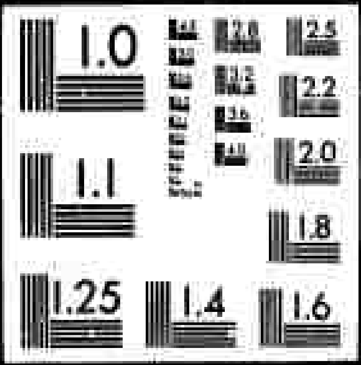
AD
737 531



Security Classification

DOCUMENT CONTROL DATA - R & D	
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)	
1. ORIGINATING ACTIVITY (Corporate author) Carnegie-Mellon University Department of Computer Science Pittsburgh, Pa. 15213	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP
3. REPORT TITLE C.ai: A COMPUTING ENVIRONMENT FOR AI RESEARCH	
4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)* Scientific Interim	
5. AUTHOR(S) *(First name, middle initial, last name)* G. Bell, P. Freeman, M. Barbacci, S. Bhatia, W. Broadley, R. Chen, L. Erman, H. Goldberg, W. Huen, M. Knudsen, D. McCracken, A. Newell, R. Reddy, S. Rege, G. Robertson.	

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
May, 1971	55	none

8a. CONTRACT OR GRANT NO. F44620-70-C-0107	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO A0827-5	
c. 61101D	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d. 681304	AFOSR - TR - 72 - 0379
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.	
11. SUPPLEMENTARY NOTES TECH, OTHER	12. SPONSORING MILITARY ACTIVITY Air Force Office of Scientific Research (SRMA) 1400 Wilson Blvd. Arlington, Va. 22209

13. ABSTRACT

A computer for artificial intelligence research is examined. The design is based on a large, straightforward primary memory facility (about 8 million 74 bit words). Access to the memory is via at least 16 ports which are hardware protected; there is dynamic assignmentof the memory to the ports. The maximum port bandwidth is 8,600 million bits/sec. Processors for languages (e.g., LISP) and specialized terminals (e.g., video input/output) can be reliably connected to the system during its operation. The approach is evolutionary in that high performance processors, such as the Stanford AI Processor, can be connected to the memory structure, giving an overall power of at least 100 times a PDP-10 (and 200 to 300 times a PDP-10 for list processing languages) for 10 processors -- although 20 processors can be attached. Using this approach we might expect 40 - 80 million operations/second.

At the same time, special language processors (P.$\mathcal{L}$) can be designed and attached. These processors give even larger power increases, but for restricted language use. Two processors, P.LISP and P.L*, were examined for the LISP and L* languages and are reported on separately.

A plan for building the machine in increments over the next three to five years is examined. Specific schedules are proposed.

Concurrent with the operation of the machine, there should be research into the design of hardware, software and theory of constructing large scale computing facilities with maximum modularity.

DD FORM 1 NOV 65 1473

Security Classification

C.ai: A COMPUTING ENVIRONMENT FOR AI RESEARCH

Overview, PMS and Operating System Considerations

by

G. Bell (Co-Chairman)

P. Freeman (Co-Chairman)

and

M. Barbacci
S. Bhatia
W. Broadley
R. Chen
L. Erman
H. Goldberg
W. Huen
M. Knudsen
D. McCracken
A. Newell
R. Reddy
S. Rege
G. Robertson

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213

[illegible]

May 7, 1971

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research. ~~This document has been approved for public release and sale; its distribution is unlimited.~~

TABLE OF CONTENTS

ABSTRACT

A computer for artificial intelligence research is examined. The design is based on a large, straightforward primary memory facility (about 8 million 74 bit words). Access to the memory is via at least 16 ports which are hardware protected; there is dynamic assignment of the memory to the ports. The maximum port bandwidth is 8,600 million bits/sec. Processors for languages (e.g., LISP) and specialized terminals (e.g., video input/output) can be reliably connected to the system during its operation. The approach is evolutionary in that high performance processors, such as the Stanford AI Processor, can be connected to the memory structure, giving an overall power of at least 100 times a PDP-10 (and 200 to 300 times a PDP-10 for list processing languages) for 10 processors -- although 20 processors can be attached. Using this approach we might expect 40 - 80 million PDP-10 operations/second.

At the same time, special language processors (P.ℓ) can be designed and attached. These processors give even larger power increases, but for restricted language use. Two processors, P.LISP and P.L*, were examined for the LISP and L* languages and are reported on separately.

A plan for building the machine in increments over the next three to five years is examined. Specific schedules are proposed.

Concurrent with the operation of the machine, there should be research into the design of hardware, software and theory of constructing large scale computing facilities with maximum modularity.

ACKNOWLEDGMENTS

The basic PMS structure of C.ai and that of the L* and LISP language processors was developed in the Carnegie-Mellon University Computer Science Department, in a project seminar on list processing machines. This report is based on the working papers developed in the CVAM seminar.

The group was informally organized and met frequently as a whole, but the main work for the reports was carried out in the following sub-groups:

CVAM:PMS - W. Broadley (Chairman), S. Bhatia, and S. Rege

Specification of PMS, circuitry, technology, and overall performance.

CVAM:OS - P. Freeman (Chairman), S. Bhatia, L. Erman, W. Huen, D. McCracken, R. Reddy and G. Robertson

Specification of an operating system.

CVAM:P.L* - D. McCracken and G. Robertson (Chairmen), R. Chen

Design of a hardwired processor for an L* kernel.

CVAM:P.LISP - M. Barbacci (Chairman), H. Goldberg, and M. Knudsen

Design of a hardwired processor for LISP.

Many members of the Carnegie-Mellon Computer Science Department served as critics and consultants on the design, especially A. Kendziora, J. McCredie, A. Newell, R. Reddy and W. Wulf. Alan Kay of Stanford also interacted with the group frequently and in detail.

A draft of this document was presented at a working meeting held at Bolt, Beranek and Newman in Boston, April 26-27, 1971. Representatives from the Advanced Research Projects Agency, Bolt, Beranek and Newman, Carnegie-Mellon University, Massachusetts Institute of Technology, Stanford Research Institute, Stanford University, and System Development Corporation were present. Their comments and suggestions were of value in preparing this final version.

Gordon Bell and Peter Freeman integrated the working papers into this document and did the final editing.

OVERVIEW OF THE SYSTEM

Consideration of the problem of designing and building an optimal computer for ai research quickly leads one to the realization that there may not be a feasible solution. The numerous constraints, wide variations in computing style, and the impossibility of defining the ai problem narrowly seem to make this a certainty. Thus, the major premise of the design we are about to present is that if one wishes to provide ai researchers with better computing tools, one must, in fact, provide an environment in which many varied tools may be developed and used. Our design should be viewed as a specification of such an environment.

REQUIREMENTS FOR AI COMPUTING

In Bell and Newell's Computer Structures (McGraw-Hill, 1971)* a number of special function computers ranging from business to scientific are described. The characteristics of machines used for ai research appear to span this spectrum, exhibiting the max of each characteristic attribute.

Memory Size

The primary memory is larger in an ai environment than with almost all scientific computers because the local program data base is typically larger than for scientific applications. Here we assume that the average program size in this environment is 250,000 74-bit words (64 bits of

*The PMS notation presented therein and used throughout this report is based on seven primitive component types: P-processor, M-memory, S-switch, L-link, K-controller, T-transducer, D-data operation. A computer composed of primitive components is represented by C; hence, C.ai for "ai computer."

information) and the working-set size is 100,000 74-bit words. The ratio of secondary/primary memory we have proposed is quite low, based more on the swapping mode of current PDP-10's, rather than a demand paging system. A better model is needed to get a "balanced" or more "cost-effective" system rather than take this worst case approach.

Processor Power

The central processor power requirement is less than the largest scientific processor because the very large memory is accessed sequentially (in relatively inefficient fashion). Thus, the more powerful facilities of a scientific processor may go unused. Many ai processes are compute bound, however, and need significant processor power in critical areas.

PMS Structure

The ai computer usually requires better communications facilities to external equipment than either a scientific or a business computer. Typically, these characteristics are like the process control computer, except that higher data rates are involved for speech, video and mechanical transducers which operate at human interaction rates. These considerations are particularly critical for ai research groups engaged in robot, hand-eye, and speech research.

The Instruction Set (ISP)

Typically, the ISP is the characteristic that usually comes to mind when ai computing is discussed. These operations can be for a hardwired stack, a garbage collector, hash-coding instructions on linked lists, etc. On the other hand, a simple processor with good floating point arithmetic seems to be adequate because decisions can be bound in software and later changed.

There is almost uniform agreement (at the meetings on the ARPA list processing machine) that a large (and probably linear) addressing space is essential. The other noticeable point of agreement is that the PDP-10 instruction set with only a few modifications is all that is needed for the immediate future.

There is clearly a need for general-purpose processors with clean order codes and specialized instructions for characteristic ai processing operations. But, in addition, the magnitude and closely defined nature of many ai tasks makes it very desirable to have highly specialized processors as well.

The PDP-10 As The Current ai Computer

Because the DEC PDP-10 is the current most popular machine for ARPA-supported ai research, it is worth briefly examining the reasons for its popularity.

1. Price: It is the only computer that the group can afford. (A 360 Model 44 is also a candidate in this range that has been overlooked and it is worth asking why.)

2. ISP: The instruction set is very straightforward, providing power but with none of the anomonalies in addressing, instruction set size, data types, etc. that accompany most machines (e.g., 360, 1108 or Sigma 7 family).

3. PMS Structure: The machine has been easily approachable by all to interconnect any kind of device from foreign memory to TV camera. Indeed, the PMS structure, now eight years old, is only being slightly revised to handle larger and faster memories. This ease of interfacing was initially used internally by DEC to administer various memory and peripheral designs; later, this policy boomeranged by allowing anyone to connect any kind of memory to a PDP-10. In contrast, the IBM processor-memory interfaces are usually so well guarded and obscure that

its own engineers can barely carry out designs (e.g., the bus control unit in the model 65, 67 and 75).

