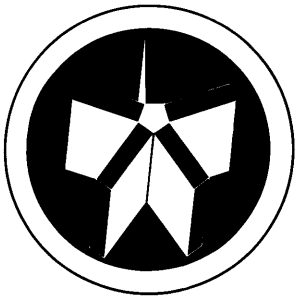
Energy and Technology Review

Lawrence Livermore Laboratory
September 1979

The S-1 Project: developing high-performance digital computers

Under the auspices of the U. S. Navy, we are designing and implementing a multiprocessor (the S-1) with at least ten times the computational power of the Cray-1. Our first step is to develop a general-purpose uniprocessor with a performance level comparable to the Cray-1; the multiprocessor will then be made up of 16 of these uniprocessors, sharing a main memory. The uniprocessors can be used together for large problems or separately for several smaller problems. To reduce average memory-access time, each uniprocessor has a private cache memory. We have also developed a powerful design system (SCALD) that supports extremely efficient structured design of digital logic. Using advanced compiler and verification techniques, SCALD can complete the details of a computer design starting from a high-level specification.



For further information contact
L. Curtis Widdoes, Jr. (422-0758)
or Steven Correll (422-0758)

Our S-1 Project has as its general goal the development of advanced digital processing technology for potential application throughout the U.S. Navy. This work involves the design and implementation of extremely high-performance general-purpose computers.

The basic goals of the S-1 Project may be divided into development-oriented and research-oriented sets.

The primary development-oriented goal is to establish methods for faster design and implementation of advanced digital processors. Our approach to this goal includes the development of a design system that supports structured computer-aided logic design and the development of automated implementation and debugging techniques.

A second development-oriented goal is to provide prototype implementations of highly cost-effective digital processors against which the Navy may measure commercial offerings. We approach this goal in three ways: by developing a durable and extensible uniprocessor instruction-set architecture (the S-1 Native Mode) that will evolve in such a manner that the developing software base is unaffected by changes in the underlying hardware, by designing a common underlying hardware structure for a class of cost-effective, high-performance S-1 Uniprocessors, and by developing a multiprocessor architecture and implementation that allows the S-1 Uniprocessors to be used in a wide variety of applications, particularly those requiring very large computing rates or high operational reliabilities.

Our primary research-oriented goal is to invent and evaluate in use

the concepts and languages necessary to support practical, high-level, general-purpose digital logic design. A second goal is to provide a practical multiprocessor research environment, by implementing multiprocessor hardware with sufficient computing capability to solve real problems of interest to real users. At the same time, we intend to implement and evaluate a fundamental new multiprocessor architecture consisting of a fully-connected network of independent processors, each with a private, hardware-managed cache memory. Finally, we must invent and evaluate operating-system, language, and hardware constructs that will support the partitioning of single large problems across multiple independent processors.

The following sections divide discussion of the Project's work toward these basic goals into three categories: S-1 Multiprocessors, their constituent S-1 Uniprocessors, and the S-1 Design System that supports the design of these S-1 processors.

Multiprocessors

A multiprocessor is a network of computers that concurrently execute a number of independent instruction streams on separate data streams while closely sharing main memory. A multiprocessor design offers significant advantages over a uniprocessor design that provides an equivalent computation rate. The advantages result from the modularity inherent in a multiprocessor architecture and can be categorized as advantages of reliability, economy, and size.

The advantage of reliability has been validated by the very reliable commercial systems that handle, for example, banking transactions and computer network communications.[1] In a well-designed multiprocessor system, failure of a single module (for example, a component uniprocessor, a crossbar switch, or a memory bank) does not entail failure of the entire system. Indeed, the operating system for the S-1 Multiprocessor (called Amber) is intended to detect such module failures and automatically replace the function from the available complement of reserve modules.

Advantages of economy occur during both the design and the construction phases. The design cost per processing element is reduced asymptotically to zero as the processing element is replicated. The economy during construction is extremely important for semiconductors, since the unit replication cost of very large scale integrated-circuit chips varies nearly inversely with the replication factor, except for a small additive base cost.

