C.ai—A computer architecture for AI research*

by C. GORDON BELL**
Carnegie-Mellon University
Pittsburgh, Pennsylvania

and

PETER FREEMAN
University of California
Irvine, California

INTRODUCTION AND MOTIVATION

A recent article analyzes the need for and possibility of an ultimate computer.[1] While the ultimate machine is still distant, much current research on system structures has the goal of significantly increasing computing power along one or more dimensions (e.g., processing speed, memory size, functional capability, reliability).[2,3,4,5,6]

The most obvious way of increasing power is through parallelism. The earliest proposal to achieve parallelism is the coupling of multiple processors to a shared primary memory—multiprocessing. Yet, the parameters of a design, especially the connections between processors, memories, and the outside world, can take on many different values.

The architecture presented here is intended to increase the computing power available for a particular application—artificial intelligence research. It was formulated under the constraint that if built, it would have to be operational within two years. Its lifetime was assumed to be on the order of five years, with a slow rejuvenation replacement process occurring during use. Although it was formulated to be used via a network such as that of the Advanced Research Projects Agency (ARPA), the result has implications for the design of any currently feasible, very large computer.

We present two major parts of the design—the structure of the hardware (and some of its details*) and the operating system—as originally stated** and then briefly discuss its implications for research on multiprocessing.

Requirements for ai computing

A number of special function computers ranging from business to scientific have been described.[8] The characteristics of machines used for ai research appear to span this spectrum, exhibiting the maximum of each characteristic attribute. The more important of these attributes will be examined from the standpoint of ai computing.

Memory size

The primary (program) memory is larger in an ai environment than most other computers because of the local program and data base. Usually data is highly interrelated and linked for such programs as natural language processing. We have assumed an average program size of 250,000 74-bit words and a working-set size of 100,000 74-bit words. The ratio of secondary memory (e.g., drum) to primary memory (e.g., core or integrated circuit) we have proposed is quite low, based

* 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.

** Present address: Digital Equipment Corporation, Maynard, Massachusetts.

* The PMS notation[7,8] 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."

** The design was first presented in a technical report[9] in May, 1971.

more on swapping operation than demand paging. This appears justified since the list structure of many of the programs may not permit small working-sets, although a better model of a typical program is clearly needed. A coupling to a larger tertiary memory is provided for files.

Processor power

The central processor power requirement for ai computing seems less than for the largest scientific processor because the typically large memory is accessed more or less randomly in a relatively inefficient fashion. Thus, the more powerful facilities of a scientific processor may go unused because the data-types (e.g., floating-point) are unused. In some cases ai processes are compute-bound, and need significant processor power in critical areas.

PMS structure

The ai computer usually requires better communications facilities to external equipment and humans 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 (conventional and specialized)

The design permits the use of existing conventional processors as well as specialized processors. Conventional processors can be either 32 or 36 bits. In particular, since the PDP-10 is widely used for ai research, PDP-10 processors can be used (e.g., DEC's KA10 and KI10, and the Stanford Artificial Intelligence Laboratory version).

There is also a need for a facility which can be used for experimentation with new instruction-sets for ai processing. Specialized language processors can be fabricated, tested, and used within the environment. Typically, the instruction-set is the characteristic that usually comes to mind when ai computing is discussed. Operations for a stack, a garbage collector, hash-coding on linked lists, etc., are obvious candidates. On the other hand, a simple processor with floating point arithmetic is often adequate because decisions can be bound in software and later changed.

FUNCTIONAL OVERVIEW OF C.ai

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 this design 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. This design should be viewed as a specification of such an environment.

The aim is to provide a collection of virtual machines, organized along multiprocessor lines. It is thus important to understand what the system provides and what the users see.

The user sees

- a collection of functionally specialized interconnected computers, each with large memories;
- the potential for processes on separate processors to communicate and share resources;
- a large secondary memory for temporary use and a very large tertiary memory (i.e., files) for permanent storage of information.

An operating system on a processor sees

- a mechanism for allocating and overseeing sharing of the three level memories (primary, secondary and tertiary);
- a mechanism for the transfer of files between memory levels;
- a mechanism to handle the transfer of information to the outside world.

The overall operating system sees

- processors competing for memory;
- requests to share resources between processors;
- logical communication channels between processors and the outside world;
- files to be created and moved;
- memories to housekeep;
- accounting information to be logged and displayed.

