# Microprocessor Implementation of a Parallel Processor *

by

Gary J. Nutt
Department of Computer Science
University of Colorado
Boulder, Colorado 80309

#CU-CS-091-76 July 1976

* This work was funded by the National Science Foundation under Grant Number MCS74-08328 A01.

ABSTRACT

Microprocessor Implementation
of a Parallel Processor

A wide variety of uses have been proposed for the spectrum of currently available microprocessor systems. Included in this set of applications is the use of microprocessors for implementing larger systems; here, the possibility of employing bit slice microprocessors for various parts of a multiple control unit SIMD processor is discussed. A brief summary of bit slice microprocessor architecture is given, followed by an outline of individual applications to various components such as control units and arithmetic/logic units of the SIMD processor.

## INTRODUCTION

Large scale integrated circuit technology has created a number of interesting alternatives to computer system designers that were not available with previous technologies. In particular, LSI technology has made great strides in the areas of memory packaging and microprocessor design. The existence of inexpensive microprocessors has led many to consider the design of larger computer systems incorporating multiple microprocessors to implement various functions of the overall system, (e.g., see [1,4,5]). In this paper, the applicability of bipolar bit slice microprocessors to implement various portions of a particular parallel processor system is considered. The general class of computer systems for which this discussion applies includes parallel processors and ensembles, as defined by Thurber and Wald [11]. As a particular instance of this class, the Multi Associative Processor (MAP) system is the subject machine for which microprocessor implementation is considered, [6].

Figure 1 is a general block diagram of the MAP machine; the primary components include an input/output subsystem, a main memory (MM) subsystem, eight control units (CUs), a distribution switch, and a set of 1024 identical processing elements (PEs) each with its own local memory (PEM). The architecture of MAP deviates from the class of SIMD parallel processors in the following respects:

- Eight control units, rather than one, are included to allow up to eight SIMD programs to be executed simultaneously.
- MAP is a stand alone system in the sense that no host processor is used to execute system software, or otherwise control its

operation. All software is executed directly on MAP.

- There is no linear relationship imposed on the set of processing elements through fixed interconnections among the elements; interconnections are controlled by the software.

Briefly, the rationale for such a design is that the system ought to be able to effectively execute a variety of programs on the eight CUs even when some of the programs tend toward sequential operation; this is possible since processing elements are allocatable resources used by each CU as the need arises. Thus, system software, with substantial sequential processing, can execute on one or more CUs at a time, using one PE for sequential tasks and multiple PEs for SIMD parallel tasks such as scheduling and deadlock detection. The absence of fixed interconnections between PEs allows them to be treated as identical system resources to be allocated upon request by a CU.

The design is not without its liabilities; rapid I/O to and from the local processing element memories is not possible as it is in the ILLIAC IV [3]. Processing Element Memories must be loaded via the main memory and control unit to which the given PE is currently allocated. Efforts have been made to make this process as fast as possible under the overall design, [2,6]. The lack of fixed interconnections between PEs limits the speed with which all PEs can simultaneously move data to an immediate neighbor; however, any individual PE can move data to any other arbitrary set of PEs in a single machine cycle.

The architecture of MAP is discussed in detail elsewhere, [2]. Our purpose here is to outline a relatively inexpensive microprocessor implementation of various machine components, (in particular, the control units and processing elements), and the impact of this implementation on the distribution switch shown in Figure 1. Additional description of the

architecture will be provided only as it applies to the microprocessor implementation. The operation of the I/O and main memory systems are not considered in any detail in this paper.

## BIT SLICE MICROPROCESSORS

Although fixed instruction set, MOS microprocessors have been suggested as possible components in larger computer systems, [9], the bipolar bit slice microprocessors such as the 2-bit Intel 3000 series or the 4-bit AMD 2900 series show more promise as components for MAP control units and processing elements. There are three characteristics of these microprocessors that lend themselves to such an application:

- Arithmetic units can be interconnected to implement larger word length units, (e.g., 16 to 64 bit words).
- The microprocessor control units are microprogrammable, allowing quite general operation of the arithmetic unit and additional hardware.
- Minimum cycle times of the processors are as low as 100 ns, although realistic configurations may cause the cycle times to be 2 or 3 times larger than the minimum cycle time.

The speed and generality exists in these microprocessors to use them in control unit and processing element design.

