EVALUATION DURING MACHINE DESIGN: A CASE STUDY

by

Gary J. Nutt
Bruce W. Sanders
Kimbal S. Smith
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309
(303) 492-8728

CU-CS-125-78 October, 1977

ABSTRACT

A case study of an emerging design methodology for multiple processor computer systems is described, where the technology is based on bit slice microprocessors. The evolving techniques emphasize multi-level simulation models. At the lowest level is a bit slice microprocessor interpreter for detailed investigation of the functions which can be microprogrammed and designed into these components. The next higher level model is a machine language interpreter whose detailed characteristics are determined by the results gained from the lowest level model. At the highest level are several models to investigate operating system and architectural strategies. The same approach is applicable to many design studies of multiple processor systems implemented as VLSI chips or bit slice microprocessor technology.

KEYWORDS

simulation modeling
parallel processors
multiprocessor architecture design
design evaluation

INTRODUCTION

An important consideration in the design of computer systems has been to enable the combined software and hardware system to utilize the processor(s) as much as possible. Multiprogramming was introduced so that several processes could share a common processor, thus ensuring that a costly resource (the processor) was more heavily utilized, perhaps at the expense of some less costly resource (e.g. memory). This approach of maximizing processor utilization has had a profound influence on the direction of operating system and architecture research. One basic argument for employing virtual memories was that multiple level memories could be used to more effectively utilize both the processor and the first level of store, [2]. Large scale integration of circuit components has changed the set of design criteria for computer systems; the costs of memory components and processors have decreased to the point that these parts of the system are relatively insignificant and their utilization need not be a constraint for optimization. Although many current aspects of (operating) system design will continue to be important, others such as multiprogramming or otherwise multiplexing a processor will be deemphasized.

The changing character of the economics is not the only reason for altering design constraints; although processors and memories are now much less costly, their architectural components are vastly different. Constraints on the amount of primary memory are no longer determined by cost, power dissipation, etc., but by the address space in a processor that accesses the memory. Processor technology has also changed drastically; eight bit, fixed instruction set microprocessors are now available for twenty dollars. These microprocessors will probably never replace large scale main frame processors, but they have become a popular way to build units to handle peripheral devices, [18, 19]. Future microprocessors will incorporate larger and larger word sizes, as well as incorporating more comprehensive instruction sets; even now, sixteen bit MOS microprocessors have become available, [20].

There are two other developments in hardware that will also have an impact: The first is Very Large Scale Integration (VLSI), and the second is the availability of bit slice (bipolar) microprocessors. VLSI circuits are expected to be up to 100 times as dense as current technologies, allowing multiple functional units to be incorporated into a

single chip, [4,5]. For more units to be placed on the chip, the logic designer will have to be familiar with the software systems that the chip will support, (or operating system designers will have to start designing VLSI chips). In order for the density to be used adequately, well planned system architectures will have to be used.

VLSI technology is still in the future, but bit slice microprocessors are currently available, [19]. These units are modularly constructed so that several bit slices can be cascaded in order to form large word length processors with relatively fast processor cycles. Because of the lack of a priori knowledge as to the end use of these components, these chip sets are intended to be microprogrammed. It is possible to construct a fully microprogrammed, 64 bit word processor for a very modest cost (a few thousand dollars), e.g. see [18].

The result of these technology advances is that future computer architectures are likely to employ several processors, and the design of the processors themselves, their interconnections, and their supporting facilities will be strongly influenced by memory costs, microprocessors, and VLSI technology. The system designer will have to be able to work with LSI and VLSI components in order to design the total software/hardware system; integration of the two fields can no longer be delayed.

The design methodology will also have to evolve in order for computer architectures to be successfully built in this environment. Circuit designers and logic designers have often used simulation models to aid them in their design, e.g. see [17]; similarly, operating system designers have used analytic and simulation models for designing software systems, e.g. see [15]. The mew methodology must incorporate new tools for modeling and evaluating systems during the design phases; the components are at a higher level than those normally associated with logical design, yet, at a lower level than those of operating system design.