THE HARDWARE STRUCTURE OF C.ai

Figure 1 shows a simplified diagram of C.ai. It is a multiprocessor system with a centralized cross-point

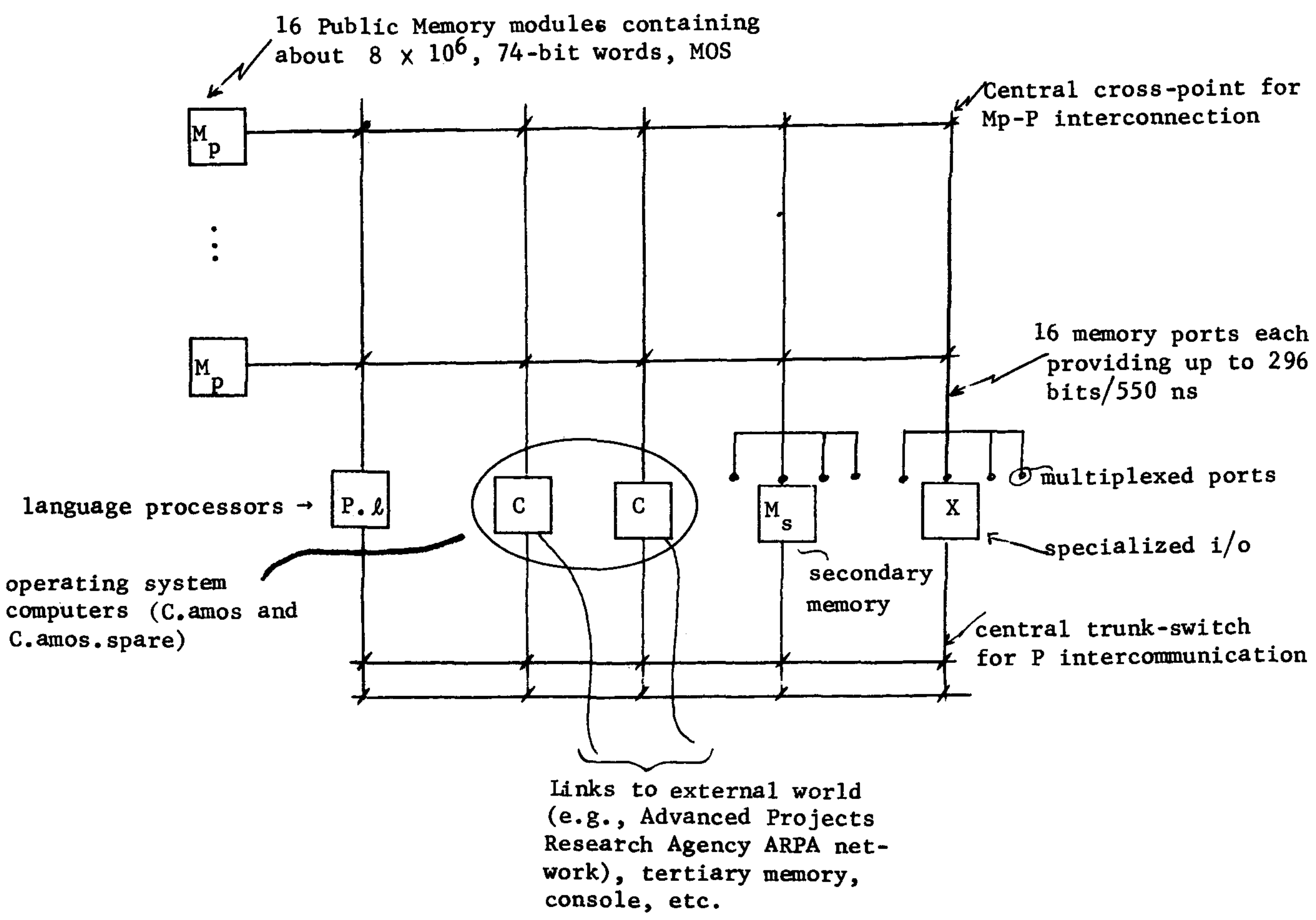

16 Public Memory modules containing about 8 x 10^6, 74-bit words, MOS
Mp
Central cross-point for Mp-P interconnection
16 memory ports each providing up to 296 bits/550 ns
multiplexed ports
language processors
P.ℓ
C
C
Ms
X
specialized i/o
operating system computers (C.amos and C.amos.spare)
secondary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

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

switch which allows up to 16 processor or secondary memory (e.g., drum) ports to be connected to the primary memory. Some ports can be multiplexed between several components. The characteristics of the primary memory (the dominant component of the system) are: MOS, 550 ns cycle time; 2^{21} 296-bit words accessible as 74, 148, 222 or 296 bits in a single access.