4. Operating System: The monitor is small, relatively approachable and less bad than alternatives since it was based on CTSS and the SDC time-sharing systems.

RESPONSE TO THE REQUIREMENTS

It is difficult if not impossible to obtain a more accurate characterization of what is meant by "ai computing" than what we have given above. We have not been able to reach any agreement among ourselves or with those with whom we have talked. It is clear that we should have a better characterization; and we have stressed the need for measurement facilities herein so that in the future we will better understand the nature of our computing requirements. The diversity, of course, is such that ai computing is really equated with general computing. And therein lies the crux of the problem.

Our response has been to design an environment that will demonstrably improve by the needed factors our current ai computing resources and, more importantly, provide a context for the development of even more powerful tools. In particular, it is clear to us that various groups must be able to experiment with highly specialized processors (e.g., L*, signal processing, etc.) in order to gain maximum power from the hardware. Yet this experimentation must be evolutionary to guard against the transients caused by switching everyone to a new machine at the same time.

This report examines the feasibility of designing a relatively large-scale, simple computer which can provide such an environment for artificial intelligence research. Several design goals that guided us are posited in Appendix 1. The resultant design has the following characteristics:

1. Mp - A simple primary memory structure with 16 multiple ports for connecting a variety of high speed processors of the Stanford AI processor design, special language processors, secondary memories and specialized i/o transducers.

    a. A primary memory size of 620 megabits with individual port rates of 74, 148, 222 or 296 bits/port memory access (550 ns); 135, 270, 405, or 540 megabits/sec per port;

    b. A total information rate to all ports of 2.1 to 8.6 gigabits/sec;

    c. Memory port widths of 72+2 parity or 64+10 single error correction and double error detection bits;

    d. The 16 ports can be further demultiplexed to provide more ports;

    e. Each port has a memory port control for dynamically reassigning 64k word blocks to each processor and protecting the memory from neighboring processors. The port control also includes statistics data gathering and error control.

    f. A mode of operation which provides nearly 100% uptime.

2. Ms - Secondary memory bandwidth to allow swapping. The loading time for a single 100,000 word program would be roughly .14 sec. The worst case swap time would be .3 seconds. Thus, assuming five drums, 15 programs could be swapped per second.

3. P - Multiple approaches for providing processing power. These range from a conventional PDP-10 to specialized hardwired list processors. This permits development of highly functionally specialized processors.

3a. PDP-10 central processors (KA10) modified to run the BBN Tenex operating system could be used.

3b. The forthcoming DEC KI10 processor would provide at least a factor of 2 improvement over the KA10.

3c. The Stanford AI processor (basically a PDP-10) could easily be used and would provide:

   a. A primary memory per processor of .8 million 74 bit words assuming 10 processors, hence essentially a factor of 8 times the memory of a maximum size PDP-10.

   b. An instruction execution rate of roughly 10 times a PDP-10 for 1 processor or 100 times a PDP-10 for 10 processors.

   c. Assuming a specially microcoded LISP interpreter or compiler in the processor, the execution rate would be a factor of 200 to 300 times a PDP-10.

   d. A secondary memory rate of about 250 megabits/sec versus 10 for a PDP-10.

3d. An alternative design which we also discuss herein is incorporated in the overall framework using an approach based on specialized language processors (P.ℓ) and processors for the operating system. In this report we assume the use of a combination of conventional processors of the Stanford variety and special language processors. The mixture is unimportant, and the overall performance for list processing may only vary by a factor of 2 to 4.

Two language processor designs have been explored for interpreting the LISP and L* languages which show a speed-up of 20 to 40 times a single PDP-10 processor or 200 to 400 times a PDP-10 for multiple processors. The cost of these processors appears to be quite low ($50,000-$100,000 each). They are described in two reports of the CMU Computer Science Department: "C.ai (P.L*) -- An L* Processor for C.ai," D. McCracken and G. Robertson, and "C.ai (P.LISP) -- A LISP Processor for C.ai," M. Barbacci, H. Goldberg, and M. Knudsen. The abstracts of these reports appear as Appendices 2 and 3 of this report.

3e. Another alternative would be to construct a central processor to give an instruction encoding improvement over the PDP-10. For example, a processor based on the stack and instruction format of the PDP-11 might be desirable. This approach may allow high performance processors to be built more easily.

4. Modularity in PMS structure. Here we hope to do significantly better than the PDP-10 by providing better protection among the processors at the memory ports. We believe that it will never be necessary to turn off computer power for any reason (except for cooling equipment failure).

5. Uniform interprocessor communication. A second type of interface has been added which has a protocol like the PDP-11 and allows intercommunication among the various processors. Like the PDP-10 i/o and memory busses, this type bus should be able to be used over a significant period of time. It allows the transmission of data at high rates. Unlike most other busses,

this type of bus (called Unibus in PDP-11) allows all components to be at the same level of the hierarchy; thus, all components communicate with one another easily.

6. In the design of the language processors, we have discovered that multiple caches can be used effectively. The language processor designs each use two caches, giving us roughly a factor of two gain in parallelism within the processor. One cache is used to hold the interpreter (data which is only read) and might normally have been called the microprogram memory. The second cache holds the program (instructions and data) being interpreted. A third cache might possibly control an execution unit for, say, typed data, etc.

7. The operating system is evolved from our current operating systems. The main assumptions are that the network will carry out many of the operating system functions. The remaining functions will be distributed throughout the system on various specialized software controllers (e.g., at file swapping, memory and tertiary memory sites).

8. The facility should be an ARPA network resource, but with surrounding laboratory facilities to make the operation of the computer reliable, but yet extensible. This mode of operation is similar to current usage at many ARPA contractor sites. For example, using the memory for video input and output, experiments would require only 45 megabits/sec port to memory. Two resources would be required.

a. A large inventory of modules that would facilitate design of special experiments. The module types would include: caches, general purpose microprogram controls, arithmetic units, buffers (queues), port switches (to increase the number of memory ports), mapping, interfaces to other computers, component exercisors, etc.

b.. Programmers and engineers to carry out many of the designs and assist in system integration of the devices.

The design as proposed would operate at a single central site to give large memory and to decrease the operating cost. However, the design and the construction strategy are such that at almost anytime in the project the machine could be partitioned physically for multiple site operation. The most practical size for economy and reliability would no doubt require at least two processors and 1 to 2 million words of primary memory.

9. The system could be operational within one year at a central site. At this time although any amount of primary memory could be available, only two PDP-10 Tenex processors need be available. Over the next few years, more processors and memories could be added.

10. Research should go along with making such a modular system laboratory. The general direction would be to explore hardware, software, and theory that made the interconnection of modules of the above type easily interconnected. In this way, a system of any type could be constructed easily.

THE PMS STRUCTURE OF C.ai

Figure 1 shows a simplified PMS diagram of C.ai. It is a multiprocessor system which allows up to 16 processor or secondary memory (e.g., drum) ports to be connected to the primary memory. Some of the ports can be multiplexed between several components. The characteristics of the primary memory are: MOS; 550 ns cycle time; 2^{21} 296-bit words accessible as 74, 148, 222 or 296 bits in a single access. The memory will be described in more detail below.

The 16 memory modules are interconnected to the 16 processor ports via a central 74-bit cross-point switch. The main reason for a central cross-point is to limit the cables from $p \times m$ to only $p+m$. Also, by using a non-bus arrangement, processor and memory modules can be removed while the system is operating. It must be noted, however, that the weakest link in the design of C.ai is the switch, and we are beginning to investigate this design. It will most likely be a form of the dual-cross-point switch (see Bell and Newell, Computer Structures, McGraw-Hill, p. 68).

Current logic technology is ideally suited to the packaging of a centralized switch. Although it is centralized, the physical packaging can be carried out to provide independence among the processor and memory ports. Logically, the memory controls and the processor controls are quite independent. By partitioning the switch into four 8x8 switches, even more independence can be gained.

Another type of switch is used to provide intercommunication among the processors. This is shown in the lower part of the figure. This