Another economy is the potential reduction in the time between the design of the system and the delivery of the first operational unit. By replicating a relatively simple processing element many times and using a regular interconnection network, this time lag can be made very small; it is virtually independent of the processing power of the total system. As a result, the semiconductor technology used in a properly designed multiprocessor can be much more up to date than the technology used in a more complex processing structure. One additional economy results from the freedom of the multiprocessor designer to choose the most cost-effective uniprocessor element structure, regardless of the processing rate of the element.

Independent of these economic advantages is the advantage of size. Regardless of whether it is economical to build arbitrarily powerful uniprocessors, at some point it becomes physically impossible (with state-of-the-art technology). Multiprocessors, however, because of their modularity, can have larger processing rates. This advantage of multiprocessor structures is important because maximal computing rates will be necessary for certain applications (numerical weather prediction with its real-time constraints, for example) into the foreseeable future.

S-1 Multiprocessors

We are developing a multiprocessor that computes at an unprecedented aggregate rate on a wide variety of problems. Figure 1 is an artist's conception of the

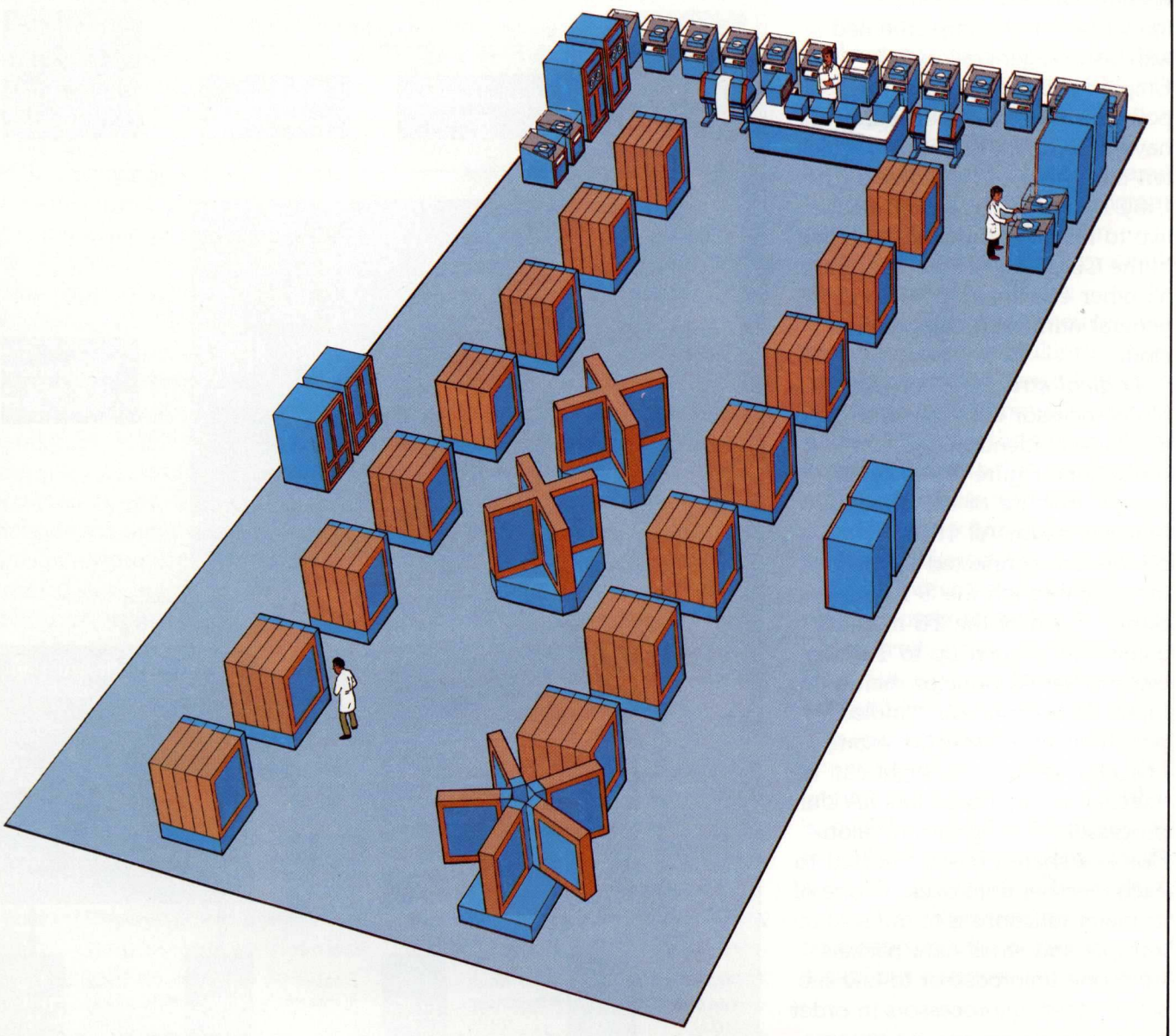