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 $p+m$ and shorter distances. Also, by using a non-bus arrangement, processor and memory modules can be removed while the system is operating.

Another type of switch is used to provide intercommunication among the processors. This is shown in the lower part of the figure and is described later in more detail.

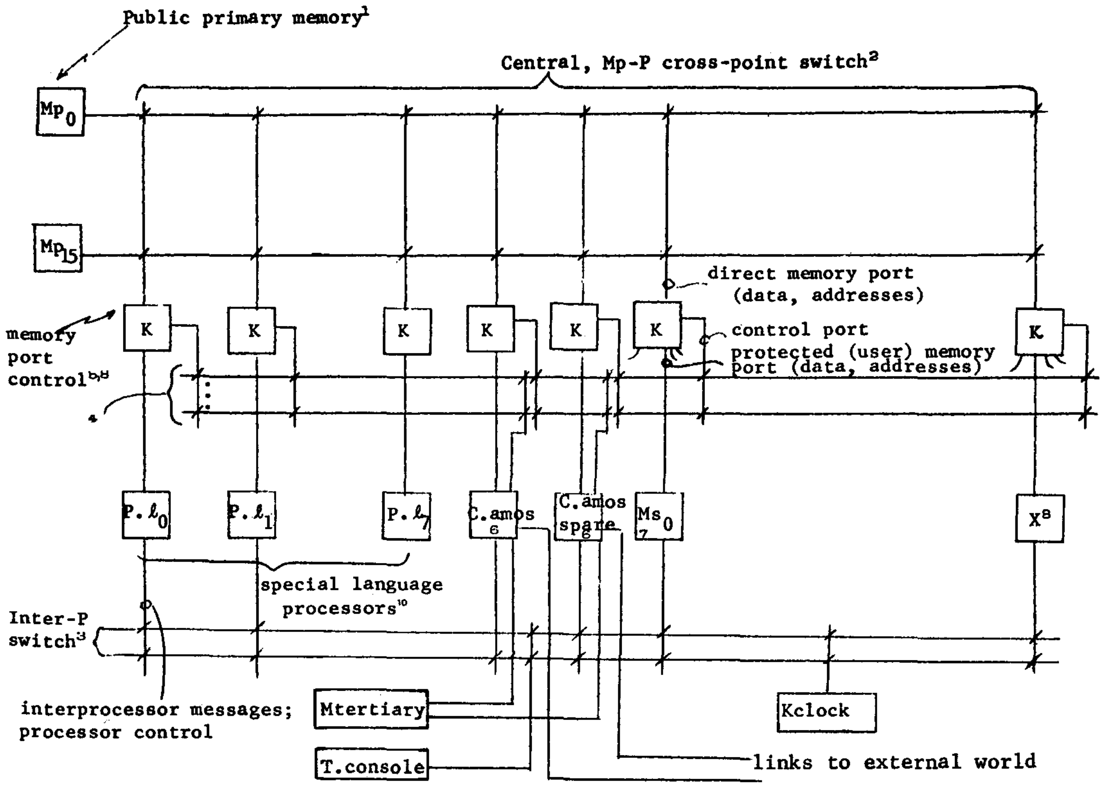
A central computer (C.amos) executes the operating system. It has some private memory, interconnection to i/o devices, terminals, and tertiary memory. Two C.-amos computers are shown, but only one will be in use at any time (see below). The other processors are general-purpose or specialized language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

Figure 2 provides a more detailed description of the structure shown in Figure 1. It will be used to aid the description.

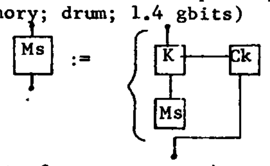
Primary memory

The characteristics of various memories are given in Table I. We have used specific quotations for cost, performance, etc.