In the remainder of this section, the architecture of the AMD 2900 series slice microprocessor is reviewed, [12]. This microprocessor is briefly discussed since it is slightly more general than the Intel 3000 series in the control unit organization, and it is considerably simpler in its overall design. The basic components are the 4 bit arithmetic unit slice (AM 2901), a 4 bit sequencer (AM 2909), a microprogram control store, and interconnecting logic. An array of two 4 bit arithmetic unit

slices can be interconnected with carry-lookahead units so that arithmetic on words of length 4xi can be executed using carry-lookahead arithmetic. Within the unit, addition, subtraction, OR, AND, exclusive OR, exclusive NOR, and shifting operations can be executed on any pair of 16 registers. Operands can also be passed to the unit from external sources for computation.

The arithmetic unit receives commands specifying the function and register selection indirectly from the sequencer. This sequencer operates on 4 bit addresses (i.e. operation codes), producing 4 bit control store addresses; however, sequencer units may also be interconnected to support larger op codes and a larger control store address space. Sequencer operation is straight forward in that the operation begins by placing an initial control store address in a sequencer register. The sequencer fetches the contents of that address to a microword register, and the contents then specify control signals to the arithmetic unit array, to the sequencer array and to any other external components desired. The sequencer continues fetching control words until interrupted from an external source.

## MAP COMPONENT FUNCTIONS

The input/output subsystem is assumed to be conventional in its design, and is composed of peripheral devices, controllers, and data channels. All I/O is directed to/from the main memory. The main memory may hold three types of information: Data that are being loaded into, or unloaded from, PE memory are temporarily stored in the main memory; global data that may be applied to all data streams under the control of a single instruction stream are stored in the main memory; all instructions are stored in the main memory. One crucial aspect of the MAP architecture

has to do with the main memory design, since memory contention is likely to be a bottleneck to system performance; several potential designs have been proposed and tested using simulations, [7,10]. It is noted in passing that two factors tend to relax the memory contention problem in this context. First, MAP control units will normally be accessing the main memory only during the instruction fetch cycle, (and only occasionally for data references). Secondly, with current memory technology, memory cycle times can be matched to control unit cycle times in a cost-effective manner.

The purpose of a control unit is to provide a mechanism to implement the software notion of a process, where the process itself is composed of parallel tasks; each control unit must manage control and data flow for a particular SIMD program. In order to accomplish this purpose, the following functions must be implemented in each CU:

- Instruction fetching from the MM.
- MM address formation.
- Distribution switch communication.
- Communication with other control units.
- CU instruction decoding and execution, i.e. branch instructions, index register manipulation, etc.
- At least partial decoding of instructions to be executed by PEs allocated to the CU.
- Broadcasting PE instructions and global data over the distribution switch.

Each processing element roughly corresponds to an arithmetic/logical unit in a conventional processor. Computations on a data stream are executed in a PE. The operation of a PE is complicated by the need to selectively activate and deactivate the unit for certain instruction

sequences being broadcast by the owning CU, i.e., depending on local conditions that exist within a PE, the program may wish to deactivate that PE while some sequence of instructions is being broadcasted. In MAP, a Select Register is used to save a set of local conditions, and the PE is activated or deactivated on the basis of a predicate applied to the conditions indicated by this register. It is this aspect of the machine that leads to the term "associative" in the name of the machine, and those instructions which control the activation state are called associative instructions. The list of functions to be implemented in each PE includes:

- Receiving the PE instruction.
- Arithmetic and logical instruction execution.
- Associative instruction execution.
- Data references to the PE memory.
- Receiving/sending data from a CU.

The purpose of the distribution switch is to provide a mechanism for any CU to broadcast instructions or data to any subset of PEs. The switch must also allow data to be moved from any single PE to any number of other PEs allocated to the same CU.

In the remainder of this paper, the implementation of control units and processing elements as bit slice microprocessors is discussed, and a design of the distribution switch is given.

## CONTROL UNIT IMPLEMENTATION

For the purposes of this discussion, assume that the MM word length is 32 bits, with one instruction stored in each word. If a 64 bit word size were used, (with 2 instructions stored in each word), then the CU logic becomes slightly more complicated. A block diagram of an AMD 2900

series implementation of a control unit is shown in Figure 2.

As mentioned in the previous section, the CU has a number of functions to perform, and they cannot all be performed by sequencer chips. Instead, each CU is a complete, bus-organized, microprocessor in itself, with the arithmetic/logical unit used to implement CU functions. Five additional registers are used to communicate with units external to the CU. CUMDR is the data and instruction interface to the main memory system, and CUMAR is the interface to the main memory address register. DBR is the broadcast register used to broadcast undecoded machine-level instructions and data to allocated PEs, and to receive data from PEs. ID is an eight bit CU identification register used by the distribution switch. CUSR is an interface to a CU communication unit to be discussed below. The ALU array, composed of eight 4-bit AM 2901 chips is used to implement other CU programmable registers, (such as the instruction counter, main memory index register(s), and an operand comparison register), and as working registers for the microprograms. Thus address formation and main memory references are handled by the ALU.