Fig. 1 *The S-1 Mark IIA Multiprocessor system as it might be assembled in a computer center. The system includes 16 S-1 Mark IIA Uniprocessors (the beige and blue booklike devices arranged in two rows of 8 each along the sides of the room), 16 main memory banks (housed 4 each in the 2 blue cabinets on each side of the room near the middle of the rows of Uniprocessors), 2 Crossbar Switches (the X-shaped devices in the middle of the room) for transferring data between the Uniprocessors and the main memory, and peripheral equipment at the far end of the room, including disk drives, tape drives, printers, and a control console. The main memory shown consists of 128 million bytes but is expandable up to 16 billion bytes. The compact arrangement shown is not essential; the Uniprocessors and the memory banks may be hundreds of feet apart.*

system. The S-1 Mark IIA Multiprocessor, to be implemented with second-generation S-1 Uniprocessors, each about as powerful as a Cray-1 computer, will have a computation rate roughly ten times that of the Cray-1. The Cray-1, in turn, has a performance two to four times greater than that of the CDC 7600 and outperforms all other existing computers in general numerical computation work.

Logical structure. A typical S-1 Multiprocessor consists of 16 independent, identical S-1 Uniprocessors. Figure 2 shows the logical structure of the Mark IIA Multiprocessor. All 16 uniprocessors are connected to main memory through the S-1 Crossbar Switch. Each of the 16 memory banks can contain up to 1 billion bytes of semiconductor memory. Input and output are handled by peripheral processors (for example, LSI-11s); as many as eight can be attached to each S-1 Mark IIA Uniprocessor. The Synchronization Box is a shared bus connected to each member uniprocessor; one of its major functions is to transmit interrupts and small data packets from one uniprocessor to any subset of other uniprocessors in order to coordinate processing streams. Each module in an S-1 Multiprocessor is connected to a