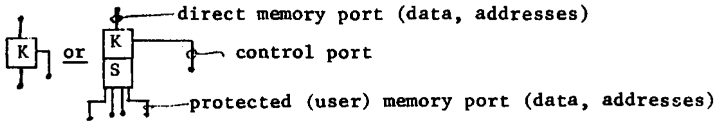
The primary memory, Mp, gives an overall memory size of about 620×10^6 bits and (because of the 16 ports) a bandwidth of 8,600 million bits/sec (16 ports$\times$296 bits/port; .550 μsec cycle). The access time, as measured at the processor, will be about 350 ns. The actual


Public primary memory[1]
Central, Mp-P cross-point switch[2]
Mp0
Mp15
direct memory port (data, addresses)
control port
protected (user) memory port (data, addresses)
memory port control[5,8]
4
K
P.ℓ0
P.ℓ1
P.ℓ7
C.amos 6
C.amos spare
Ms0 7
X8
special language processors[10]
Inter-P switch[3]
interprocessor messages; processor control
Mtertiary
T.console
Kclock
links to external world

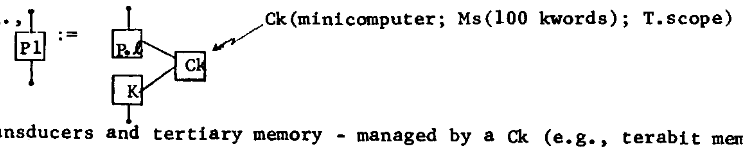
[1]Public, primary memory; $2^{23} \sim 2^{24}$ words; 74, 148, 222, or 296 b/w; 550 ns/w; MOS
[2]Central; cross-point switch; Mp-Pc|Ms dialogues; 16x16; width: 74 bits
[3]Central, trunk switch; inter P dialogues; ≥ 2 trunks } identical protocols
[4]Central, trunk switch; P-K(Mp.port) dialogues; ≥ 2 trunks } identical protocols
[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

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. Alternatively, the 74 bit word will allow present 36-bit processors, such as the PDP-10, 1108, and Honeywell 645 to utilize the memory. The memory will consist of 16 modules and will connect to the processors through a 74-bit wide cross-point switch. There are 16 processor ports each 74 bits

TABLE I—C.ai Memory Characteristics

Device	function***	t.access (t.cycle)	i.rate mb/s	i-width (bits)	total i.rate (mb/s)	power (watts)	size module 10^6b	cost (k$)	cost/bit ¢/b	controller costs (k$)
photomemory	t						$\geq 1\times 10^6$			
moving-head disk	t	0~55 + 20 ms	3.3**	1×3	3.3×3	15 kw/ unit	200×20=4000	300	0.0075**	300
drum	s	8 ms	3.3	8~16/drum	50/drum	1 kw/drum	70×20=1400	900	0.048	5×40
shift register	s	131 μs 780 μs	1	296	296	200 kw	1400	1800	1.25 0.7	5×40
core	p	.7 μs (1.8 μs)	.55	296×16	163 2605	200 kw	620	10,240	1.65	500
MOS	p	500 nsec (1.2 μsec)	.83	296×16	246 3947	200 kw	620	5760	.93	500
MOS	p	250 ns (400 ns)	2.5	296×16	740 8600	200 kw (.2 mw/ bit)	620	9500*	1.52	500
Bipolar	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	

*Estimate

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

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

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.

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 (mapping) 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. This is a concatenation, instead of an addition, and thus should be quite fast ($\leq$50 ns). The relocation (mapping) 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), protection and sharing among processors, and data parity checking.

The relocation registers will be transparent to each processor and will serve the function of manual, address switches on the memory. Thus, no manual switching need be provided on the individual memory modules; the same effect is achieved by informing the memory control processor to vacate the desired module and consider it unusable. All relocation and protection, as commonly found in timesharing system processors, will be included as part of a 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 connected 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 and detection hardware. Here, since we assume that some faulty processors will be attached to the ports, the port control