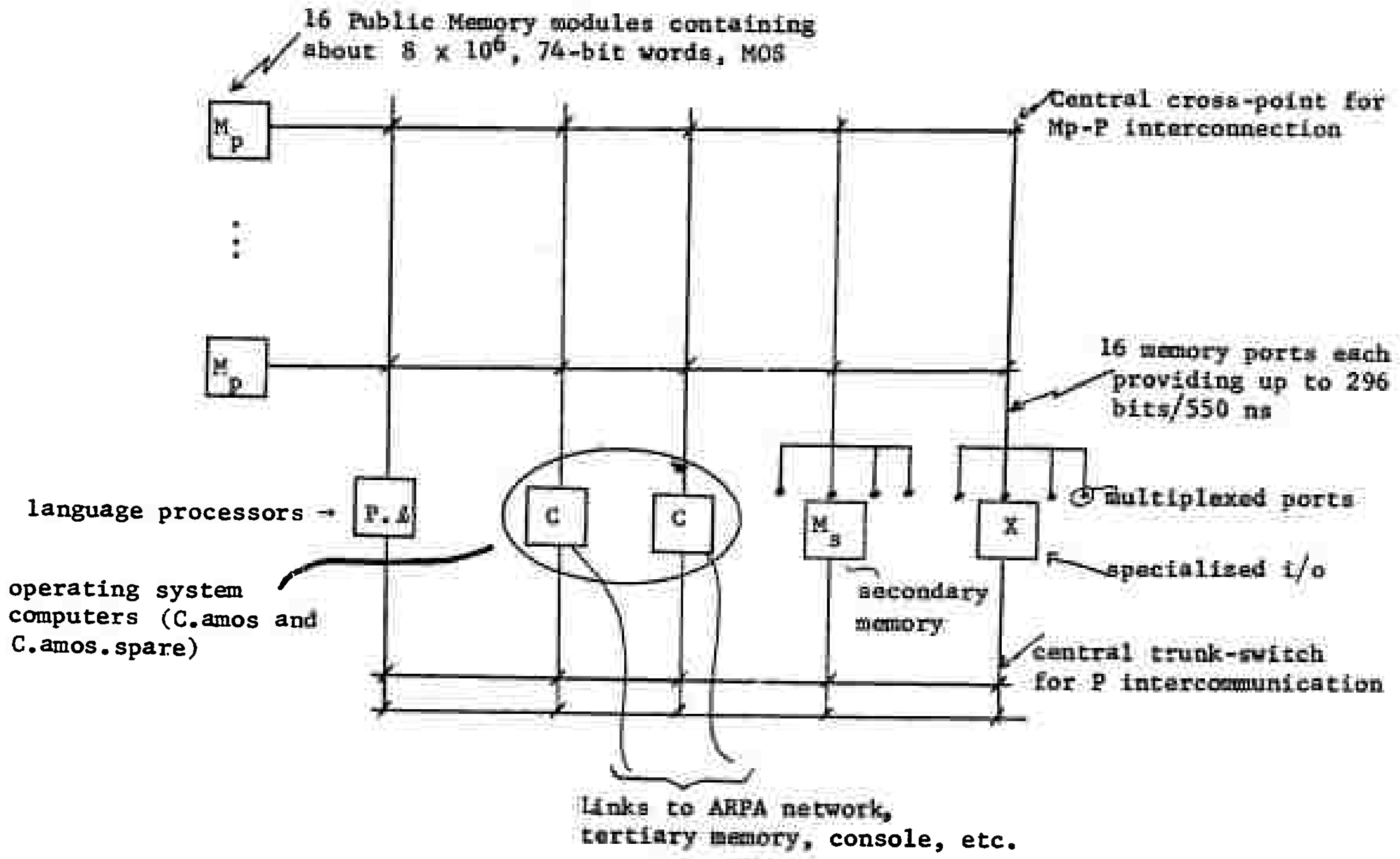
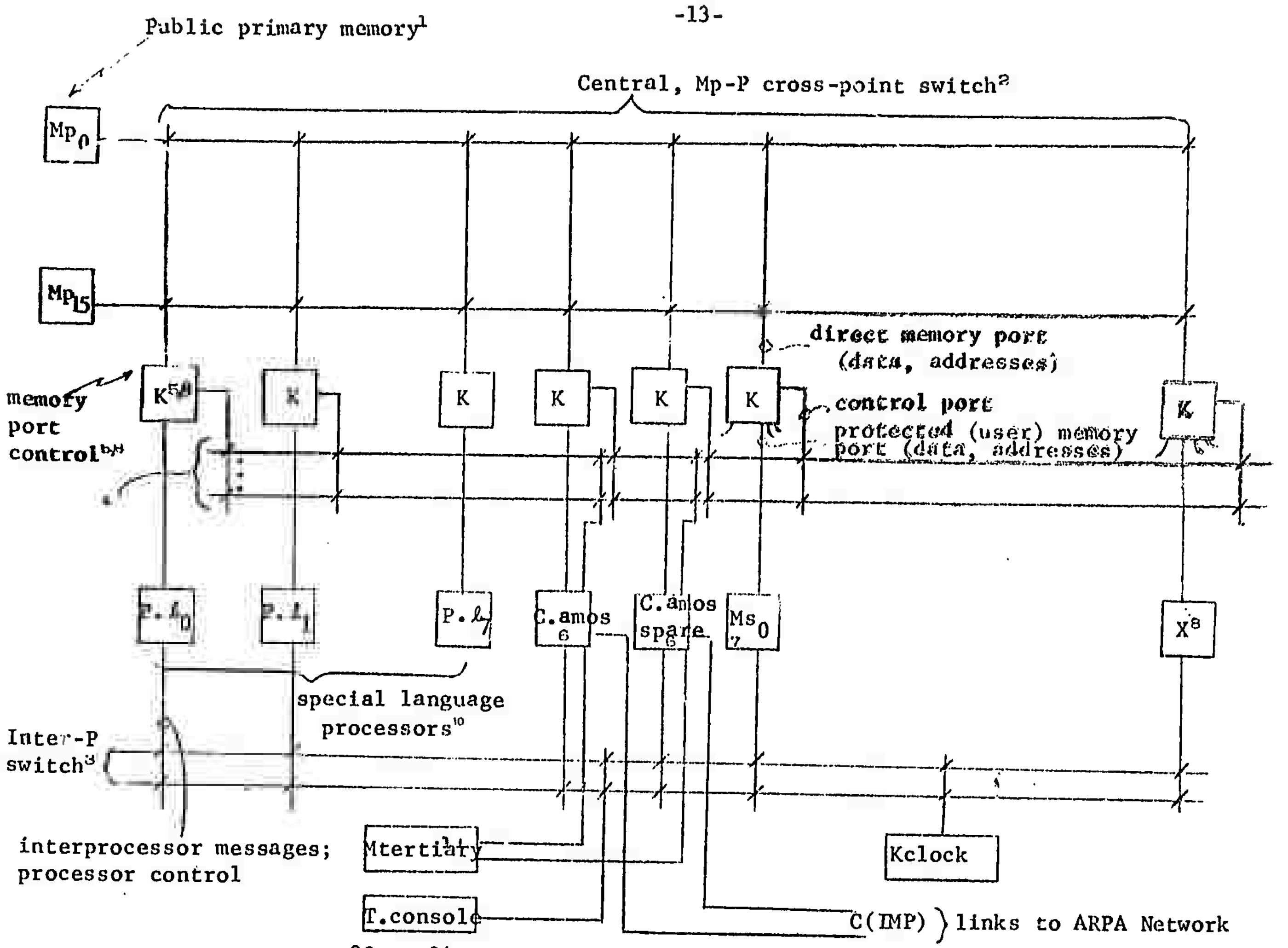

16 Public Memory modules containing about 8 x 10^6, 74-bit words, MOS
Central cross-point for Mp-P interconnection
M p
M p
16 memory ports each providing up to 296 bits/550 ns
language processors →
P.Δ
C
C
M s
X
multiplexed ports
specialized i/o
secondary memory
operating system computers (C.amos and C.amos.spare)
central trunk-switch for P intercommunication
Links to ARPA network, tertiary memory, console, etc.

Figure 1. PMS diagram (simplified) of C.ai

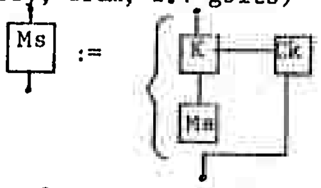
is a centralized trunk switch with an adequate number of trunks to handle the traffic among the various components which intercommunicate. The switch is described in more detail below.

A central computer (C.amos) executes the operating system. It has some private memory, interconnection to i/o devices, terminals, and tertiary memory. Two C.amos computers are shown, but only one will be in use at any time (see below). The other processors are general-purpose or specialized language processors. Each of these may have a minicomputer which acts as the control for starting and stopping, maintenance, data gathering, context switching, etc.

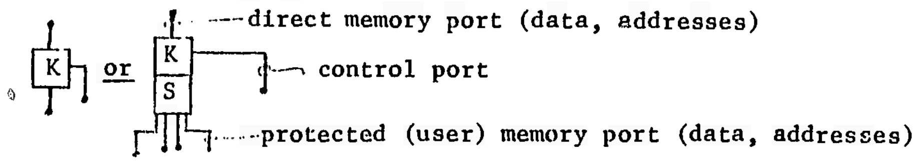
Figure 2 gives a PMS diagram that has sufficient detail for deriving performance characteristics of the computer. We assume each processor will cost in the neighborhood of $50,000 to $100,000. The critical parameter for determining the performance is the number of memory ports and the processor operation-rate, so that the interference among the processors can be determined. Each processor is able to obtain up to 296 bits in 550 ns (or 540 megabits/sec). For example, the demand for memory that a PDP-10 has is roughly two words, each 3.5 μs at 300,000 op/sec. Thus, a word can be used each 1.25 microseconds or its port needs only 28 megabits/sec. The above system supplies roughly 20 times this amount; eight words/access and a cycle time of 550 ns contribute to this gain. Since the eight words are accessed at one time, no use can be made of the extra words in a normal PDP-10 processor. The Stanford AI Processor, however, does use up to four 37-bit words in a single access to fill its cache.



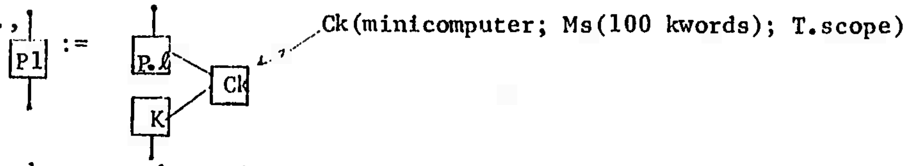
[1] Public, primary memory; $2^{23} \sim 2^{24}$ words; 74, 148, 222, or 296 b/w; 550 ns/w; MOS
[2] Central; cross-point switch; Mp-Pc|Ms dialogues; 16x16; width: 74 bits
[3] Central, trunk switch; inter P dialogues; ≥ 2 trunks } identical protocols
[4] Central, trunk switch; P-K(Mp.port) dialogues; ≥ 2 trunks
[5] K(Mp.port) Relocation, protection, error correction, error detection; see[9]
[6] C(Operating-System) - with local primary and secondary memory
[7] Ms(secondary memory; drum; 1.4 gbits)



[8] X(special i/o interfaces; e.g., TV)
[9]


K
or
K
S
direct memory port (data, addresses)
control port
protected (user) memory port (data, addresses)

[10] E.g.,



[11] Transducers and tertiary memory - managed by a Ck (e.g., terabit memory)

Figure 2. PMS diagram of C.ai

By using a cache memory the bandwidth into primary memory is significantly reduced. In the Stanford processor there is about 95% of the data in the cache. Executing 4 million instructions/sec, at most, one word would be required per 125 ns (i.e., 8 million words/sec). Now since 95% of the data is in the cache, the requirements for primary memory access are only one access each 20 memory accesses. Thus the effective memory cycle time is only one word each 2.5 microseconds. The interference among ports is therefore negligible. At the very least, i/o devices, such as the drums, could share a port. In one table we have shown a second processor connected to a port (for a total of 20 processors).