Upon fetching an instruction from the main memory, the ALU must do a preliminary decode of the instruction to see if it applies to a CU, to the PEs, or to both (in the case of global data transfers). The design as indicated in Figure 2 implies that this preliminary decoding is done within the ALU array, although it might as easily be done with random logic while the instruction is still on the CU bus. In either case, PE instructions must be routed to the DBR for broadcasting. This aspect of PE instruction processing differs substantially from that proposed in the MAP architecture report, [2], where microinstructions were broadcast to the PEs rather than machine level instructions. By

broadcasting machine instructions, more logic must be incorporated into each PE, but standard LSI circuits can be used in this construction, reducing cost and simplifying the distribution switch. This tradeoff is discussed at length below.

During the instruction fetch cycle, an 8-bit operation code is passed to the Op Code Mapping Memory. This memory is a small (256 x 12 bit) ROM that determines a 12-bit initial address for the microprogram. The 12-bit sequencer array then executes the given microprogram producing j-bit words from the 4096xj bit Control Store. The Control Store word length, j, is unspecified here but must be sufficiently wide to contain control information for the ALU array, feedback information to the Sequencer, to provide signals for the I/O system and distribution switch, etc.. Preliminary designs indicate that j is about 40.

There are at least two aspects of this implementation that lead to complexities. First, since some instructions are partially executed on a CU and partially executed on a set of PEs, timing and synchronization are critical. An example of such an instruction is a global data load operation, which fetches a global datum from the MM and causes it to be loaded into a register internal to each active PE.

The second problem has to do with CU communication instructions. Intercommunication must be synchronized by a unit distinct from each CU, but which communicates with all CUs. A solution to this problem is to incorporate a Control Unit Supervisor into the design, where all communications instructions are executed within this unit. The unit can be constructed as a small bit slice microprocessor that is invoked by interrupts from the CUs. Details of such a unit have not been worked out at this time.

PROCESSING ELEMENT IMPLEMENTATION

The processing element implementation is simpler than that of a control unit, although it does incorporate an ALU array and a sequencer array. Figure 3 is a block diagram of a bus-organized microprocessor implementation of a PE.

The most complex part of the PE design is the interface between the distribution switch, (Data Bus in the figure), and the internal PE Bus. As discussed later, the Data Bus is shared among several PEs, thus a mechanism must be provided for gating data between the Data Bus and the PE Bus. The 8-bit OWNER register is used for this purpose by comparing the ID tag on the Data Bus with OWNER. If the compare is true, then data can be gated between the two busses.

Since machine level instructions are being broadcasted to PEs, a sequencer array is used to decode the instruction into a microprogram exactly as in the design for a CU. However, each microinstruction is much simpler than in the CU case, since the unit need not be concerned with instruction addresses, peripheral I/O, nor distribution switch control. The set of microprograms for a PE is much simpler than those used in the CU microprocessor.

One aspect of the design that should be mentioned is that of activity state computations, i.e., executing associative instructions and de-activating a PE. This is handled entirely by the microprograms, with the Select Register being implemented as an internal ALU array register. If the PE is logically deactivated, a flip flop in the State and Synchronization Unit records this event. To simplify timing and clock enable hardware, this flip flop simply disables the Microword Register outputs until a new associative instruction redefines the PE to be active. Thus, an inactive PE still executes a microprogram, but the

effect of the microprogram is negated.

## THE DISTRIBUTION SWITCH

The distribution switch offers the most challenge for a cost-effective design. Although it does not incorporate microprocessors in its organization, its design is heavily influenced by the use of microprocessors to implement CUs and PEs. The basic requirement is to allow any of eight CUs to broadcast information simultaneously to any subset of the 1024 PEs.

Before considering the microprocessor implementation, the organization shown in Figure 4 was proposed. Each CU was presumed to be broadcasting PE microinstructions at a frequency of 10 $MH_z$; each microinstruction was 56 bits in width. Because of the high broadcast frequency and narrow bus width, a full 8x1024 crosspoint switch was included for instruction broadcasting. It should be pointed out that the switch did not incorporate resolution and queuing hardware at each point, because crosspoint conflicts would be avoided by the PE allocation mechanism. Nevertheless, an expensive proposition.