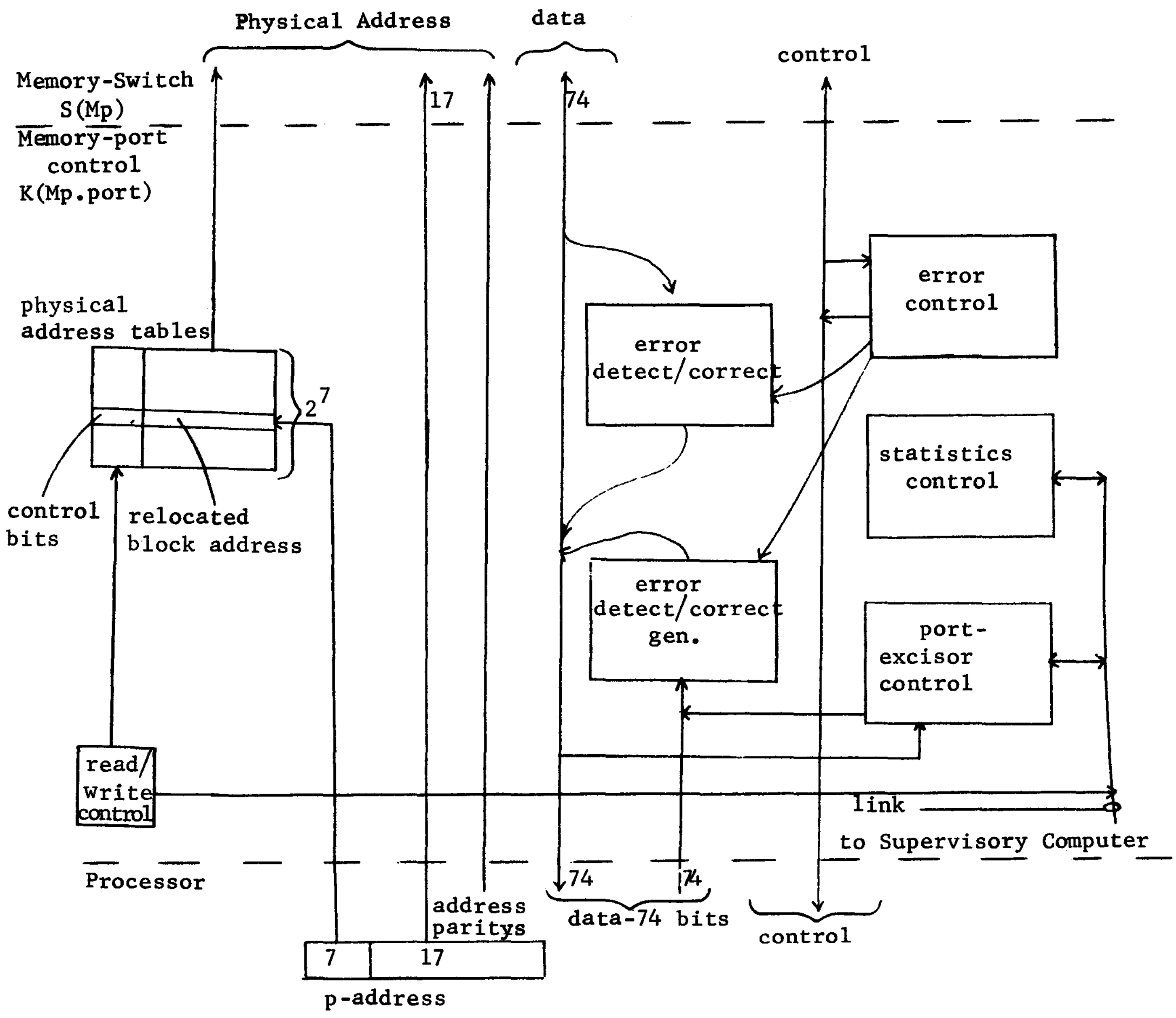


Figure 3—K(Mp.port) memory-port mapping (relocation), error-detection, error-correction and control

must supply correct data to the memory in order that other devices (e.g., drums, disk memory) can detect faults.

Primary memory switch

Since the switch is critically central, a dual cross-point may be preferable. This is essentially a compound switch consisting of two cross-point switches and a (m+p)×S(1-input, 1-output) switch as shown:

Mp *m-inputs*

$$\begin{tikzcd}[column sep=small] & \text{S(cross-point; m}\times\text{p)} \arrow[dr, dash] & \\ \text{S(m; 2)} \arrow[ur, dash] \arrow[dr, dash] & & \text{S(p; 1) } p\text{-inputs P} \\ & \text{S(cross-point; m}\times\text{p)} \arrow[ur, dash] & \end{tikzcd}$$

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 8×8 switches, even more independence can be gained.

The memory switch can utilize current MSI (medium scale integration) logic. The switch will utilize 16 bit multiplexors with a typical data propagation time of 10 nsec (assuming the data select lines have been settled for ≥ 30 nsec). Faster circuits, such as Schotky TTL and LSI switching modules, will probably be available in time for any actual construction.

Secondary memory

The secondary memory investigated includes: (1) mechanical devices (drums, fixed head discs, etc.), (2, shift register or other block-oriented solid-state memory, (3) and random access memory (RAM). Current characteristics for these devices are shown in Table I. Mechanical devices will probably hold a price advantage of an order of magnitude for several years. It does not appear that shift register memory will become sufficiently cost effective over random access memory for our purposes. On the other hand, block-oriented random access memories may be available shortly.

A secondary memory system might consist of 20 drums, for example, with the characteristics given in Table I. 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.

