, Pa 15224
(412) 621-6250

**For Jericho:**
Jim Calvin
Bolt, Beranek, and Newman
50 Moulton St.
Cambridge, Mass. 02138
(617) 491-1850 x4615
CALVIN BBN-TENEXG

**For Spice:**
Scott E. Fahlman
Department of Computer Science
Carnegie-Mellon University
Schenley Park
Pittsburgh, Pa. 15213
(FAHLMAN @ CMUA)

## Search

*(continued from page 6)*

### References


Amarel, S On representations of problems of reasoning about actions In D Michie (Ed), *Machine Intelligence 3* New York: American Elsevier, 1968 Pp 131-171

Feigenbaum, E A, and Feldman, J (Eds) *Computers and Thought* New York: McGraw-Hill, 1963

Fikes, R E, Hart, P, and Nilsson, N J Learning and executing generalized robot plans *Artificial Intelligence*, 1972, *3*, 251-288

Fikes, R E, and Nilsson, N J STRIPS: A new approach to the application of theorem proving to problem solving *Artificial Intelligence*, 1971, *2*, 189-208

Good, I J A five-year plan for automatic chess In E Dale and D Michie (Eds), *Machine Intelligence 2* New York American Elsevier, 1968 Pp 89-118

Kowalski, R And-or graphs, theorem-proving graphs, and bi-directional search In B Meltzer and D Michie (Eds), *Machine Intelligence 7* New York: Wiley, 1972 Pp 167-194

Minsky, M Steps toward artificial intelligence In Feigenbaum and Feldman, 1963 Pp 406-450

Newell, A, and Ernst, G The search for generality In W A Kalenich (Ed), *Information Processing 1965 Proc IFIP Congress 65* Washington: Spartan Books, 1965 Pp 17-24

Newell, A, Shaw, J C, and Simon, H A Empirical explorations with the logic theory machine: A case history in heuristics In Feigenbaum and Feldman, 1963 Pp 109-133

Newell, A, and Simon, H A *Human Problem Solving* Englewood Cliffs, N J Prentice-Hall, 1972

Newell, A, and Simon, H. A Computer science as empirical inquiry Symbols and search The 1976 ACM Turing Lecture *Comm ACM*, 1976, *19*, 113-126

Nilsson, N J *Problem-Solving Methods in Artificial Intelligence* New York: McGraw-Hill, 1971

Nilsson, N J *Principles of Artificial Intelligence* Palo Alto, Calif: Tioga, 1980

Pohl, I Bi-directional search In B Meltzer and D Michie (Eds), *Machine Intelligence 6* New York American Elsevier, 1971 Pp 127-140

Polya, G *How to Solve It* (2nd ed) New York Doubleday Anchor, 1957

Raphael, B *The Thinking Computer* San Francisco Freeman, 1976

Sacerdoti, E D Planning in a hierarchy of abstraction spaces *Artificial Intelligence*, 1974, *5*, 115-135

Samuel, A L Some studies in machine learning using the game of checkers In Feigenbaum and Feldman, 1963 Pp 71-105

Shannon, C E Programming a computer for playing chess *Philosophical Magazine* (Series 7), 1950, *41*, 256-275

Shannon, C E A chess-playing machine In J R Newman (Ed), *The World of Mathematics* (Vol 4) New York: Simon and Schuster, 1956 Pp 2124-2133