Data broadcasting used an entirely separate path, as indicated by the solid lines in Figure 4. The frequency with which PE operand addresses and data were to be passed to the allocated PEs was much lower than the instruction broadcast frequency, (and was a function of the program mix being executed). This led to the strategy, shown in Figure 4, called bus sectoring. A bus sector is a crossbar that may be connected to any CU data bus, but is shared by a number of PEs. In order to test the feasibility of such a design, and to determine the number of bus sectors required, an intensive analysis of bus sectoring under realistic machine operating conditions was performed, [7]. The load for the bus system

was obtained by interpreting and monitoring MAP assembly language programs one-at-a-time. Combinations of programs were then used to create the load on the data bus system. The results from the study indicated two interesting facts: The number of bus sectors required should be in the range of 8-12; the scheduling of PEs to CUs had a significant effect on the amount of bus contention experienced in the model. Further studies into PE allocation algorithms uncovered some cost-effective strategies for PE allocation, [8].

The proposed microprocessor implementation employs an entirely different distribution switch design than originally proposed, since machine level instructions, rather than microinstructions, are to be broadcast to PEs. The motivation for this change lies in the micro-instruction width used by the AMD 2900 series; 25 bits of information must be supplied in order for a Sequencer Array to specify the operation of an ALU array. Furthermore, information flow is bi-directional, since the sequencer must know the status of arithmetic operations, e.g., overflow conditions, zero detection, etc.. These requirements render the original instruction bus design untenable.

The strategy employed in the microprocessor implementation takes advantage of the lower instruction broadcast frequency required by the design outlined in the previous two sections. The instruction bus and data bus are combined into a single sectored bus system corresponding to the data bus (solid lines) in Figure 4. To investigate the feasibility of this approach, the following simple analysis of bus utilization is presented. First, a rough set of microprograms to implement the MAP instruction repertoire were written; (the microprograms were "rough" in that several details of the microcode, as well as CU communication instructions, were omitted). The microprograms were used to obtain an estimate of the amount of time required to execute each machine instruction.

The average PE instruction execution, excluding floating point arithmetic instructions, required about 9 microinstructions, and the average CU instruction execution required about 6 microinstructions. Experience with MAP programs indicates that no more than one out of every four instructions executed is a CU instruction. Thus, each CU broadcasts an instruction over the set of sectors in which it contains PEs about once every 8 cycles. Assuming a maximum of one datum broadcast per PE instruction, the CU must use one or more bus sectors, (depending on the program and PE scheduling algorithm), about once every 7 cycles. This argument indicates that bus sectoring for machine instruction and data broadcasting is feasible. It does not show conclusively that the scheme will be cost-effective, nor does it indicate the number of sectors that should be configured into the machine. A more detailed analysis, using simulation, is currently being done on the design. Although this analysis is incomplete at this writing, preliminary results support the above conclusion and further indicate that no more than 16 sectors will be required.

## CONCLUSION

Using microprocessors as a medium of implementation for a parallel processor is promising in a number of respects, not the least of which is economics. Another area of promise is the generality provided by the approach. Since each PE is microprogrammed, it would be possible to implement the control store as a read/write memory. This would allow firmware to dynamically reconfigure the machine in two interesting ways. First, each PE control store could be loaded by its owning CU to define the set of actions caused by the machine instructions broadcast by that

CU. In this manner, PEs are tailored to fit the process being executed on the CU. A more exotic approach would allow different PEs allocated to the same CU to have different microprograms, thus distinct operations would be applied to distinct data streams under a single instruction stream. It is difficult to see how one might write effective software to take advantage of this latter approach, although the possibility does exist.

The overview of the bit slice microprocessor implementation of a parallel processor given in this paper is intended to show the feasibility of such an approach. The resulting implementation could be built at a cost much less than that of a random logic design; conservative estimates of the chip costs are $1000/CU and $750/PE. The economic savings in the distribution switch would also be substantial. The resulting system would have a lower instruction execution rate, perhaps by a factor of 2, than the random logic design if timings are based on current technology. This reduction in speed is offset by the cost, and with the realization that bit slice microprocessor times are likely to decrease in cycle times as newer circuit technologies are used to build microprocessor chips.

ACKNOWLEDGEMENT

The author wishes to thank the National Science Foundation for supporting this work under Grant Number MCS74-08328 A01.