Initially, the secondary memory 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 could take advantage of the 4-word sequential feature of the primary memory. Since one wishes to maximize the bandwidth between secondary and primary memory, as the system grows to use many drums on several ports it will probably be necessary to insert a computer to control the secondary memory, C(Ms). The secondary-tertiary memory system might then look as shown in Figure 4.

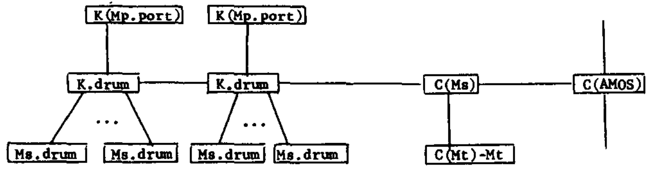


Figure 4—Eventual secondary-tertiary memory structure

Tertiary memory

Clearly, a computer of this capacity requires some on-site mass storage. This will permit programs to reside on tertiary memory until they are brought into either primary or secondary memory for more rapid access. The tertiary device will be controlled by its own processor. Aside from its size (on the order of 10^{12} bits), it has not been specified any further.

Console

Scopes will be used to display the overall allocation of resources to tasks, and the status of each processor and the overall system. Several scopes may also be employed for human intervention required in the management of the system.

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 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. A processor will be able to communicate with itself on the bus.

If the proposed interprocessor traffic appears to warrant it, more than two trunks can be added. However, processors may communicate at high data rates through shared primary memory.

A SOFTWARE STRUCTURE FOR C.ai

This section provides both an overview and first level design of AMOS, a minimal operating system for C.ai. The system is not specified completely, however.

In a system with multiple active units (processors, in the case of C.ai) and shareable resources there is a spectrum of possible systems ranging between the extremes of having all control of resources vested in a single active element to having no distinguished component with respect to resource allocation. AMOS is a classical design in which ultimate control of all shared resources is by a single component although all non-shared resources (e.g., processors*) control themselves.

* Processors may, of course, be shared among processes on a local basis. Our concern here is with the global management of the system.

Likewise, there is a spectrum ranging from the highly uniform in which the user is unaware of the existence of multiple components working on his task to the highly diverse in which the user of one component is unaware of the existence of the others. AMOS tends toward the latter extreme. The hardware architecture does not prevent the former, however.

Design goals and guidelines

While not thoroughly defining the space of systems we are interested in, the following provide a partial specification:

1. Time and effort needed to construct AMOS must be small.
2. The functions provided by AMOS must be minimal, consistent with managing the hardware resources of C.ai.
3. The "users" of AMOS are the operating systems for each processor. Thus the total operating system is a two-layer object: an overall operating system (AMOS) plus distinct operating systems on each processor. In most cases a human user and/or his program sees only one of the individual systems, not AMOS.
4. Specification and construction of the operating system for a processor is the responsibility of its designer.
5. AMOS should usurp as few design prerogatives as possible. That is, it should influence only minimally the design of operating systems and programs on individual processors. Further, it should not greatly influence the design of C.ai as a whole so that in the future it will be possible to replace AMOS with a completely different operating system.*
6. It should be possible to build very simple operating systems on the processors if desired. They should not have to handle transfers to i/o devices and their communications with AMOS should be simple.

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 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 primary memory. 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. Secondary and tertiary memories 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 processors and the external world. In order to keep the operating systems on the processors simple, 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 processors. Occasional cold starts of the entire system will be necessary. Individual processors may come up and go down as well.
6. Movement of files between memory levels. Processors should not have to deal with physical i/o. Further, large stores must be a shared resource.

Structure providing the required functions of AMOS

Different structures can be chosen to provide the functions of AMOS. Those selected below seem to be sufficient for the task and consistent with the design objectives. A more detailed overall design and/or simulation may, of course, indicate the choice of alternative structures.

Primary memory allocation

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

In order that several processors be able to handle

* C.ai is clearly a unique opportunity for implementingr adically new virtual machines that exploit its parallel and functionally specialized parts. 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 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.

large jobs at the same time, it will be necessary for the operating system on each processor to release primary memory on a second-by-second basis whenever it is free. This might be handled on a gentlemen's agreement basis with perhaps some monitoring in AMOS to insure that no processors use too much core. The specific algorithm to use is a policy decision.

Secondary and tertiary memory

As far as a processor is concerned the basic unit of storage will be an arbitrary length file. (For efficiency, information actually may be stored on secondary and tertiary memory in standard block sizes.) A processor can request AMOS (via the communications bus) to create a file; AMOS will check if the processor 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 processor can request information to be transferred among memories.