COMPUTER PERFORMANCE

Table 1 shows a comparison of C.ai performance with some current large scale computers. Various attributes of these machines are given in order to give the reader an idea of the balance of the computer in terms of memory size, processing capacity and cost. We have also included the measures Roberts* has used to compare the performance with these machines. In some sense, the chart is extremely misleading since C.ai has 20 times the memory of the next largest machine (STAR). This has been adjusted in footnote 3 to the table. The computation is based on 36 bit operations. Using a larger word would increase the performance indicator, though perhaps not the real performance.

INTERPROCESSOR COMMUNICATION

Interprocessor communication will be carried out over a data bus similar to DEC's PDP-11 Unibus, with the exception that it would be a

* Lawrence G. Roberts, Data Processing Technology Forecast, Advanced Research Projects Agency, April 23, 1969.

Table 1. Comparison of C.ai with Other Computers

	Mp.width (b/w)	Mp.size (mwords)	Mp.size (mbits)	Mp.i-rate (mwords/sec)	Mp.i-rate (mbits/sec)	Ms.i-rate (mbits)	Pc.i-rate (mop/s)	Cost(m$);Bitxops/sec;[2] Bitxops/sec/$
PDP-10	36+1	0.26	9.7	4	144	9	0.4	1; 18; 18
Stanford AI-10	144+4						4.0	1; 180; 180
Model 91	64+8	0.52	37	21	1370	10x(1~2)	6.0	7.7; 500; 65
CDC 6600	60	.26	15.4	32	1920	12x10 (i/o) 600 (ECS)	3.0	5.5; 145; 27
C.ai	74~296	8.3	620	29~120	2150~ 8600	50x5	(4~12)x10 to (4~12)x20	13;1440~4320; 110~330 16;2880~8640; 180~540
C.ai/4	74~296	2.3	155	8~30	504 ~ 2150	50x2	(4~12)x3 to (4~12)x6	
CDC STAR	512	.131	74	25	12,800	(50~100)x (1~5)	100	10; 3200; 320
ILLIAC IV[1]	64+	.131	8.2	64x4.1 261	16,800	1000	256	10; 8200; S20

[1]Specifications taken from early ILLIAC IV paper by Barnes, et al.
[2]L. Roberts - Data Processing Technology Forecast, April 23, 1969.
[3]Assumes $8m for memory, $2m for peripherals and $300K per processor. Adjusting the memory size to that of STAR, yields $7m (total); 1440 ~ 4320; 205 ~ 620 to $10m; 2880 ~ 8640; 288 ~ 864.

dual or multiple trunk bus to increase bandwidth, decrease response time and increase reliability. A processor making an interprocessor transfer would place a request on the bus and the actual transfer would take place on the trunk that first responded. Each message will be tagged with the identity of the transmitting processor. It will useful if a processor can communicate with itself on the bus.

If the proposed interprocessor traffic appears to warrant it, more than two trunks can be added. In addition, processors can communicate through shared primary memory.

PRIMARY MEMORY

The characteristics of various memories are given in Table 2. In this section we are only concerned with primary memory, however.

System Configuration

The primary memory Mp(MOS; 550 ns cycle; 2^{21} wds; 296 b/w gives an overall memory size of about 620 X 10^6 bits and (because of the 16 ports) a bandwidth of 8,600 million bits/sec (16 X 296/.550). The access time, as measured at the processor, would be about 350 ns. The actual memory word will be 296 bits with a processor specifying which word (74 bits) or configuration of words it wants. The actual information bits are expected to number 64 with 10 bits to be used for error correction or detection at the processor. The 74 bit word would also allow present 36-bit processors, such as the PDP-10 and 1108, to utilize this memory. The memory will consist of 16 modules and will connect to the processors through a 74-bit wide cross-point switch. There will be 16 processor

Table 2. C.ai Memory Characteristics

Device	function[3]	t.access (t.cycle)	i.rate mb/s	i-width (bits)	total i.rate (mb/s)	power (watts)	floor space ft^2	size module 10^6b	cost (k$)	cost/bit ¢/b	controller costs (k$)
photomemory	t							$\geq 1\times10^6$			
moving-head disk (e.g., Calcomp)	t	0~55 + 20 ms	3.3[2]	1x3	3.3x3 10	30 kw (15 kw/ unit)	10x20 = 200	200 x20 = 4000	300	0.0075[2]	300
drum (e.g., GI)	s	8 ms	3.3	8~16/ drum	50 drum	20 kw (1 kw/ unit)	10x20 =200	70x20 =1400	900	0.048	5x40 200
shift register (AMS)	s	131 µs 780 µs	1	296	296	200 kw		1400	1800	1.25 0.7	5x40 200
core (Ampex)	p	.7 µs (1.8 µs)	.55	296 x16	163 2605	200 kw		620	10,240	1.65	500
Lockheed core	p	275 nsec (650 ns)	1.54	296 x16	455 7286	200 kw		620	8070	1.3	500
MOS AMS	p	500 nsec (1.2 µsec)	.83	296 x16	246 3947	200 kw		620	5760	.93	500
MOS (AMS)	p	250 ns (400 ns)	2.5	296 x16	740 8600	200 kw (.2 mw/ bit)		620	9500[1]	1.52	500
Bipolar (Cogar)	i	40 ns (80 ns)	10	296	2960			100 ~ 2000 words	36	6	
Bipolar ROM	i	40 ns (80 ns)	10	50~296	2960			100 ~ 2000 words	12	1 ~ 2	
									11,049		1000

[1] Estimate - Cogar has quoted such a system at $13,100K for delivery in 1973.

[2] These assume current densities. We can safely assume double density, hence lower cost, more storage, and higher transfer rates.

[3] t/tertiary; s/secondary; p/primary; and i/internal processor (program control, accumulators, etc.)

ports each 74 bits wide. A transfer of more than one 74-bit word in an access will be done sequentially at a rate of 75 ns per additional 74 bit word for up to four words.

Slower Memory Considerations

Although we have assumed relatively fast MOS memories, slower memories can be used since the proposed processors have caches. Our preliminary investigation does not show a significant price decrease as MOS memory (or other memory) speed decreases. Actually, the switching system should be designed to operate at a high data rate in order to use later, faster technology memories as they become available.

Processor Port Control - K(Mp.port)

The local port control is shown in Figure 3. Each processor port provides access to 2^{24} words (a 24 bit address). The upper 7 bits of the address specifies one of the 128 relocation registers the address will use. The relocation register will supply the high order bits of the physical address and the processor address will supply the low order 17 bits. Since this is a concatenation instead of an addition, this relocation should only take 50 ns. The relocation registers and other controls associated with the port are accessible only to the overall operating system. Various protection type bits might also be included in the memory port control box to assist the processor. The relocation unit serves these functions: maintenance, dynamic memory assignment (reconfiguration) control, protection and sharing among processors, and data parity checking.