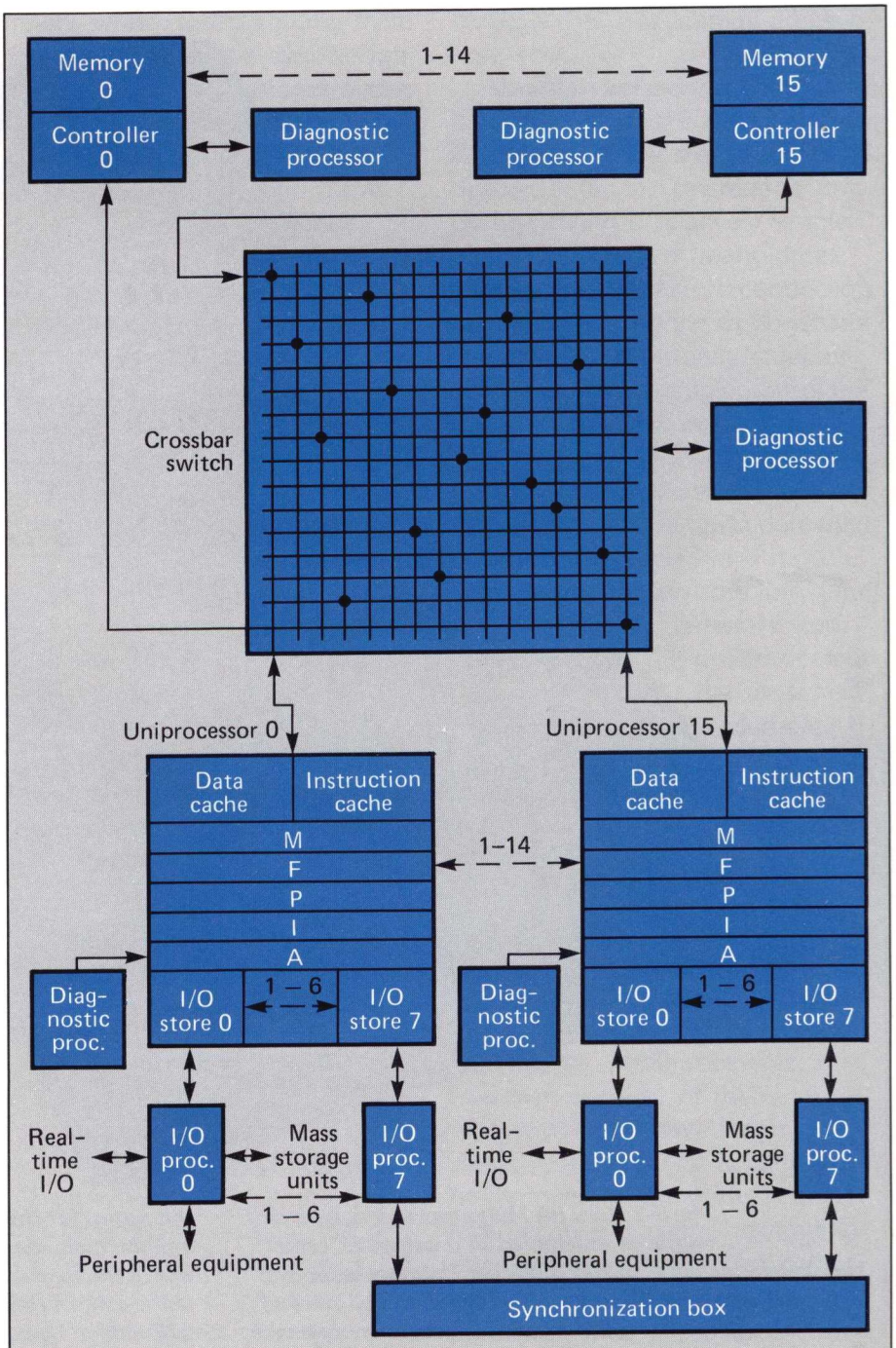


Fig. 2 *The logical structure of the S-1 Mark IIA Multiprocessor, greatly simplified. Only the first and last of the 16 Uniprocessors, one of the two high-bandwidth Crossbar Switches, and the first and last of the 16 main memory banks are shown. As indicated, each of the Uniprocessors communicates with any part of the main memory through the Crossbar Switch. In the access pattern shown (dots at intersections of the Crossbar-Switch grid), each of the Uniprocessors is connected to a different memory bank. When two or more Uniprocessors request the same memory bank, the Crossbar Switch enforces queuing rules that guarantee each contender one turn in the contested memory bank before any other Uniprocessor has two. Private caches with very fast but expensive memory components within the member Uniprocessors effectively hide the combined latency of the switch and memory system.*

diagnostics-and-maintenance processor (an LSI-11) that allows convenient remote display-oriented maintenance and control of the multiprocessor.

All 16 S-1 Uniprocessors can execute independent instruction streams on independent data streams. Thus, all 16 uniprocessors can cooperate in the solution of a single large problem (for example, by means of a Monte Carlo-based algorithm, an increasingly popular and easily partitioned approach to physical simulation). The high-bandwidth, low-latency interprocessor communications provided by the Crossbar Switch facilitate the partitioning of physical simulation problems with little efficiency loss, but the 16 uniprocessors can also process completely independent tasks, so that each S-1 Uniprocessor might service a different set of users. Memory requests from the member uniprocessors are serviced by 16 memory banks with an aggregate maximum capacity of 16 billion nine-bit bytes. Any processor can uniformly access all of main memory through the S-1 Crossbar Switch. The programmer thus sees a huge, uniform address space, because each memory request from each uniprocessor is decoded by hardware in the Crossbar Switch and sent to the appropriate memory bank. The Crossbar Switch has a maximum peak bandwidth of more than 10 billion bits per second when all its 16 channels are transferring data simultaneously.