REFERENCES


[1] Arden, B.W. and Berenbaum, A.D., "A Multi-Microprocessor Computer System Architecture", Proceedings of the Fifth Symposium on Operating Systems Principles, (Nov., 1975), pp. 114-121.

[2] Arnold, R.D. and Nutt, G.J., "The Architecture of a Multi Associative Processor", University of Colorado, Department of Computer Science, Technical Report No. CU-CS-070-75, (June, 1975).

[3] Barnes, G.H., et al, "The ILLIAC IV Computer", IEEE Transactions on Computers, Vol. C-17, No. 8, (August, 1968), pp. 746-757.

[4] Codespoti, D.J. and Maryanski, F., "Simulation Model for Microprocessor Enhanced Operating System", Kansas State University, Department of Computer Science, Technical Report No. CS76-01, (Jan., 1976).

[5] Colon, F.C., et al, "Coupling Small Computers for Performance Enhancement", Proceedings of the NCC, Vol. 45, (1976), pp. 755-764.

[6] Nutt, G.J., "A Parallel Processor for Evaluation Studies", Proceedings of the NCC, Vol. 45, (1976), pp. 769-775.

[7] Nutt, G.J., "Memory and Bus Conflict in an Array Processor", to appear in IEEE Transactions on Computers.

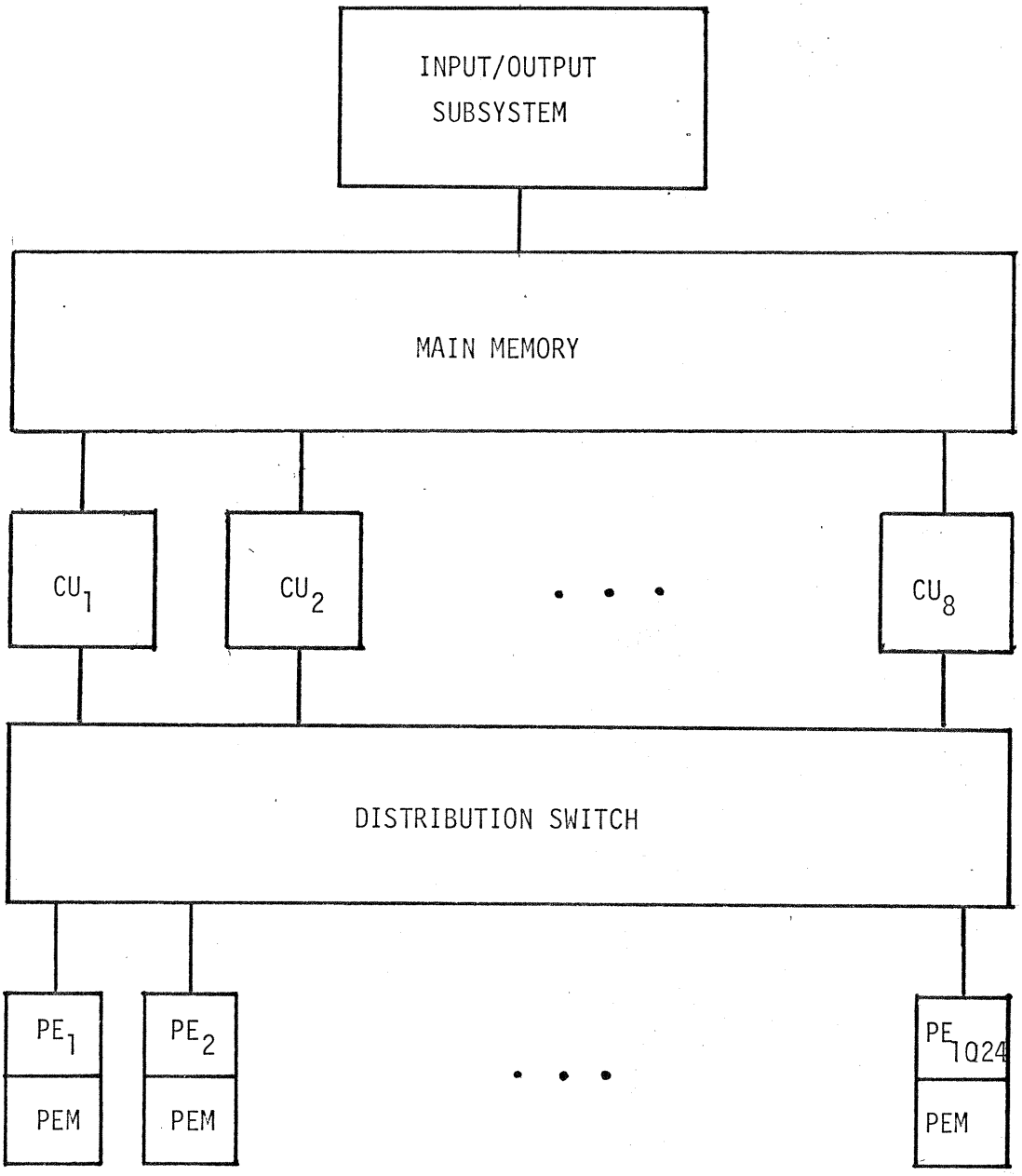
[8] Nutt, G.J., "Some Resource Allocation Policies in a Multi Associative Processor", to appear in Acta Informatica.