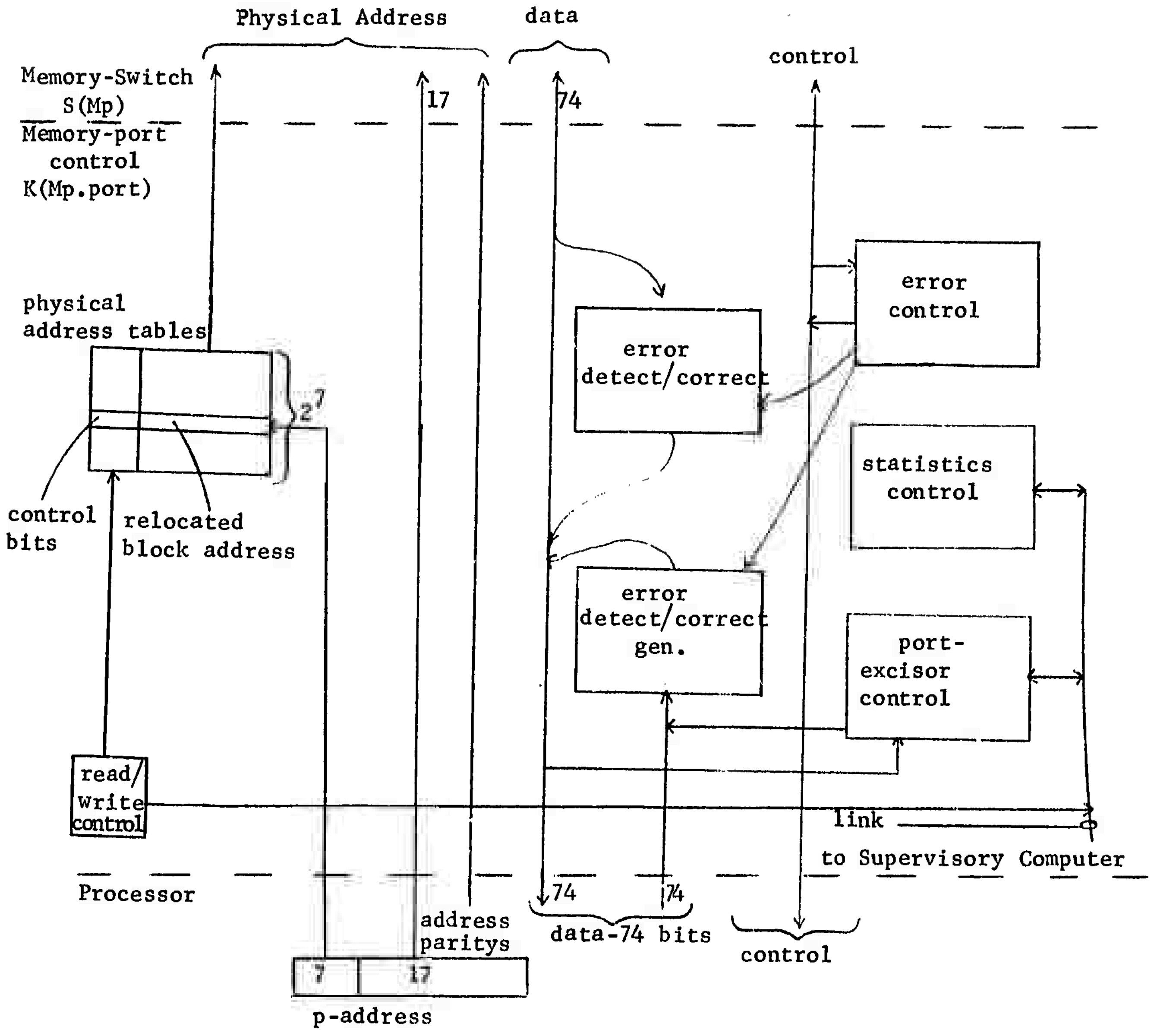


Figure 3. K(Mp.port) Memory-port relocation, error-detection, error-correction and control.

The relocation registers will be transparent to each processor. The relocation registers will serve the function of manual, address switches on the memory (e.g., on the PDP-10). Thus, no manual switching need be provided on the individual memory boxes; the same effect is achieved by informing the memory control processor to vacate the desired box and consider it unusable. All relocation and protection, as commonly found in timesharing system processors, will be included as part of the processor. The only reason a processor might want to consider the port relocation registers is to effect 65k word block transfers, i.e., to ask the memory control processor to change addresses, e.g., 128k-192k to 64k-128k.

The statistics control shown is passive. Although not detailed here, it will be hooked up to provide appropriate information on accesses, errors, and transfer rates to a measurement unit. Another part of the port interface is the capability of being exercised at low data rates via the controlling computer. Thus, data can be transferred via each port from the control computer. Within each port control there is error correction, detection hardware. Here, since we assume that some faulty processors will be attached to the ports, the port control must supply correct data to the memory in order that other devices (e.g., drums, disk memory) be able to detect faults.

Circuitry

Currently, Advanced Memory Systems (AMS) is quoting approximately $160,000 for 256k 37 bit word memory system with an access time of 150 ns and a cycle time of 400 nanoseconds for delivery in 1972. At those rates the 8,192k word memory (74 bits) would cost approximately $9,500,000. This is a projected price, but it is possible that memory prices will decrease even more than projected. AMS is the lowest bidder, but there is some question as to whether they could manufacture a 620 megabit memory in the time frame desired in addition to their current commitments. Cogar has quoted a price of $13,100,000 for a similar system for delivery in 1973. The memory prices are quotes for memory systems with TTL-compatible interfaces. The power dissipation for the AMS system is 180 milliwatts per 1024 bits or approximately 100kw for proposed system; support circuitry will double this to 200kw.

The memory switch will utilize MSI (medium scale integration) logic whenever possible. The data switch is currently envisioned as utilizing the Texas Instrument SN74150N 16 bit multiplexor with a typical data propagation time of 10 nsec (assuming the data select lines have been settled for ≥ 30 nsec). By the time the switch is constructed, faster circuits, such as Schotky TTL, will probably be available.

Since the switch is so central, a dual cross-point probably should be used. This is essentially a compound switch consisting of two cross-point switches and a (m+p) X S(1-input, 2-output) switch as shown:

$$\begin{tikzcd}
 & \text{S(cross-point; m}\times\text{p)} \arrow[dr, dash] & \\
\text{Mp } \underline{\text{m-inputs}} \text{ S(m;2)} \arrow[ur, dash] \arrow[dr, dash] & & \text{S(p;2) } \underline{\text{p-inputs}} \text{ P} \\
 & \text{S(cross-point; m}\times\text{p)} \arrow[ur, dash] &
\end{tikzcd}$$

The memory switch itself should not exceed $200,000 in cost (parts and labor). A quick calculation shows that in just the data and address switching logic, 2800 SN74150N 16 input multiplexors would be needed at a cost of $19,000. This does not include any support circuitry such as encoders, line drivers, printed circuits, etc. Through future price reductions and competitive bidding, we might be able to reduce the cost of the SN74150N's to $7,000.

SECONDARY MEMORY

The choices for secondary memory are (1) mechanical devices (drums, fixed head discs, etc.), (2) shift register memories, (3) and random access memory (RAM). Current characteristics for these devices are shown in Table 2. Mechanical devices will probably hold a price advantage of an order of magnitude for several years. It doesn't appear that shift register memory will become sufficiently cost effective over random access memory.

The following figures assume a 2 to 1 swapping ratio. A secondary memory system could consist of 20 General Instrument SA 7012-1024-H drums at $45,000 each for a total of $900,000, in quantities, exclusive of controller and channels. Judging from smaller, more modest systems, the secondary storage system should cost approximately $1,300,000 complete. Such a system would give 1.400 megabits of storage, an average access time of 8 msec, and a transfer rate of 50 megabits/second.

Initally, the Ms controllers will simply permit multiplexing several drums into one port. An additional feature that could easily be included in the secondary memory channel would be a memory-to-memory connection that would be able to take advantage of the 4-word sequential feature of the primary memory. Since one wishes to maximize the bandwidth between the Ms system and Mp, as the system grows to use many drums on several ports, it will probably be necessary to insert a C(Ms). The Ms - Mt system might then look like Figure 4.

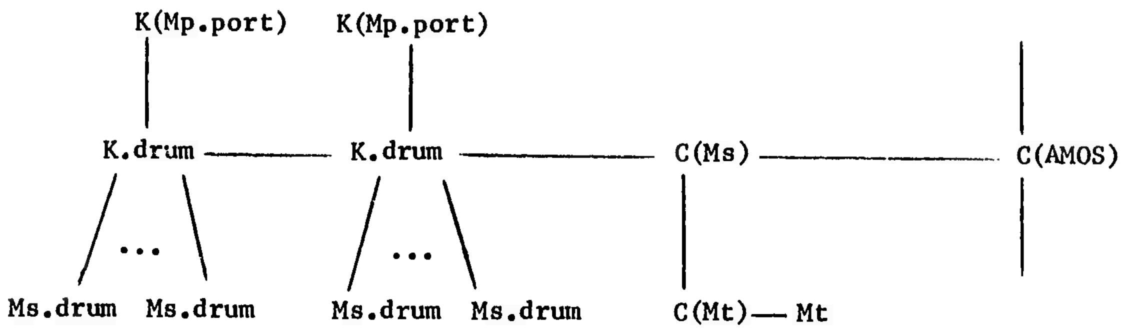


Figure 4. Eventual secondary-tertiary memory structure.

TERTIARY MEMORY

Any storage of programs and data outside the local C.ai environment will tend to put a fairly heavy transmission load on the current, 50,000 kb ARPA network. For example, we have assumed average program sizes of 250,000 74-bit words or 18.5 million bits. The transmission of a program this size on the present ARPA network would require approximately 460 seconds. If a session lasted an hour, about 15 minutes of the hour would be spent in file transmission. If 4 such users were on the system, then the whole ARPA network would be swamped.

Clearly, some on-site permanent storage must be provided. Programs can reside on a tertiary memory until they are brought into either primary or secondary memory for more rapid access.

OPERATING SYSTEM COMPUTER

C.amos is the ultimate overlord of all physical resources in CVAM. It should have the following characteristics (among others):

- off-the-shelf existence;
- low cost and plugability to permit keeping an on line spare;
- good interrupt structure;
- already debugged network control program (if possible);
- sufficient speed;
- word size commensurate with 64;
- ability to address all of C.ai:Mp
- proven reliability;
- local Ms and Mp;
- already existing software to support AMOS development.

Its Mp should be minimal -- only large enough to handle initial startup data, recovery programs for catastrophic C.ai:Mp failures, and stand-alone C.ai hardware test programs. Likewise, its Ms should be minimal -- only enough to hold startup and recovery information. The alternative of using a larger Mp(amos) and Ms(amos) is tempting, but cost-effectiveness demands that C.amos use the main Mp and Ms as much as possible.

For reliability, two identical C.amos computers are shown in Figure 1. One of them will remain idle or perform non-essential housekeeping functions while serving as the spare. It could perhaps be used as a second connection to the ARPA network to increase bandwidth.

Exact specification of C.amos will require further refinement of C.ai. It appears, however, that a PDP-11 Model 45 size machine will be sufficient if enough other minis are used for Ms, Mt and network control.

CONSOLE

Scopes would be used to display the overall allocation of resources to tasks, and the status of the computer. Several scopes might also be employed for human intervention required in the management of the computer.

Each of the processors would occasionally require a certain amount of console activity which would be done by small computers as described in the section on the language processors. Due to the nature of the switching involved in the individual processors, it might be possible to use only one small computer to serve several large processors.

AMOS: A MINIMAL OPERATING SYSTEM FOR C.ai

The following privides an overview and first level design of an operating system for C.ai. The system is not specified in complete detail since (a) it is wasted effort to do so at this stage of the total machine design and (b) much of the information necessary to do so is not available (e.g., user characteristics).

In a system with multiple active units (processors in the case of C.ai) in which resources are to be shared there is a spectrum of possible systems ranging between the extremes of (a) having all control of resources vested in a single active element and (b) having no distinguished element and performing allocation by race or bargaining. AMOS is of a classical design in which ultimate control of all shared resources is in the hands of a single element (with the exception of the interprocessor bus) and all non-shared resources (e.g., processors) control themselves.