At the University of Colorado, a project has been in progress in which we have been forced to devise approaches to design incorporating bit slice microprocessor components. In the remainder of this paper, we discuss our experiences with this design project as they relate to an emerging methodology for designing architectures with (V)LSI components.

CHOOSING A FAMILY OF ARCHITECTURES

There are a number of very high level choices that can be made about the general family of machines in which to design. For example, in a multiple processor machine, how closely coupled should the processors be? The spectrum ranges from SIMD machines which synchronize at the beginning of each instruction cycle, through MIMD machines in which the processors may synchronize at task or job initiation, to networks of processors where the processors may never synchronize. A number of papers have been written about parameters which should influence the choice of one architecture over another at this general level, e.g. see [3]. In the case study discussed here, the general architectural family that was chosen was a variant of the SIMD machines. The Multi Associative Processor (MAP) family of machines all incorporate multiple control units (multiple intruction streams) each of which uses a set of processing elements to implement a SIMD computation. Details of MAP are available elsewhere, [1,9]; only a brief synopsis of the machine is given here.

Figure 1 is a block diagram of the MAP architecture; the main components are an input/output system, a main memory, a set of control units, a distribution switch, and a set of processing elements. The machine operates without the aid of a host machine, hence the operating system program is executed by a combination of one or more control units and one or more processing elements, [13]. Programs and data are loaded into the main memory by the input/output system; data that are to be operated on in parallel (by a single instruction stream) are loaded into a subset of processing elements allocated to one control unit. The control unit decodes a single instruction stream from the main memory, broadcasting commands to the appropriate subset of the processing elements over the distribution switch.

The first level of refinement to the overall design was to specify the main memory and distribution switch organizations in more detail. This refinement initially consisted of specifying the memory system as consisting of multiple, distinct modules that might be interleaved or organized as blocks of consecutive addresses. (It was not known how many such modules should exist in the system.) The distribution switch

was specified to be a variant of a crossbar switch, since the unit was required to allow any control unit to broadcast a set of commands or other information to any arbitrary subset of the processing elements simultaneously. In order to further investigate the applicability of these decisions and to refine the details to a lower level, models were constructed of the designs, [12]. The results from these models indicated that the approaches were feasible provided that certain design parameters were used. The main memory system would have to be composed of at least as many (consecutively-addressed word) modules as there existed control units. The crossbar switch variant, in which groups of processing elements shared a crossbar bus, was feasible provided that suitable processing element allocation algorithms were used in the operating system.

Once these two critical portions of the design were analyzed, more detailed modeling and evaluation could take place within the family of architectures. These models essentially indicated that the family was worth further, more detailed study.

The methodology used for this phase of the study was a prototype for modeling and evaluation work on MAP since that time (1974-1975). The system was modeled at two levels: the lowest level was an interpreter executing a primitive instruction set. This interpreter was used to test potential applications of the architecture and to provide a realistic workload for such a machine. The workload was monitored, and used to drive the second level of models of MAP. These second level models were explicitly prepared to investigate memory and bus contention. The technique was very successful in the sense that architectural family characteristics could be thoroughly tested. However, the interpreter did not model an exact hardware implementation; it relied on detailed designs that were unrealizable at the time. As work progressed within the particular family of architectures, the modeling methodology became more precise, (as described in the next section).

DESIGNING WITHIN THE MAP ARCHITECTURE

The MAP architecture incorporates multiple control units, each of which can fetch and decode a single instruction stream, causing execution of that instruction stream to be applied to multiple data streams implemented in processing elements (one data stream per processing element). In order to have a testbed for application programs, a number of different MAP assemblers and interpreters have been developed. The software is used to create a programming environment similar to that provided by one control unit in conjunction with an arbitrary number of processing elements, i.e., the interpreter is strictly a SIMD interpreter, (except in the Version III system described below). In order to speed up the interpreting process, all programs to be executed are translated into an absolute binary object program; source programs are written in a symbolic assembly language.