Cache memory. The design of the S-1 Multiprocessor allows component uniprocessors and memory banks to be physically distributed over distances that are limited only by average bandwidth requirements (which degrade linearly with increasing cable length). To reduce the delays in accessing main memory that result from long cables, Crossbar-Switch transaction time, and relatively slow (but highly cost-effective) memory chips, each member uniprocessor contains private cache memories. These caches automatically (that is, without guidance from the programmer) retain recently referenced data and instructions in a relatively small amount of ultrahigh-performance memory, in the expectation that those data will be referenced again soon. Whenever a reference to such a retained datum or instruction is made, the information is immediately delivered directly from the cache, thus eliminating the need for a main-memory transaction. Although a similar efficiency can be realized if main memory contains a special high-speed area, such a design places on every programmer the burden of managing a variety of memory systems in order to maximize the efficiency of program execution.

The presence of caches in a multiprocessor necessarily introduces problems of cache coherence; that is, each uniprocessor must be able to read or write data in the other caches without any observable inconsistencies.[2] Without a guarantee of cache coherence, programming of certain problems in a cache-based multiprocessor would be inconceivably difficult.

The caches of the member uniprocessors of S-1 Multiprocessors are private in the sense that there are no special communication paths connecting the caches of one uniprocessor with the caches of any other uniprocessor; the cache coherence problem is therefore especially challenging. To solve it, the S-1 Multiprocessor includes a design closely related to one independently proposed in Ref. 2. A small tag is associated with each 16-word line in physical memory. This tag identifies the only member uniprocessor (if any) that has been granted permission to retain (that is, owns) the line with write access and all the processors that own the line with read access. The memory controller allows multiple processors to own a line with read access but responds with a special

error flag when a request is received to grant read or write access for any line that is already owned with write access or to grant write access for any line that is already owned with read access. Any uniprocessor receiving such an access-denial response is responsible for requesting (through a simple interrupt mechanism) that other uniprocessors flush the contested line from their private caches. This procedure maintains cache coherence dynamically, and hence extremely efficiently, without requiring any effort by the programmer.

Error detection and correction. For reliability, all single-bit errors that occur in memory transactions are automatically corrected, and all double-bit errors are detected, regardless of whether the errors occur in the switch or in the memory system. For protection against single-point failures, the S-1 Multiprocessor allows permanent connection of multiple Crossbar Switches that can be selected electronically; the S-1 Multiprocessor can thus continue operating in the event of a single-switch failure. Furthermore, the Crossbar Switch can be configured electronically to keep a backup copy of every datum in memory, so that failure of any memory bank will not entail loss of crucial data. Each input–output peripheral processor may be connected to input–output ports on at least two uniprocessors, so failure of a single uniprocessor does not isolate any input–output device from the multiprocessor system. To make maintenance easier, each member uniprocessor, each crossbar switch, and each memory bank is connected to a diagnostic computer that can probe, report, and change the internal state of all modules that it monitors, with very high time and logic resolution.

S-1 Uniprocessors

We are developing a line of S-1 Uniprocessors to serve as the computational nodes in the S-1 Multiprocessor. The first-generation S-1 Uniprocessor (Mark I) has been implemented and evaluated in use,[3] the second-generation (Mark IIA) machine is under way, and future generations (Mark III, Mark IV, and Mark V) have been planned in varying amounts of detail. These generations of S-1 Uniprocessors vary greatly in performance because of generation-to-generation advances in microcode, hardware structure, and implementation technology. However, all of them can conform to an identical instruction-set architecture, thereby making software transportable from uniprocessors of any earlier generation to those of any later one.

Instruction-set architecture. The instruction-set architecture of a computer consists of those principles of its operation that a programmer without a stopwatch is capable of observing; that is, it includes no timing information. The complete hardware and microcode structure that executes an instruction-set architecture is called the implementation. The implementation of the S-1 Mark IIA Uniprocessor has been designed to allow high-speed emulation of several existing instruction-set architectures, including the DEC-10 and Univac AN/UYK-7, in addition to the S-1 instruction-set architecture (S-1 Native Mode).