[9] Radoy, C.H. and Lipovski, G.J., "Switched Multiple Instruction, Multiple Data Stream Processing", Proceedings of the 2nd Annual Symposium on Computer Architecture, (Jan., 1975), pp. 183-187.

[10] Siegmann, H., "Design and Simulation of a Main Memory-Control Units Interface for the Multi Associative Processor", University of Colorado, Department of Computer Science, M.S. Thesis, (July, 1976).

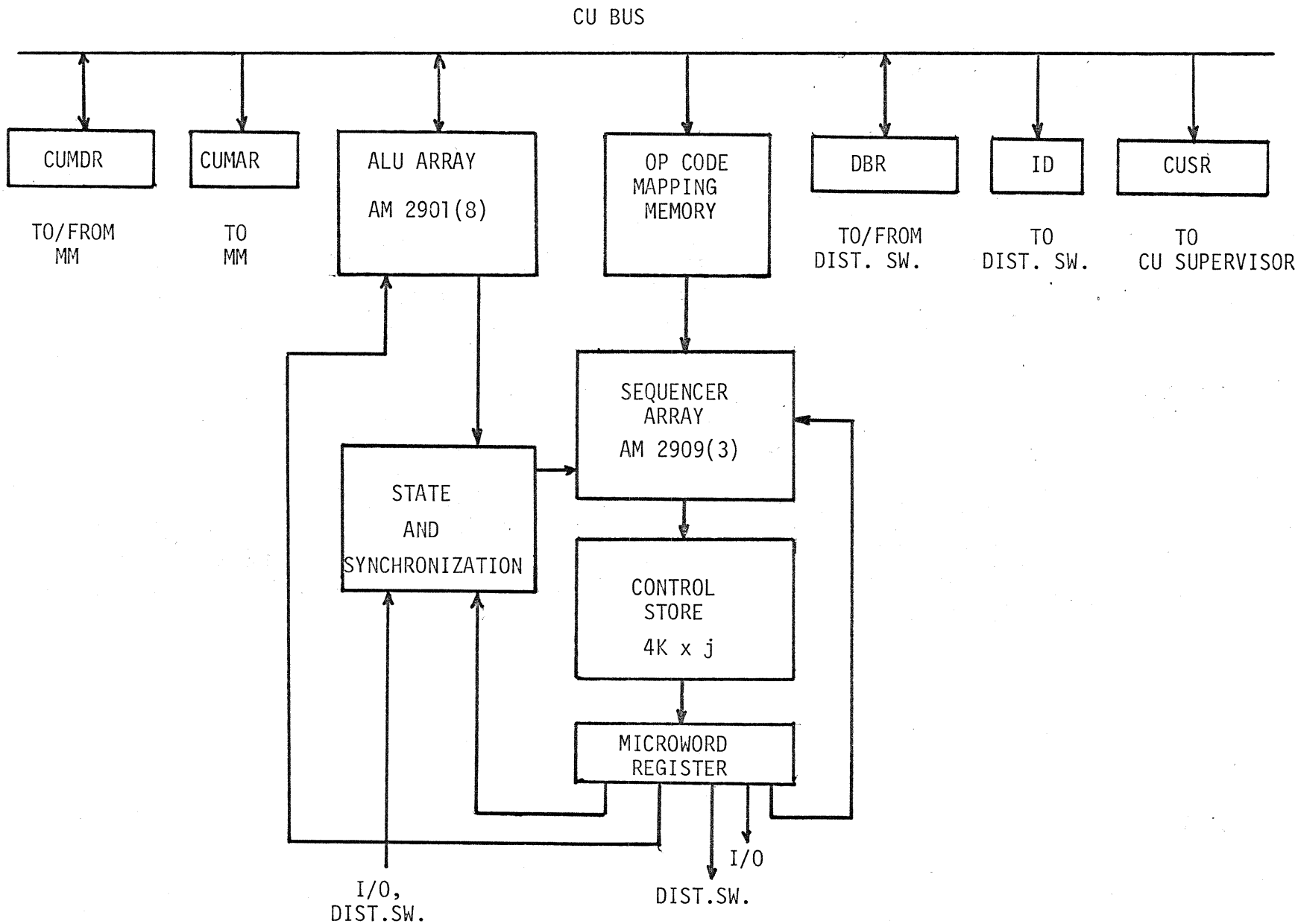
[11] Thurber, K.J. and Wald, L.D., "Associative and Parallel Processors", ACM Computing Surveys, Vol. 7, No. 4, (Dec., 1975), pp. 215-255.