The Version I assembler and interpreter were written in 1974 in the assembly language of the Control Data 6000 series computers. This "backward" approach to software development was justified by the classic reasons for choosing an assembly language over a higher level language, namely, the sequential computer would have to simulate parallel execution of individual instructions of the MAP machine, therefore the simulator would have to be as time-efficient as possible. The Version I instruction set was derived by comtemplating the proposed architecture and by hypothesizing the character of programs. The first instruction set was, therefore, designed with no detailed knowledge of the structure and character of programs for the machine; this circumstance is not unlike that of hardware designers building a machine without carefully discussing the applicability of the design with the software designers. (The result is predictable.) One other basis for the Version I instruction set was that it could be implemented on a microprogrammed machine; microinstructions, rather than machine instructions, could be broadcasted to the processing elements. Unfortunately, the detailed design of the underlying machine were incomplete due to missing detail or a reliance on better technology before a part of the system could ever be implemented. During these first days of the design, the goals of the architecture were to test

its feasibility for construction sometime in the future, and to provide a medium for studying measurement and evaluation techniques. Thus, the impracticality of the design was not of serious consequence.

The Version I study was educational and set the stage for a viable methodology for investigating the machine. The object program interpreter not only provided a testbed for investigating applications based on an experimental instruction set, but it offered a mechanism by which detailed measurements of machine performance could easily be taken. These measurement data were then used to drive several of the higher level models relating to the asynchronous, parallel operation of the control units, as described above. One other technique that was "discovered" during the Version I studies was that of applying the Control Data 6000 COMPASS assembler macro capability to build symbolic assemblers for MAP, [14]. This technique was subsequently used on other versions of the machine, and has also been used by others to assemble code for fixed instruction set microprocessors, e.g., see [16].

The Version II Study

The Version II interpreter and assembler were written in the latter part of 1975. Version II employed the same techniques developed during the previous phase of modeling and presented a more realistic instruction set with regard both to software design and current hardware sophistication.

Whereas the Version I software had been designed with space and execution cost constraints (a rigidity which led rapidly to the design of Version II), the new interpreter was created almost entirely by deeply nested macro expansions thereby facilitating the ease with which the instruction set could be adjusted, i.e., one motivation for redesigning the interpreter was to provide for easier maintenance and alteration. Furthermore, Version II supplied the programmer with a virtual machine environment closely modeling true hardware configuration (multi level memory, distinct processors, etc.).

Included in Version II was a simulated hardware monitor which served primarily for program tracing, but also enabled one to perform

crude statistical analysis of hardware utilization. The interpreter also included instruction count mechanisms and, in an attempt to categorize execution times, the execution path was monitored using the CDC 6000 series specifications to model reasonable instruction timings. (The CDC 6000 specifications were intended to provide reasonable ratios of execution time between any two MAP machine instructions.)

Version II became the center of a small, though intense experiment in software design. Early programs attempted to implement algorithms designed for serial machines on the SIMD architecture. It became apparent, though not suprising, that a considerable "twist" in many algorithms is required to make efficient use of the multiple processor architectures; those transformations are not always obvious (for example, the use of Gauss Jordan elimination rather than Gaussian techniques for the solution of linear systems.) [8].

Several major codes were written including software designs for network simulations and studies in artificial intelligence, (the latter remains an active research project). All codes exceeded 1000 assembly statements and executed in times measured in hundreds of virtual run-time seconds. These codes provided sound tests of both the hardware configuration and the instruction set.

While Version II was able to establish the viability of the MAP architecture, it left many questions unanswered and had, in a period of some eighteen months, accumulated a significant number of grievances.