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 secondary or tertiary memory to primary memory. Alternatively, external information can be transmitted directly to or from a file (see below). Files can be erased by request.

Note that the processors specify where they want their files to reside. This seems essential since only they will know the use. Pricing structure, time limits, and allocation limits can be used to insure proper migration.

It is assumed that lower level memories 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 to the outside world

AMOS will know nothing about specific users. It will have only logical channels that it can connect between a processor and some external entity transmitting messages to C.ai. Since C.ai is intended to be a resource for use among a large number of users via a network (in this case the ARPA network) this will provide the mechanism for establishing contact between users and their processes.

AMOS may receive messages from entities for which it has no logical channel set up, requesting access to a given processor. The processor may have told AMOS that it will take all callers, 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 a user can sign on, he is given a unique identification by AMOS and a logical channel is established to the desired processor. Until the connection is broken by the processor any incoming information headed by that identification will be sent to the proper processor (deposited in a section of his Mp or on one of his Ms files) with a signal going from AMOS to the processor whenever a transmission is completed.

A mode will be available for the transfer of large blocks of data directly to a secondary or tertiary memory file without interrupting the processor until the transfer is finished (even if it takes many transmissions). Similarly, a processor 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., memories, i/o gear, external links) by each processor. The information will 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 more detailed specifications of how individual processors 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 start-up 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 start up any of the other processors by a signal over the bus. Once started, however, AMOS has no control over the processor. This means that AMOS will have available the operating system (or a bootstrap) for each processor. 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 the 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 primary memory sharing, A may tell AMOS that B can have read access to one of its pages, say P. A will communicate directly to B that it has permitted access to P. Later, if B requests access to P, AMOS will set one of B's bounds registers to permit read sharing. Permission for sharing (or relinquishing by B) can occur at any time. A and B must 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 memory systems, optimal allocation of primary memory, and correct bandwidth to the outside world are examples of decisions requiring extensive measurement of usage.

Measurements should occur on a number of levels. Common facilities such as the interprocessor bus and the primary memory could be monitored by passive hardware devices connected to a separate computer, C.pm. This independence would insulate the C.pm from changes in AMOS and in the processors. 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 must be able to interrogate C.pm to obtain current data for scheduling and resource allocation and for system status requests from processors. Thus, the C.pm should have the following features:

(a) Pc-Mp-Ms (processor-primary memory-secondary memory) type of structure;
(b) ability to reduce its own data and keep current system information available for AMOS;
(c) ability to 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 processors 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 processor information may be required. Information of this type would place certain constraints on processor implementors, but the sharing of common resources requires some standardization.

Each processor 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.

PERFORMANCE CHARACTERISTICS AND EVALUATION

Table II 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. The measures used by Roberts[10] were included to compare the performance with these machines. In some cases the chart is misleading since C.ai has 20 times the memory of the next largest machine (STAR). However, for ai research, memory size is probably the single most im-

TABLE II—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$);Bit×mops/ 10 sec; Bit×mops/sec/$
PDP-10	36+1	0.26	9.7	4	144	9	0.4	1; 14.4; 14.4
Stanford AI-10	144+4 (36b/instr)						4.0	1; 144; 144
Model 91	64+8	0.52	37	21	1370	10×(1~2)	6.0	7.7; 384; 49.9
CDC 6600	60	.26	15.4	32	1920	12×10 (i/o) 600 (ECS)	3.0	5.5; 180; 32.7
C.ai	74~296	8.3	620	29~120	2150~ 8600	50×5	(4~12)×10	13*;1440~4320; 110~330
							(4~12)×20	16;2880~8640 180~540
C.ai/4	74~296	2.3	155	8~30	504~ 2150	50×2	(4~12)×3	
							(4~12)×6	
CDC STAR	512 (32b/instr)	.131	74	25	12,800	(50~100)× (1~5)	100	10; 3200; 320
ILLIAC IV[5]	64+	.131	8.2	64×4.1= 261	16,800	1000	256	10; 16384; 1638.4

*Assumes $8m for memory, $2m for peripherals and $300K per processor (10 Pc's in first case, 20 in second); Stanford AI-10 assumed. Adjusting the memory size to that of STAR, yields $7m (total); 1440~4320; 205~620 and $10m; 2880~8640; 288~864.

portant characteristic. This has been adjusted in the footnote to the table. The computation is based on 36 bit operations. Using a larger word would increase the performance indicator, though probably not any real performance for this task.