[12] _____, "AM 2901, AM 2909 Technical Data", Advanced Micro Devices, Inc., Sunnyvale, California.



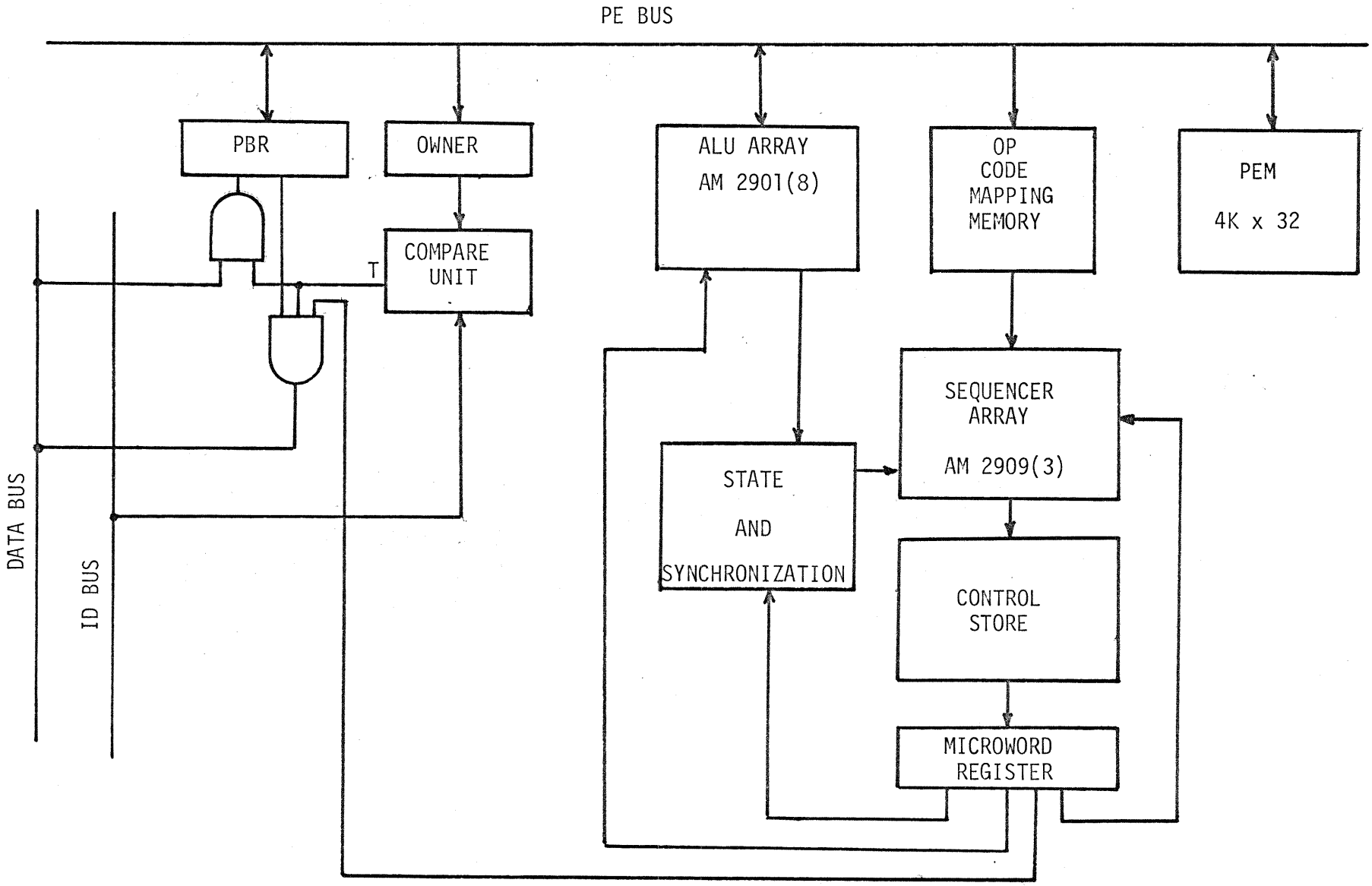

MAP BLOCK DIAGRAM

Figure 1