Of primary concern was the fact that the MAP architecture is truly a vector of SIMD machines creating a MIMD configuration. The Version II design included several constraints barring a simple extension to a MIMD (multiple control unit) interpreter. Foremost, data structures and FORTRAN supporting modules had been designed without considering expansion of the model, therefore, neccessitating a major rewrite if an extension was to be successful. While the use of macros to implement the instruction set enabled ready modification to individual facets of the interpreter (the instruction set in particular), they produced an unwieldy mask clouding the internal workings of the interpreter.

These problems aside, an attempt was made to extend the Version II interpreter so that it would simulate the activity of multiple control

units. However, the extended Version II interpreter had significant synchronization problems since the SIMD interpreter was inherently designed with the single virtual instruction step being indivisible.

A second problem which plagued the Version II interpreter was its inability to provide accurate definitions for those instructions needed to test systems designs. While a reasonable representation of the MAP architecture was presented for pure applications codes, Version II fell far short when regarded in this light. As implemented, all system functions were provided by appeals to the FORTRAN run time environment and were therefore inflexible to experiment. (Indeed, it had been tacitly assumed that the MAP architecture would perform poorly where I/O bound processes were concerned. As a result, no serious attempt was made to provide for a reasonable file capability.)

Irregularities and deficiences in the instruction set became more apparent with the writing of large codes. Problems stemming from antisymmetric instructions were easily corrected however, certain missing operations could not be easily implemented, partially because of the host machine's hardware. Further, a notable lack of registers became a severe problem in the more massive applications. Finally, the lack of computing power in the control unit instruction set forced any operation more complex than integer addition and subtraction to be distributed to one or more processing elements.

Yet another problem with the Version II software was its size and execution speed. Assembly and simulation of MAP programs were quite costly. Since no reasonable method for correction of object files was developed, errors almost always necessitate reassembly of entire soure programs.

The problems with the Version II assembler, and in particular, the interpreter, led to a decision that a new simulation package was needed.

The rapidly increasing availability of microprocessors and LSI components made the possibility of realizing a prototype processing element more tangible. If such a device was constructed, it would be desirable to interface it with the interpreter. This approach pointed away from an interpreter executing on a large host machine towards one implemented on a dedicated processor.

The Version III Design

Until design work on the third version of MAP was initiated, an underlying assumption was that the machine need not be realizable in current technology. This resulted in a number of portions of the machine being ignored or assumed to be built with some future technology. As interest increased in the pending third version, (in which the new interpreter was to support multiple control units), it became necessary to consider several of those details more carefully. For example, it was not reasonable to implement a multiple control unit interpreter without considering a mechanism for interprocess communication. The simulation study had become sufficiently detailed that results could be obtained only after the architecture became defined nearly to the point of prototype construction. Versions I and II had been designed "top down", i.e. the underlying hardware details were never firmly defined; Version III is the first in the series of MAP designs to combine "bottom up" techniques with an existing top down design. The Version III design rests squarely on current LSI technology, and in particular, on the generous application of bit slice microprocessors to the design of MAP, [10,11].

Bit slice microprocessors are not as compactly packaged as the more widely known fixed instruction set MOS microprocessors, but such microprocessors are faster and more flexible. The basic chips required to construct a CPU include a sequencer (or control unit), an arithmetic-logical unit, and a control store. Each bit slice ALU can perform a few arithmetic and logical operations on 2-4 bit operands. The ALU will contain logic to perform the operations, to shift operands, and to incorporate a bank of internal registers. Multiple ALUs can be interconnected to form a single ALU that operates on words with 64 or more bits. These resulting ALUs perform relatively fast arithmetic operations, since carry lookahead logic can be used to combine the individual bit slices.

The control unit of a bit slice microprocessor may have a fixed design, (as in the case of the Intel 3001), or it may also be expandable in a manner similar to the ALU, (as in the case of the AMD 2909). The first option restricts the flexibility of design, and the second option forces the designer to provide more chip interconnections. In

either case, the control unit is microprogrammed, and it can be used to economically implement any of a wide variety of instruction sets once the data paths for the hardware have been established.