The PMS diagram of Figure 2 has sufficient detail for deriving basic performance characteristics of the computer. Each processor is assumed to cost approximately $100,000. A 16×16 switch should run approximately $200,000.

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). By comparison, a PDP-10 demands 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, a cache next to a processor will be necessary to make full use of the capability.

By using a cache memory the needed bandwidth into primary memory is significantly reduced. One would expect about 95 percent 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 percent 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 quite small. At the very least, i/o devices, such as the drums, could share a port. In Table II we have assumed two processors per port (for a total of 20 processors).

CONCLUSIONS

An overall argument has been given as to the feasibility and desirability of building a computer to be used in ai research. The design is a very conservative, simple approach built on current computers and technology. Only conventional performance processors were assumed (i.e., each has about the same performance as a 360 Model 85).

Given the overall results, this design provides a basis for specification of the next level of detail. We believe that an approach that departs from a conventional structure (e.g., by placing specific interpretation on ad-

* Cache simulation for LISP interpreter.

dressing) will both decrease the performance and also make the memory too specialized, thereby eliminating unspecified future use that might be made of such a large facility.

The real emphasis of the structure is simplicity, yet it provides much potential power (bandwidth). Also, the design is not presumptuous about how particular future processors will use the facility. In particular, additional power can be gained by developing special purpose processors to be used on C.ai.

Although there are no plans to implement C.ai, a project at Carnegie-Mellon University, the C.mmp multiminiprocessor computer[11] has a similar architecture. Indeed, C.ai has already influenced the structure and instigation of that project. C.mmp is being fabricated and should provide concrete operational evaluation of the design proposed here, particularly the central switch, processor intercommunication, and operating system. The C.mmp machine is being built to provide computing power for speech processing and is thus more than an architectural research experiment.

Multiprocessing is often taken in a rather limited sense, but it can encompass a range of computing modes: parallel processing, pipelining, networking, functional specialization, and independent but cooperating processors. The simplicity of C.ai and the fact that the ai computing environment is so general makes this design well-suited to support research into the various forms of multi-processing.

Just as experimental observations of the physical world and theorems are presented for their own merit, we believe that system designs should be made known and studied as a source of ideas for other designs. Hopefully the architecture presented will serve this purpose.

ACKNOWLEDGMENTS

The design reported here was developed in a project seminar on list-processing machines run by the authors at Carnegie-Mellon University during the Spring of 1971;[9,12,13] the participants worked on the design at all levels and their contributions are acknowledged. Valuable assistance and feedback was provided by Professors A. Newell, R. Reddy and W. Wulf. A Kendziora and Professor J. McCredie provided the section on performance measurement. The referees' suggestions greatly reduced the size and improved the organization and readability.

REFERENCES

1 W H WARE
The ultimate computer
IEEE Spectrum March 1972 p 84

2 C G BELL R CHEN S REGE
Effect of technology on near term computer structures
IEEE Computer March/April 1972 p 29

3 D J FARBER K LARSON
The structure of a distributed computing system software
Proceedings of XXII Polytechnic Institute of Brooklyn Symposium April 1972

4 M J FLYNN A PODVIN
Shared resource multiprocessing
IEEE Computer March 1972 p 20

5 J H BARNES R M BROWN M KATO
D J KUCK D L SLOTNICK R A STOKES
The ILLIAC IV computer
IEEE Transactions on Computers C-17 Vol 8 p 746 August 1968

6 S A HOLLAND C J PURCELL
The CDC STAR-100: a large scale network oriented computer system
Proceedings of the IEEE Computer Conference September 1971 p 55

7 C G BELL A NEWELL
The PMS and ISP descriptive systems for computer structures
SJCC 1970 p 351

8 C G BELL A NEWELL
Computer structures
McGraw-Hill 1971

9 C G BELL P FREEMAN et al
C.ai: a computing environment for ai research
Computer Science Department Carnegie-Mellon University Pittsburgh Pennsylvania May 1971

10 L ROBERTS
Data processing technology forecast
Advanced Research Projects Agency April 1969

11 W A WULF C G BELL
C.mmp: a multiminiprocessor
This volume

12 M BARBACCI H GOLDBERG M KNUDSEN
C.ai(P.LISP)—a LISP processor for C.ai
Computer Science Department Carnegie-Mellon University Pittsburgh Pennsylvania May 1971

13 D McCRACKEN G ROBERTSON
C.ai(P.L)—an L* processor for C.ai*
Computer Science Department Carnegie-Mellon University Pittsburgh Pennsylvania May 1971