Likewise, there is a spectrum of possible operating systems ranging from a highly uniform one in which the user is essentially unaware of the existence of the multiple active elements that are, in fact, working on his task to a highly diverse one in which the user of one element is unaware of the existence of the others. AMOS tends toward the latter extreme.

DESIGN GOALS AND GUIDELINES

While not thoroughly defining the space of systems we are interested in, the following principles clearly must be observed.

1. Time and effort needed to construct AMOS must be small. Operating systems often lag and prevent the timely use of new machines. C.ai is meant to be operational in 2.5 years so there is little time for bringing up software.

2. The functions provided by AMOS must be a minimal set consistent with managing the hardware resources of C.ai. Complicated systems take a long time to build and are more open to problems.

3. The "users" of AMOS are the operating systems for each special-purpose P.ℓ. Thus the total operating system is a two-stage object: an overall operating system (AMOS) plus distinct operating systems on each processor. In most cases a human user and/or his program see only one of the individual systems, not AMOS.

4. Specification and construction of the operating system for a P.ℓ is up to the group responsible for that processor.

5. AMOS should usurp as few design perogatives as possible. That is, it should influence only minimally the design of operating systems and programs on individual P.ℓ's. Further, it should not greatly influence the design of C.ai as a whole in order that it will be possible in the future to replace AMOS with a completely different operating system.*

6. It should be possible to build very simple operating systems on the P.ℓ's if desired. They should not have to worry about physical level i/o and their communications with AMOS should be as simple as possible.

7. AMOS should be simple. This not only aids construction time and effort but also understanding (an essential point to insure correct operation and usage).

FUNCTIONS TO BE PROVIDED

It is easiest to specify what AMOS is to do by listing the major functions it is to provide. Elaborations of these functions will be

*C.ai is clearly a unique opportunity for implementing radically new virtual machines that exploit its parallel and functionally specialized parts. It is hoped that the computer will be available for research along this line. The understanding of such a machine, how to break up a load computationally, the characteristics of the programs run on it, etc. is so meager at present that initially the only sensible way to use it is as a collection of independent systems that happen to share some physical resources. AMOS and its hardware should not unduly impede research on more advanced modes of usage, however.

provided below in describing their implementation. It is assumed that a few other minor functions will be needed and can be added without greatly perturbing the design of C.ai or AMOS.

1. Allocate Mp. Individual processors must be given access to varying amounts of main memory. Addressing ranges and access protection must be set.

2. Allocate and control other on-site memory. Ms/secondary and Mt/tertiary must be allocated, but control must remain with AMOS in order to enforce security of parts allocated to different processors (and processes).

3. Handle communication between P.ℓ's and L(ARPA). All i/o other than to Ms and Mt is assumed to be over the ARPA network. In order to keep the operating systems on the P.ℓ's simple, network communication must be handled by AMOS.

4. Provide system status and accounting information. An on-site console must be maintained in addition to logging accounting information.

5. Startup of C.ai and individual P.ℓ's. Occasional cold starts of the entire system will be necessary. Individual P.ℓ's may come up and go down as well.

6. Movement of files between Mp, Ms, and Mt. P.ℓ's should not have to deal with physical level i/o. Further, large stores must be a shared resource.

7. Provide for sharing of Mp, Ms and Mt by different processors. AMOS must administer any such sharing.

WORLD VIEWS

An operating system provides a virtual machine. It is thus important to understand what the system and the users see.

What the User Sees

- a collection of functionally specialized systems with large memories usable over the ARPA network;

- the potential for processes on separate processors to communicate and share resources;
- a large on-site Ms for temporary use and a very large Mt for permanent storage of information.

<u>What an Operating System on a P.ℓ Sees</u>

- a device for allocating and overseeing sharing of Mp, Ms, and Mt;
- a device that will move files between Mp, Ms and Mt upon request;
- a device to handle the mechanics of transmitting and receiving information over the ARPA network.

<u>What AMOS sees</u>

- P.ℓ's competing for Mp, Ms and Mt;
- requests to share resources between processors;
- logical communication channels between P.ℓ's and the ARPA network to establish and maintain;
- files to be created and moved;
- M's to housekeep;
- accounting information to be logged and displayed.

STRUCTURES PROVIDING THE REQUIRED FUNCTIONS OF AMOS

Different structures can be chosen to provide the functions of AMOS. Those we have selected below seem to be sufficient for the task and consistent with our design objectives.

<u>Mp Allocation</u>

The opaqueness of how Mp allocations to P.ℓ's are being used and their size (64K wds) implies using an extremely simple algorithm. A P.ℓ will send a request to AMOS over the bus to allocate or deallocate a page of Mp; the request will include where in the P.ℓ's address space the page is to go. AMOS will check whether or not the P.ℓ is entitled to another page (a policy decision) if a new one is being requested. It will then adjust the bounds registers of the P.ℓ appropriately and signal it of the change over the bus.

A simple algorithm of assigning new pages to physical boxes of Mp that have the fewest number of active pages seems reasonable. If good monitoring hardware is built into C.ai, "active" might be defined in terms of number of accesses; otherwise, it could simply be defined as pages that are in use. Anything more complicated seems unjustified.

In order that each processor be able to handle large jobs at the same time, it will be necessary for the operating system on each P.ℓ to release Mp on a second-by-second basis whenever it is free. This can be handled on a gentlemen's agreement basis with perhaps some monitoring in AMOS to insure that no P.ℓ hogs too much core. Which algorithm to use is a policy decision.

Ms and Mt Allocation and Control

As far as a P.ℓ is concerned the basic unit of storage will be a file of arbitrary length. (For efficiency, information actually may be stored on Ms and Mt in standard block sizes). A P.ℓ can request AMOS (over the bus) to create a file; AMOS will check if the P.ℓ can have more space and if so, create a name for the file and pass it back. (To facilitate storing, the names should be from a single continuous space.) A P.ℓ can request for information to be transferred from Mp to an Ms or Mt file.

The request can be made with a priority, thus allowing swapping or paging information to be handled just like any other file only with higher priority for performing the transfer. Likewise, files can be transferred from Ms or Mt to Mp. Alternatively, information can be sent and received over the network directly to a file (see below). Files can be erased by request.

Note that the P.ℓ's specify where they want their files to reside. This seems essential since only they will know what they are being used for. Pricing structure, time limits, and allocation limits can be used to insure proper migration.

It is assumed that Ms and Mt provide hardware detection of record and file ends so that transfers of partial files may be made. On the other hand, record transfer may impose too much additional complexity on AMOS.

Communication Over the Network

AMOS will know nothing about users. It will have only logical channels that it can connect between a P.ℓ and some entity transmitting messages to C.ai over the network.

AMOS may receive messages on the network from entities for which it has no logical channel set up requesting access to a given P.ℓ. The P.ℓ may have told AMOS that it will take all comers, only certain ones, or that it wants to be informed of all requests for connection so that it can make a dynamic decision. If the requestor cannot be attached, he will be so informed.

If he can sign on, he is given a unique i.d. by AMOS and a logical channel is established to the desired P.ℓ. Until the connection is broken by the P.ℓ any incoming information headed by that i.d. will be sent to the proper P.ℓ (deposited in a section of his Mp or on one of his Ms files) with a signal going from AMOS to the P.ℓ whenever a transmission is completed. A dump mode will be available for the transfer of large blocks of data directly to an Ms or Mt file without interrupting the P.ℓ until the transfer is finished (even if it takes many network transmissions).

Similarly, a P.ℓ can request a file of information of any size to be sent out over a given logical channel.

System Status and Accounting

AMOS will record all system resource usage (e.g., Mp, Ms, Mt, T's, L(ARPA)) by each processor. The information can be displayed in summary form on a console and made available to the processors if appropriate. A processor can access the data of another under the usual sharing rules (see below). It is up to individual processors to record their own usage and to subdivide their use of system resources among their various users.

Each processor will supply a certain amount of status information to AMOS upon request in order to produce system-wide status displays. The content of this information depends on a more detailed specification of how individual P.ℓ's will be used.

Any error checking or internal monitoring information available to AMOS will be displayed appropriately. AMOS will also be responsible for utilizing such information to warn of faulty components or potential system bottlenecks.

Initialization

C.amos will have an autoload button that will load its local memory from a startup disk with a program to initialize Mp bounds registers and load its main Mp from Ms. Its bootstrap will also be able to retrieve from its local Ms various debug, checkout, and recovery routines.

C.amos will be able to startup any of the other P.ℓ's by a signal over the bus. Once started, however, AMOS has no control over the P.ℓ. This means that AMOS will have available the operating system (or a proper bootstrap) for each P.ℓ. In some cases this may include loading a microcode store.

File Movement

All file operations are logical (not physical) as described above. AMOS will have one or more minicomputers that will initiate transfers between memory hierarchies and perform housekeeping chores.

Resource Sharing

The mechanism for sharing is basically the same for all resources. Processor A (the owner of a resource) tells AMOS over the bus that processor B may have a given type of access to that resource. If later, B requests that access, it will be granted (unless A has rescinded the access rights).

In the case of Mp this is implemented by setting bounds registers. For files AMOS must keep lists of processors (not processes) that can access given files. It is then up to the processors to control the access of their individual processes.

As an example of Mp sharing, $P.\ell_1$ may tell AMOS that $P.\ell_2$ can have read access to one of its pages, say p_{1j}. $P.\ell_1$ will communicate directly with $P.\ell_2$ that it has permitted access to p_{1j}. Later, if $P.\ell_2$ requests access to p_{1j}, AMOS will set one of $P.\ell_2$'s bounds registers to permit read sharing. Permission for sharing (or relinquishing by $P.\ell_2$) can occur at any time. It is up to $P.\ell_1$ and $P.\ell_2$ to insure that permission is not withdrawn precipitously.