Bit slice bipolar microprocessors have typical microinstruction cycle times in the range of 100-200 ns. A register-register integer add operation may require as little as one microinstruction for word widths of arbitrary size. The cost of bit slice microprocessor chips varies from about $5/bit (for the control unit and ALU) up to about $15/bit excluding the cost of control store.

The Version III design incorporates microprocessors in a number of portions of the architecture. A MAP control unit implemented as a microprocessor can easily perform the requisite tasks of instruction stream fetching and decoding, while retaining considerable local processing ability. A processing element implemented as a microprocessor can be designed to receive a machine instruction from a control unit and cause that instruction to be executed on a local data stream. Although one tends to rely on SIMD machines to minimize the instruction fetch and decode task, the microprocessor approach actually allows decoding to overlap control units and processing elements. Control units tend to be used as a program counter manager and a main memory access mechanism. A third application of microprocessors to the MAP architecture is a unit to implement interprocessor (control unit) synchronization, [10].

By committing to a microprocessor implementation of MAP, a number of questions in the design began to be answered, e.g., the number of registers available to a MAP programmer is a function of the microprocessor (and microprograms) employed. One goal of the Version III design was to determine an exact set of instructions, with timings, which could be implemented in the microprocessor-based MAP machine. In trying to attain this goal, it became necessary to add another (lower) level to our hierarchy of simulation models, viz., an interpreter for microprograms executed in the bit slice machines.

The Microprocessor Interpreter

The major reason for constructing the microinterpreter was to make it easier to write and test the microcode that would drive the MAP components. Due to the constant evolution of the detailed architecture of the machine (some of the evolution occurring as a direct result of the knowledge gained from implementing the microinterpreter itself), flexibility was needed in the simulator. An easily changeable software interpreter served this purpose much better than a hard-to-change hardware one (viz., the actual microprocessor chips). The study required a simulator to observe the behavior of the isolated components on the microprocessor level without including all the detail of the machine, evaluating architectural as well as logical design of the component. A PL/I-GASP interpreter written by Jayakumar and McCalla, [6], would serve this purpose, but it was structured to reveal detailed timing and system overload problems; a functional representation of the processor that could be molded into the various configurations that would implement the MAP components was needed. The interest was in two aspects of the MAP machine in relation to the microprocessor implementing it: 1) the macroinstruction set and 2) the hardware configuration.

There were several properties of the Version III instruction set that were to be evaluated with the microinterpreter; the major property was determining the difficulty and feasibility of implementing it in microcode. Were there any strange instructions or instructions that were nearly impossible to implement? How much microcode was required for the entire instruction set? What type of actions could be performed quickly and easily? Were there unconsidered instructions that could be performed which might be useful? The answers to these questions could provide information that might effect change in the MAP instruction set. Although critical timing constraints were not considered, timing on a higher level, the microcycle, was considered. The microcycle count for each instruction is direct input into the next higher level model, enabling more realistic system timing analysis at that level.

Architectural properties of the microprocessor implementation of MAP components were to be evaluated also. What was the difficulty of

implementing parts of the MAP hardware with bit-slice microprocessors? Did certain configurations make macroinstructions easier (or more difficult) to implement and faster (or slower) to execute? How could certain parts of the MAP hardware be implemented with microprocessors? How much additional hardware was needed (besides that included in the microprocessor chip set)? Of course, the properties of the instruction set and the properties of the hardware configuration are not totally independent; in fact, the hardware configuration has a direct effect on the answers to the questions in the preceding paragraph concerning the macroinstruction set.

The microinterpreter is an interactive program written in the PASCAL programming language [7]. PASCAL was chosen mainly because of its control structures, which aided in the design of clear, well-structured programs. There were two major difficulties encountered in the implementation of the microinterpreter--1) handling configuration dependencies, and 2) ensuring proper sequencing of events.