It was apparent early in the S-1 Multiprocessor design that no existing instruction-set architecture was suitable to serve as the S-1 Native Mode. Because then-existing instruction-set architectures had been designed under very different technology constraints than those expected to apply to S-1 systems, they variously suffered from address-space inadequacy, insufficient operations-code space, insufficient multiprocessing-oriented features, or adverse implications for high-performance implementations.

In response to this situation, we developed the S-1 Native Mode, which is probably the most widely reviewed high-performance computer architecture ever developed. Unlike the instruction-set architectures of previous high-performance computers (for example, the CDC STAR-100 or the Cray-1), which were developed by a few designers working behind corporate proprietary screens and were then frozen, the S-1 Native Mode has been analyzed, criticized, and revised by scores of computer

scientists, engineers, and application specialists in industry, academia, and Government throughout the country. It has evolved over a period of three years, during design, implementation, and operational evaluation of the S-1 Mark I Uniprocessor prototype and during design of the S-1 Mark IIA Uniprocessor.

As a consequence of this unprecedentedly extensive peer review, the S-1 Native Mode is well developed—it contains a large, consistent set of features; it is highly extensible—it can easily include new features; it is general purpose—it contains features for compiler and operating system efficiency as well as for arithmetic-intensive and real-time applications; and it is carefully tuned—it facilitates high-performance implementations of S-1 Uniprocessors and S-1 Multiprocessors.

The S-1 Native Mode allows the programmer to address uniformly, without using base registers, 2 billion nine-bit bytes of main memory, 288 times more memory than the Cray-1 (although relatively low-performance machines with large address spaces have recently appeared on the market). Indeed, it was primarily to provide for adequate address space that a 36-bit work length was adopted for the S-1 Native Mode.

Huge memories are crucial for efficient solution of large problems, such as three-dimensional physical simulations and Monte Carlo-intensive studies, which are of great current interest in a wide variety of applications that range from incompressible fluid flow studies to acoustic ray tracing in highly stratified media. The large memory addressability of the S-1 Native Mode essentially eliminates the programming costs associated with managing multiple types of computer system storage (for example, the SCM, LCM, drum, and disk memory hierarchy of the CDC 7600, to whose efficient management major portions of the careers of some programmers have been addressed). Memory technology has advanced so far since the development of small-address-space architectures such as the CDC 7600, the DEC PDP-10, and the IBM/360 that the current production cost to the S-1 Project of a 2-billion-byte main memory using 16K-bit memory chips is less than $10 million; its long-term rate of advance is so rapid that this cost can confidently be expected to decline by almost a factor of 2 each year for the next several years.

Most software produced for S-1 systems will be written in high-level compiled languages such as the developing DOD standard language, Ada. For ease of compiler writing and for rapid, efficient execution of the compiled code, these languages require certain features in the underlying instruction-set architecture. S-1 Native Mode is compiler-oriented: it is designed to support high-level languages in general, not one high-level language in particular, and it includes the full set of operators and addressing modes necessary for a simple compiler to produce efficient code. For example, S-1 Native Mode supports expression evaluation with a unique type of 2.5-operand instruction that allows the compilation of almost all forms of arithmetic expressions without using any move instructions (instructions which simply move data from one location to another without performing logical or arithmetic operations on them).

The extent of the compiler-orientation of an instruction-set architecture can be roughly measured by counting the number of instructions necessary to represent typical high-level language programs. We have observed that the CDC 7600 requires between two and three times as many instructions to represent FORTRAN programs as does the S-1 Native Mode; the bulk of additional CDC 7600 instructions are used in addressing computations. An experiment involving seven graduate-student programmers in the Computer Science Department at the University of California, Berkeley, showed that careful hand-coding of the PDP-11 requires an average of 1.5 times as many instructions to

represent a variety of high-level language programs as does the S-1 Native Mode. These and related considerations lead us to assert that no high-performance machine available today has a more compiler-oriented instruction-set architecture than the S-1 Native Mode.