PERFORMANCE MONITORING

A computer such as C.ai must have adequate performance monitoring capabilities integrated into its basic design. Many initial decisions will require modification as the system matures and usage patterns evolve. Proper design of Ms and Mt systems, optimal allocation of Mp, and correct bandwidth to the network are examples of decisions requiring extensive measurement of usage.

Measurements should occur on a number of levels. Common facilities such as the L(interprocessor) and the Mp's could be monitored by passive hardware devices connected to a separate C.pm. This independence would insulate the PM from changes in AMOS and in the $P.\ell$'s. C.amos and C.pm could communicate directly for dynamic control, but other information would be stored for later analysis. Many items can be measured passively by: (a) busy-idle bits; (b) registers to read or sample; (c) counters. AMOS should be able to interrogate C.pm to obtain current data for scheduling and resource allocation and for system status requests from $P.\ell$'s. Thus the C.pm could have the following features:

(a) Pc-Mp-Ms type of structure;

(b) reduce its own data and keep current system information available for AMOS;

(c) write to its own slow Ms for later display and analysis.

Some examples of information of interest are: (a) K(Mp.port) errors could be counted, and transmitted to C.amos, (b) the number of memory references could be counted and waiting times tabulated, (c) a central clock may be provided which all P's may access. A central timing facility might also be included at the clock. In order to keep the traffic low, a facility such as the clock might broadcast the time so that each processor could maintain its own timers (which would undoubtedly be in software).

AMOS should have a number of software monitors built into its modules. Such hooks are best when implemented in parallel with the operating system. Selected information would be either written by AMOS or read from registers by C.pm. If AMOS is to be a resource allocator, some P.ℓ information may be required. Information of this type would place certain constraints on P.ℓ implementors, but the sharing of common resources requires some standardization.

Each P.ℓ should also include its own hardware and software measurement devices with which AMOS can communicate. A mixture of software and independent hardware monitors integrated into the design of C.ai will allow for future study of this new structure, and encourage growth based on a knowledge of actual performance and utilization.

REALIZATION CONSIDERATIONS

Our goal has been to develop and present a feasible technical solution to the problem of improving the computational power available to the ARPA ai community. While we have not performed extensive cost-effectiveness calculations nor worried about a precise timetable for building C.ai, we feel that a few comments on these subjects are in order. Clearly a much deeper and more detailed set of plans and calculations must be prepared if C.ai is to be built; hopefully, however, what follows will permit a sound initial judgment to be made as to whether to take the next step in the process of realizing the design we have presented.

SCHEDULING AND STRATEGY FOR THE PROJECT

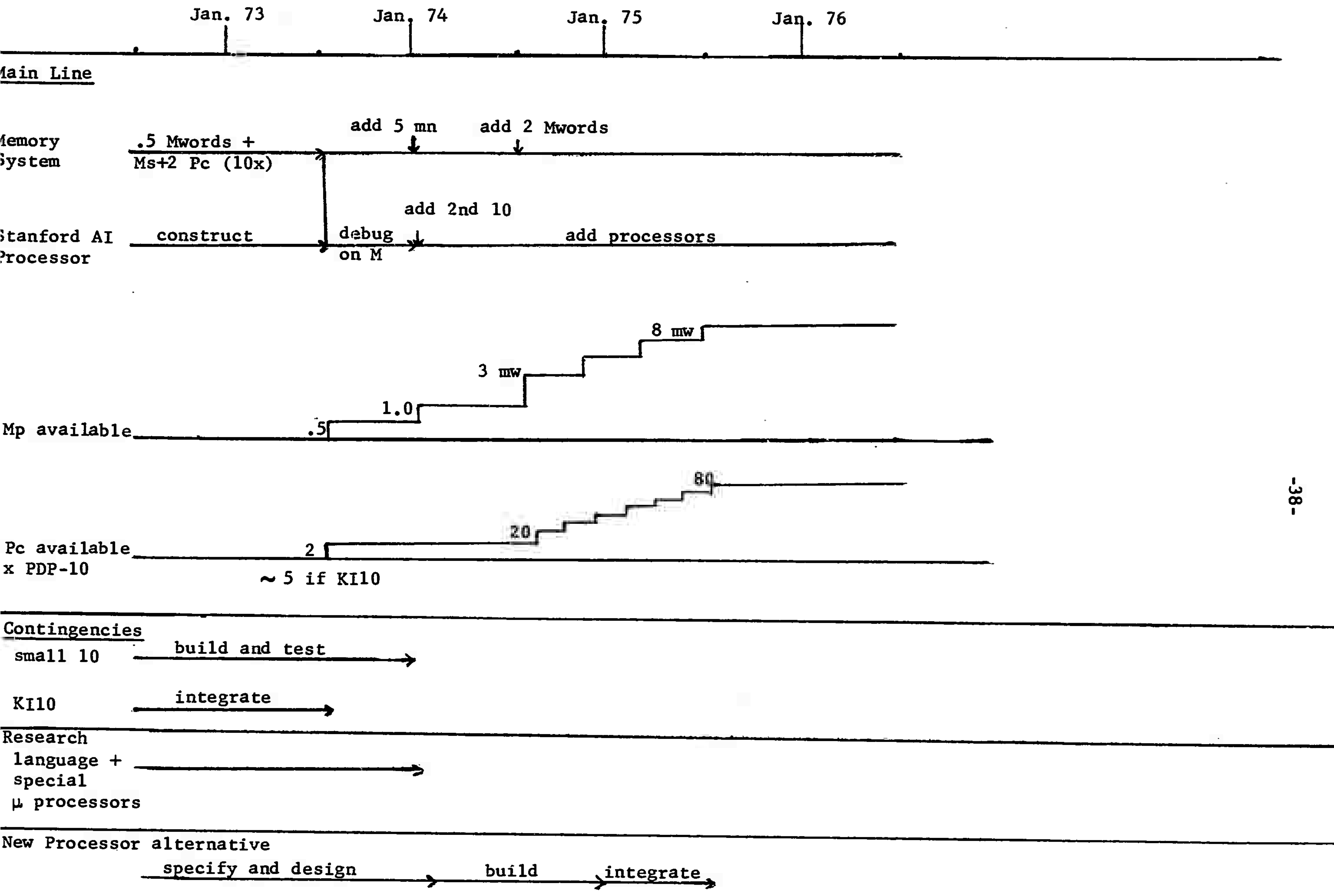
The basic strategy of the design is to provide an environment where a number of parallel efforts can be maintained, none of which is critical. In Schedule 1 we show the times at which various parts of the computer could be ready. From the schedule we see there is a deliberate attempt to have a number of independent ways of achieving success by various processor alternatives. All alternatives, however, depend on our being able to build the memory system, which is scheduled to be completed over the first year. Thus, we have tried to make the memory system the most conservative part of the design since all else depends on it working. Switches of this complexity have already been constructed; for example, in one of the early Lockheed memories attached to a PDP-10. It had 16 ports in each memory which is, in essence, the major part of the switch. In this case, the switch is twice as wide (74 points), and must accept up to four words transmitted serially. We also proposed that this switch structure be duplicated centrally so that 1/2 can fail or be worked on without bringing the system down.

Memory would be added slowly, as the processing power is increased. An initial configuration could be operating in only one year, composed of the nucleus of the memory (including the secondary memory) together with some small amount of processing power (two PDP-10 processors, running Tenex). This configuration would provide immediate benefit to some users by having a larger memory. Even more memory could be added if needed.

By proceeding slowly the operation of the computer could also evolve and the tie with the network could be made gradually. During the second year of operation the first and second Stanford AI processors could be integrated to operate with the memory system and the first two processors. The necessity for two processors is based on the fact that the facility should be able to supply a constant resource to its users and, thus, two processors would be necessary to meet the reliability requirement. At the end of the second year almost any configuration of processors and computers would be possible.

CONTINGENCIES AND RESEARCH

The center section of Schedule 1 deals with the scheduling of specialized processors for languages. Their progress would not influence the main schedule line to any great extent, but could provide backup for the extra computing power. If specialized processors prove as valuable as



Schedule 1. Possible timing of project for a single large site.

we believe they will, then such an approach would pay off and possibly negate the need for the large processors. Two of the contingencies (KI10 and a small microprogram-based PDP-10) could no doubt be contracted to DEC because they would be part of their standard product line.

NEW PROCESSOR ALTERNATIVES

The final line on Schedule 1 shows what we might expect (optimistically) from a new processor based design. In three years we might have such a system operating ready for software debugging (assuming that the design and fabrication had gone on in parallel with the hardware development). In possibly as early as four years, users might be on the system. (For example, by comparison, the PDP-11 is now about $2\frac{1}{2}$ to $4\frac{1}{2}$ years old (depending on when you count the start).

The problem with such a design appears to be that it isn't clear why any company would want to build it. The only company which has both the resources and the diverse product line strategy is Honeywell. A GE645-based processor has a large address space and possibly might be considered.

There is some evidence that a straightforward machine based on the PDP-11 structure, but with large addresses would be useful and could provide better encoding efficiency than a PDP-10. It might be much easier to build than the Stanford AI Processor, but if it were not part of a main product line it would be difficult to get it built. Buying computers which are part of another product line allows both a much lower cost and higher reliability (by not having a one of a kind).

Nevertheless ARPA could still undertake the study of a processor and possibly even build it at a university. Such a processor would then be ready to assume the large amount of computing in perhaps three to four years. The main thrust of the design and fabrication of a processor of this type would be research, since if it worked we would have significant fallout, while no harm would be done if it failed because we have no commitment to the design.

NECESSITY TO MODIFY THE STANFORD AI PROCESSOR

In order to obtain an access to a large memory within the next two to three years, about the only alternative is to use an existing processor (instruction-set). Although we have not explored the many candidate processors, the PDP-10 would probably rate very high, not because of esthetic reasons, but because of the current software and knowledge of PDP-10's within our community. The possibility of using a 360 and/or GE645 might also be explored since both of these processors exist within the community, and there is a large amount of software written for them.