The interpreter was written for the microprocessor, not for the MAP components, so that gross changes in the MAP architecture would not render the interpreter useless without major modifications. The interpreter would then be used to simulate the microprocessor in a possible MAP component configuration. Thus, there were many aspects of the interpreter which could not be finalized until some definite configuration was to be simulated. This situation was remedied by the division of the interpreter into distinct parts (viz. user-interface, configuration dependent, configuration independent, and microcycle loop) such that changes in the internal workings of each part had essentially no effect on the other parts. The user-interface part contained all procedures that controlled interaction between the user and the interpreter. The configuration dependent part contained all procedures that were not constant, i.e. all procedures that changed according to changes in the hardware configuration of the MAP component being simulated. The procedures that were not affected by hardware changes were placed in the configuration independent part. The main program was a short sequence of procedure calls representing steps in the microcycle. With this separation of procedures, routines in each part could be altered without having to worry about possible effects on procedures

in other parts. In particular, the configuration dependent part (which tended to be a relatively small collection of noninteracting procedures) could be easily changed.

Ensuring proper event sequencing tended to be a little more difficult. Events were divided into sequences of "related" events that could occur without any interaction with other "non-related" events. For example, microinstruction fetch had to be done before microinstruction execution; microinstruction execution had to be done before flags (e.g. carry, shift) could be set. Each of these three events could be represented by a sequence of sub-events (procedure calls) that did not interact with any other sequence. Thus the main program (microcycle) was just a loop containing calls to events (executed by a sequence of procedures) in their proper order. In particular, a call to the fetch sequence preceded a call to the execute sequence, which preceded a call to the flag control sequence. This allowed easy insertion and deletion of events necessitated by configuration changes in the MAP component.

The Version III Interpreter

Reasons have been suggested for implementing the machine language interpreter on a dedicated minicomputer system; perhaps the most significant one is concerned with the incremental development of a prototype system. If a multiple control unit interpreter is written on a small dedicated system, then one can begin to substitute actual hardware components of the machine for simulated components as the development progresses. Inasmuch as a Data General Nova 1200 minicomputer system was available for an extended period of time, the Version III interpreter was implemented on that system.

The goal of the Version III interpreter is to exactly model a stand alone MAP machine executing at the microprogram level. This will enable the interpreter to be easily modified to reflect changes in the architecture and microprograms as they are developed.

Since a great deal of hardware design preceded the actual development of the Version III interpreter, it seemed reasonable to build a

model which reflected the detailed schemes currently available while including such flexibility as required for further alterations and refinements.

The MAP Version III instruction set was built on sound hardware principles and fortunately, could be emulated on the NOVA 1200 processor with a minimum of difficulty. Perhaps the most elaborate device used on the Version III interpreter was the implementation of the instruction cycle by a series of reentrant coroutines. As details of the microcode actually needed to implement MAP on microprocessors become known, the coroutines can be altered to effect changes in the instruction speed and execution organization. Of equal importance is that the coroutine model provides the necessary synchronization of the virtual control units by simulating the microcycles of the microprocessor. Since the simulation is based on the notion that the smallest functional time unit of the microprocessor is the microcycle, the NOVA machine code group represented in one coroutine block corresponds to the microprogram steps executed by the microprocessor in one microcycle.

The Version III interpreter also provides precise definitions for all interprocess and interprocessor communication instructions as well as a rigorously defined I/O mechanism. However, as with the other instructions, these facets of the interpreter are extremely flexible should design modifications occur.

The Version III interpreter then, is a flexible model of a rigid design which provides a mechanism for experimenting with various architectures. Furthermore, it provides a tool suitable for research of system software design on MIMD hardware configurations as well as for investigating the algorithmic problems arising when writing software for SIMD processors.