CU ORGANIZATION

Figure 2




PE ORGANIZATION

Figure 3

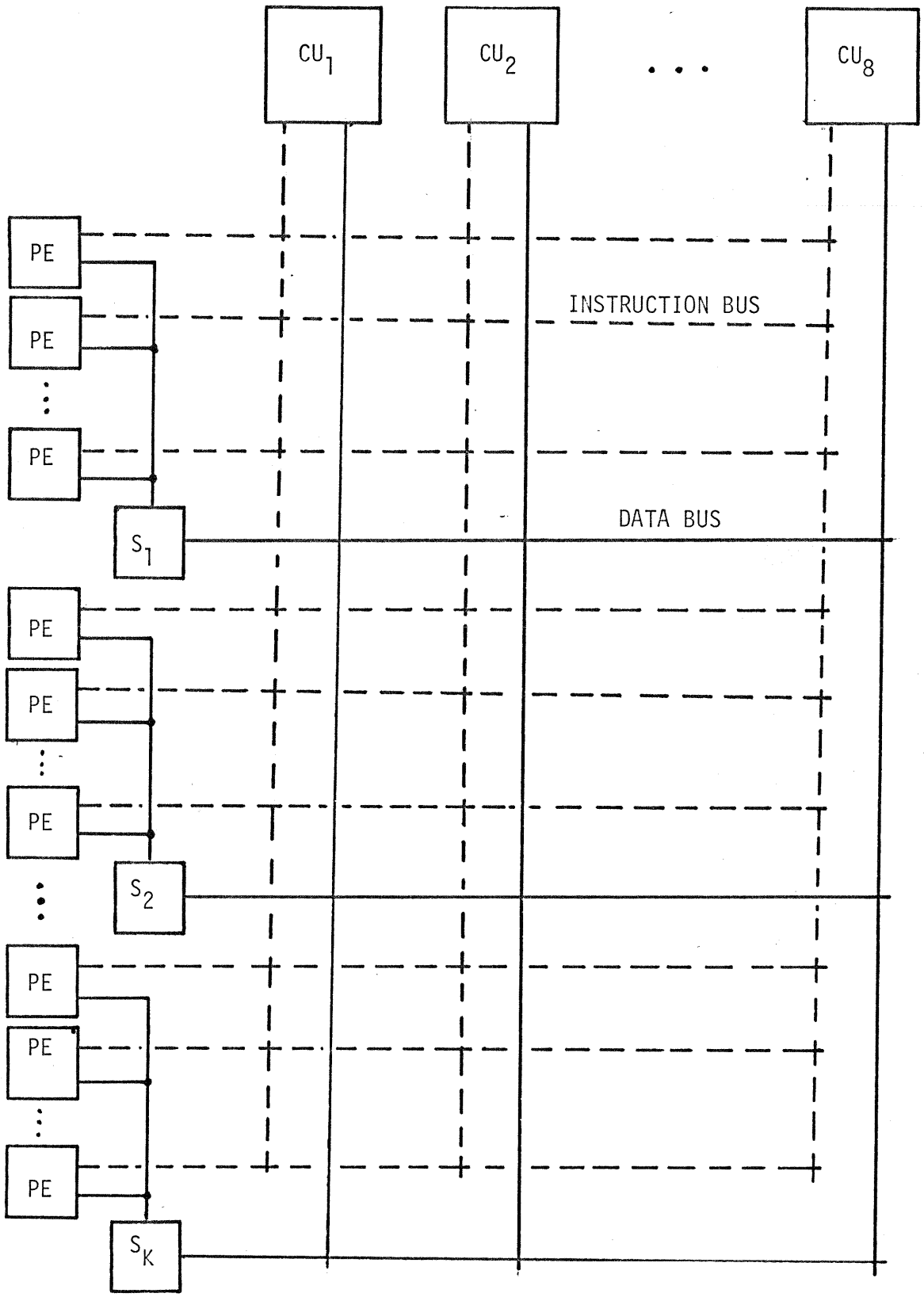

CU1
CU2
CU8
PE
INSTRUCTION BUS
DATA BUS
S1
S2
SK
DISTRIBUTION SWITCH

Figure 4