Assuming we do intend to use the Stanford AI Processor design, then there is a necessary and several desirable modifications to it. The necessary modification is the extension of the address space to 2^{24} words. This might be done in several ways: some form of segmentation, which when operating, would allow full 36 bit words to be used as addresses; some means of communicating nicely across processes; here the main problem is that list structures would point across these process boundaries; the introduction of instructions for data movement: load, store, load double, store double, push, pop, execute using a double length address, and a list data structure with large pointers. While all of these methods may be regarded as somewhat ad hoc, they should be sufficient. Thus, if we compare the modified version with some of the alternative off-the-shelf processors, making these modifications will probably make the PDP-10 no worse than the alternatives. Some of the desirable modifications will deal with list structure manipulation and typed variables. All in all, it appears from the early explorations about modifications between R. Greenblatt (M.I.T.) and the Stanford project, that they can be made without unduly distorting the Stanford design.

ECONOMIES OF SCALE

We have not carried out calculations which would allow one to decide whether it would be worthwhile making a system smaller than the one proposed. The considerations for not scaling down the computer to 1/4 the size would be:

1. Fixed operating costs of the laboratory facility (overhead), engineers, programmers, cooling, lab supplies, space.
2. The design task is fixed almost independent of machine size. Thus, it would be desirable to get as much power as possible in a single place. This would give us the highest performance/total cost.
3. Desirability of having a central facility with a very large amount of processing power attached to a single, large memory.
4. Desirability of having a single, very large memory. This would permit more users through better user-load-averaging. Also, larger programs could be run.
5. There may be some quantity discount for drums, disks and memory.
6. Higher reliability. All facilities would be duplicated in a larger computer.

7. One connection to the ARPA network.

8. As the machine is scaled down, certain fixed hardware (e.g., disk controllers) would have to be duplicated at each site.

These cost advantages tend to be overshadowed sometimes by the pride of ownership. Everyone would like to have his own! Indeed, there are usually strong reasons to have exclusive control of one's research tools. Also, the high speed devices of some projects could not be used very easily via the network. (There does seem to be some agreement, however, that real-time jobs could be broken up to run satisfactorily on a single large machine, coupled to front end machines via the network.) Lastly, there is the issue that a single machine would be more vulnerable (e.g., to power failure, fire) than several smaller machines.

Although the cost/unit of computing has to be smaller with a central facility, preliminary analysis indicates this cost probably won't be significantly cheaper than with four facilities. Again, the decision about the number of sites is comparitively unimportant and can be postponed until a first facility is operational.

CONCLUSIONS

We have given an overall argument as to the feasibility and desirability of building a computer to be used in ai research. It is basically a very conservative approach built on current computers together with what we think can be done easily with the technology. We have not assumed very high performance processors. For example, one processor we think feasible is the Stanford AI version of the PDP-10. Such a processor should not require more than two years to develop, and should be replicatable for in the neighborhood of $100,000.

Given our overall numbers, we think it is worth trying to specify the next level of detail based on our evolutionary approach. We believe that an approach that departs from a conventional structure (e.g., by placing specific interpretation on addressing) will both decrease the performance and also make the memory too special purpose, thereby eliminating unspecified future use that might be made of that large facility.

We think the real point of the structure is that it should be simple, yet provide as much potential power (bandwidth) as possible, and should not be very presumptious about how particular future processors would use the facility. In particular, we strongly believe that additional power will be gained by developing special purpose processors to be used on C.ai and that this course should be thoroughly explored.

APPENDIX 1

DESIGN CONSTRAINTS AND SOME CONSEQUENCES

0. Engage in design construction only if worthwhile performance and performance/cost can be shown to materialize.

1. Technology is hard to forecast accurately beyond 3 - 5 years.

   Consequence: Should have machine operating in $2\frac{1}{2}$ years, otherwise we'll have built an obsolete machine. Plan for some improvement in technology (factor of 4 in speed, size are possible numbers).

2. Usage is hard to forecast from a design (especially a radical design).

   Consequence: Leave many variables to be bound later (hopefully by user or by software) and expand the design when use is more apparent.

3. Operating systems take 2 ~ 5 years to design and get operational once the hardware is specified.

   Consequence: Prepare interim methods of operation; prepare for shock; question the project; keep aspirations low.

4. Certain memory technology costs in next two or three years: (see Table 2)

   Consequence: Probably require all these memories.

5. This must be an ARPA network resource.

   Consequences: Don't need terminal handling or unit record io.

   Don't need long-term archival and backup storage.

   Need a filing system locally for temporary programs, data, etc. because of bandwidth. Otherwise the response time will include that of network. Also network could be swamped just transshipping large files.

Allow all contractors to engage in certain kinds of improvements.

High reliability -- probably affects more users (say 50 ~ 100 versus 10 ~ 25 at a site).

Multiple users simultaneously.

6. This is a machine for artificial intelligence research.

   Consequences: Variety of specialized devices connected to it (e.g., robotic equipment); powerful arithmetic processing; large memory.

7. This is a symbol and list processing machine.

   Consequences: Access to very wide words so that different types of encodings are possible for various languages. The (memory size)/(memory accessed) ratio is probably large.

8. Examine all techniques (especially those which have been found to work in hardware and software):

   Examples: cache (1 or more);

   microprogramming;

   hash coding;

   highly specialized processors.

9. Design should be highly likely to materialize.

   Consequences: must be understandable;

   maximum amount of project parallelism;

   minimum amount of interaction among parts (hence well defined interfaces).

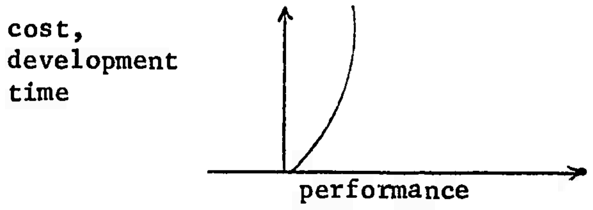
10. Base system performance on easy-to-identify techniques.

    Consequences: parallelism in
    word length,
    processors,
    memories;

specialization of function, although care must be taken to avoid little used functions;

circuit technology speed.

11. Development time, cost, and performance are related.



Therefore it is important to select a machine before diminishing returns occur.

Consequence: Small processors (also specialized processors).

12. Main cost of system will be memories (primary, secondary and supporting files).

13. Must have high reliability (availability) otherwise real performance will be low when averaged over long periods.

Consequences: at least 2 of everything (even with simple components) to give an overall high uptime;

single error correction may be worthwhile in memories (subject to analysis);

never turn off power (possibly too stringent), but a style of design to be violated if too expensive and design time-consuming;

connect other equipment during operation;

methods of on-line and off-line testing of all components.

14. Try to minimize the number of specialized components in central part of design.

Consequences: research in modular systems.

APPENDIX 2

OUTLINE OF P.L* PROCESSOR DESIGN

The following is an abstract of the report "C.ai (P.L.*) -- An L* Processor for C.ai" by D. McCracken and G. Robertson. It is available from the Carnegie-Mellon Computer Science Department or the Defense Documentation Center.

The results of a preliminary design study for a specialized language processor (P.ℓ) for L* are presented. The objective of the study is to give an example of a specialized processor for C.ai.

The L* processor is to run 20-30 simultaneous L* users with very large address spaces at a speed improvement of better than 10 times a typical PDP-10 L* system. Its cost should be low relative to the memory resources of C.ai.

The design presented is that of an L* central processor (Pc.L*) with a very low-level instruction set (about the level of typical microcode). Pc.L* is time-shared by a mini-computer that sits to the side, so that each L* user sees himself as running on a base L* processor. User contexts are switched by swapping processor status information in Pc.L*.

Each L* user has complete access to the central processor status through his address space. His machine code (microcode) can appear anywhere in the large address space, but executes out of a fast cache memory. It thus runs at microcode speeds. L* programs and data being interpreted by the machine code are accessed explicitly from a second cache memory. The initial L* kernel system will consist of ~ 1K of machine code, with some initial data and available space.

The low-level instruction set of Pc.L* does not contain any of the more complex instructions (such as floating point arithmetic and byte manipulation) that usually exist on large general purpose computers. These capabilities are meant to be written in machine code as needed by each L* user. He thus gains considerable flexibility in the exact nature of these higher level operations at the cost of increased programming effort and somewhat reduced efficiency compared to hard-wired implementations.

The results of this preliminary design effort, although still unclear in spots, shows that a specialized processor could run very large L* systems on C.ai at 20-40 times the speed of a PDP-10.

APPENDIX 3

OUTLINE OF P.LISP PROCESSOR DESIGN

The following is an abstract of the report "C.ai (P.LISP) -- A LISP processor for C.ai" by M. Barbacci, H. Goldberg, and M. Knudsen. It is available from the Carnegie-Mellon Computer Science Department or the Defense Documentation Center.

A special processor designed for efficient LISP processing is described. The processor is micro-programmable and makes heavy use of a fast scratchpad memory with several special purpose registers (small function units) and general byte manipulation capabilities. The approach taken has been to avoid unorthodox schemes of implementation and employs little in the way of unusually new (and untried) hardware.

Such a conservative approach will enable a fairly good implementation in a reasonable time. One of the places where efficiency in list processing (and in most programming applications) can be enhanced is in the ratio of instruction fetches/data fetches. To that end two things that are not usually available were required: writeable (up-datable) microcode and recursive control of microcode. With them, it is possible to implement the language interpreter as close as possible to the real hardware machine. Such a machine could also be a "shell" language processor. However, this was not a goal of the design, but rather a by-product.

The microprogrammed processes include a storage-compacting garbage-collector, which can be made to operate incrementally in parallel with user-program execution. This option avoids interruptions in LISP execution for garbage collection.