SUMMARY

In this paper a chronology of events in the (continuing) design of a parallel processor has been discussed. The emphasis has been on the various models used to investigate facets of the architecture of

MAP during the design phases. The process of constructing models of the components has been a never-ending educational process pointing to inadequate designs, and to the need for a design methodology at the level of hardware and software systems.

In the informal pursuit of a methodology, we have learned several things about modeling during the design process. First, when one investigates the activity of some portion of a system, it is useful to be able to have a flexible level of modeling so that one will not be fooled into designing a machine on unrealizable components; (a lesson hard learned in Versions I and II). In order to be able to change the level of modeling, hierarchical sets of models have been extremely useful; at the lowest level is an interpreter to test microprograms. By using this testbed, one can obtain insight into the uses of such components as well as provide realistic performance data to drive the machine language interpreter. The same application of the second level model, i.e. the interpreter, allows one to understand the utility of an instruction repertoire and the organization of programmable units. It is also possible to observe the interpretation at the middle level to provide realistic driving data for the top level models.

The MAP design work is a case study in design methodology for multiple processor computer systems. The ideas of hierarchical modeling techniques need to be applied to other systems in order to see which parts of our approach are general and which parts are specific to the project at hand.

ACKNOWLEDGMENT

The authors wish to thank the National Science Foundation for support of this work under grant number MCS74-08328.

BIBLIOGRAPHY

[1] Arnold, R. D. and G. J. Nutt, "The Architecture of a Multi Associative Processor", University of Colorado, Department of Computer Science, Technical Report No. CU-CS-070-75, October, 1976.

[2] Dennis, J. B., "Segmentation and the Design of Multiprogrammed Computer Systems", in Programming Language and Systems, edited by S. Rosen, McGraw-Hill Book Co., 1967, pp.699-713.

[3] Flynn, M. J., "Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers, Vol.C-21, No.9, pp.948-960, September, 1972.

[4] Helbig, W. A. and J. D. Stringer, "A VLSI Microcomputer: The RCA ATMAC", IEEE Computer, Vol.10, No.9, pp.22-29, September, 1977.

[5] Henle, R. A., I. T. Ho, W. S. Johnson, W. D. Pricer, and J. L. Walsh, "The Application of Transistor Technology to Computers", IEEE Transactions on Computers, Vol.C-25, No.12, pp.1289-1303, December, 1976.

[6] Jayakumar, M. S. and T. M. McCalla, Jr., "Simulation of Microprocessor Emulation Using GASP-PL/I", IEEE Computer, Vol.10, No.4, pp.20-26, April, 1977.

[7] Jensen, K. and N. Wirth, PASCAL User Manual and Report, Springer-Verlag, 1974.

[8] Nutt, G. J., "Sample Programs for a Hypothetical Computer", University of Colorado, Department of Computer Science, Technical Report No. CU-CS-058-74, October, 1974.

[9] -----, "A Parallel Processor for Evaluation Studies", AFIPS Proceedings of NCC, Vol. 45, pp.769-775, 1976.

[10] -----, "Notes on a MAP Microprocessor Implementation", University of Colorado, Department of Computer Science, Technical Report No. CU-CS-102-77, January, 1977.

[11] -----, "Microprocessor Implementation of a Parallel Processor", Proceedings of the ACM-IEEE Fourth Annual Computer Architecture Symposium, pp.147-152, March, 1977.

[12] -----, "Memory and Bus Analysis of an Array Processor", IEEE Transactions on Computers, Vol.C-26, No.6, pp.514-521, June, 1977.

[13] -----,"A Parallel Processor Operating System Comparison", to appear in IEEE Transactions on Software Engineering.

[14] Nutt, G. J., W. A. Schulz, and K. H. Williamson, "Generating Code for a Hypothetical Computer Using a Production Assembler", Software--Practice and Experience, Vol.7, No.1, pp.147-148, January - February, 1977.

[15] Rose, C. W., "LOGOS and the Software Engineer", AFIPS Proceedings of the FJCC, Vol.41, pp.311-324, 1972.

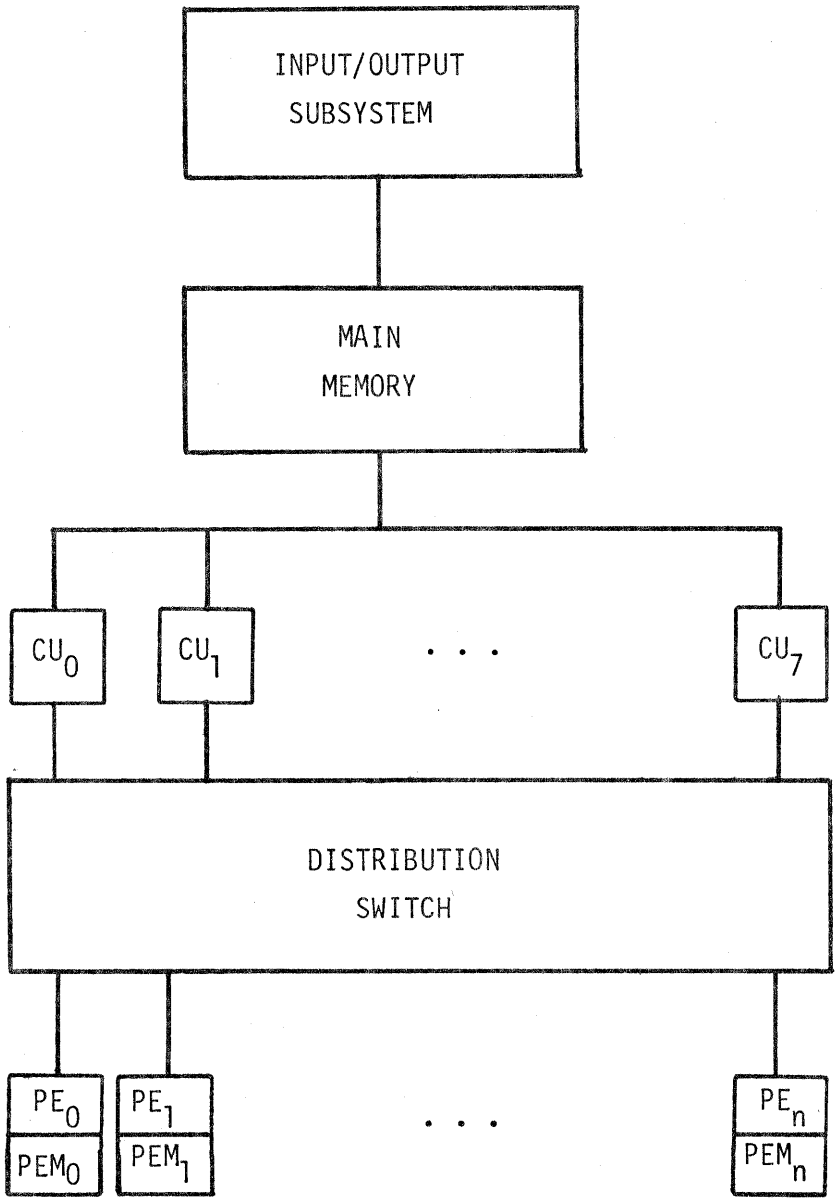
[16] Shustek, L., personal communication, September, 1977.

[17] Szygenda, S. A. and E. W. Thompson, "Modeling and Digital Simulation for Design Verification and Diagnosis", IEEE Transactions on Computers, Vol.C-25, No.12, pp.1242-1253, December, 1976.

[18] -----, Proceedings of the IEEE Spring Compcon 76, February, 1976.

[19] -----, Intel 3000 Series Reference Manual, Intel Corporation.

[20] -----, 990 Computer Family Systems Handbook, Texas Instrument Inc., 1975.



MAP ORGANIZATION

FIGURE 1