The S-1 Native Mode contains unprecedentedly comprehensive floating-point semantics. Floating-point numbers can be 18, 36, or 72 bits long, using 5, 9, and 15 bits, respectively, to represent the exponent of 2, and 13, 27, and 57 bits, respectively, to represent the signed fraction. The largest format is upwards compatible with the floating-point format of the Cray-1. The 36-bit format was designed to be the workhorse for virtually all numerical applications. The 18-bit floating-point format was specially designed to support real-time signal processing at many hundred million floating-point operations per second, but it can be highly useful in any relatively low-precision application where processing speed is at a premium (as, for example, in Monte Carlo procedures).

Compared to conventional floating-point representations, S-1 floating-point formats offer one extra bit of precision because the high-order bit of the fraction is determined from the sign and is not explicitly represented. The S-1 Native Mode also allows floating-point operations to be correctly rounded in any of several different rounding modes. For example, stable rounding minimizes expected error, and diminished-magnitude, augmented-magnitude, floor, and ceiling roundings can be used to measure the actual error developed. The S-1 Native Mode includes special floating-point symbols (not-a-number, infinities, and epsilons) which allow programs to be created and exercised that will not malfunction because of transient generation of quantities so large or small that they cannot be represented as ordinary numbers in the computer. A computer arithmetic system containing such symbols is essential for efficient use of human resources in developing and using robust computer programs.[4]

Pipelining of instructions. Pipelining is exemplified by an automobile production line, in which a number of automobiles are in production simultaneously, each in a different stage of completion; the time between completion of construction of one automobile and the next is roughly the delay of a stage in the assembly line, rather than the time required for a single car to pass through the entire line. A stream of instructions in a pipelined computer implementation is processed in a very similar fashion.

The S-1 Native Mode was designed especially to facilitate pipelined parallelism in the fetching and decoding of instructions, the associated fetching of instruction operands, and the eventual execution of instructions. Pipelined parallelism is a conceptually simple type of parallelism that can result in extremely high computer performance levels. In general, designers of advanced instruction-set architectures for commercial computers have given little consideration to the implications of extensive pipelining, because they have developed those architectures with medium- or low-performance implementations in mind. Furthermore, pipelining has thus far been used in modern computers primarily in the execution of instructions, where it appears in the streaming of vectors of operands through pipelined arithmetic or logical operation functional units.

S-1 Uniprocessors pipeline *the preparation* and execution of instructions that specify both scalar and vector operations. Every instruction proceeds through multiple pipeline stages, including instruction preparation, operand preparation, and execution. Some stages of the pipeline, particularly those dealing with operand address arithmetic and instruction execution, necessarily have a wide variety of functions, since the pipeline must process a wide range of instructions. This variability in operation is effected through the extensive use of microcode, an architecture-defining, very low-level program that precisely specifies the operation of every pipeline stage. The variability built into the microcode-controlled pipeline facilitates high-performance

emulation of other computers (for example, the Navy's Univac AN/UYK-7). The S-1 Mark I and Mark IIA Uniprocessors are the first high-performance machines to incorporate instruction-preparation pipelines fully controlled by writable microcode.

Structure and performance. Figure 3 shows the internal logical structure of the S-1 Mark IIA Uniprocessor. The machine consists of five microengines (extremely fast, relatively special-purpose programmable controllers) operating in parallel to provide high performance. Four of the microengines form the instruction pipeline, consisting of the instruction-fetch, instruction-decode, operand-preparation, and arithmetic segments. Some segments are internally pipelined (a level of detail not shown in Fig. 3). A single microengine handles memory traffic in parallel with the operation of the instruction pipeline. A one-processor system can be configured by connecting an S-1 Mark IIA Uniprocessor directly to a memory controller; this requires neither hardware nor microcode changes.

During the design of the S-1 Mark I and Mark IIA pipelines, we made significant advances in computer technology. The Mark I introduced a new, simple branch-prediction strategy to predict the outcome of each test-and-branch operation in an instruction stream before its execution, thereby allowing subsequent instructions to be prepared without disruption. The Mark I also refined the use of dual

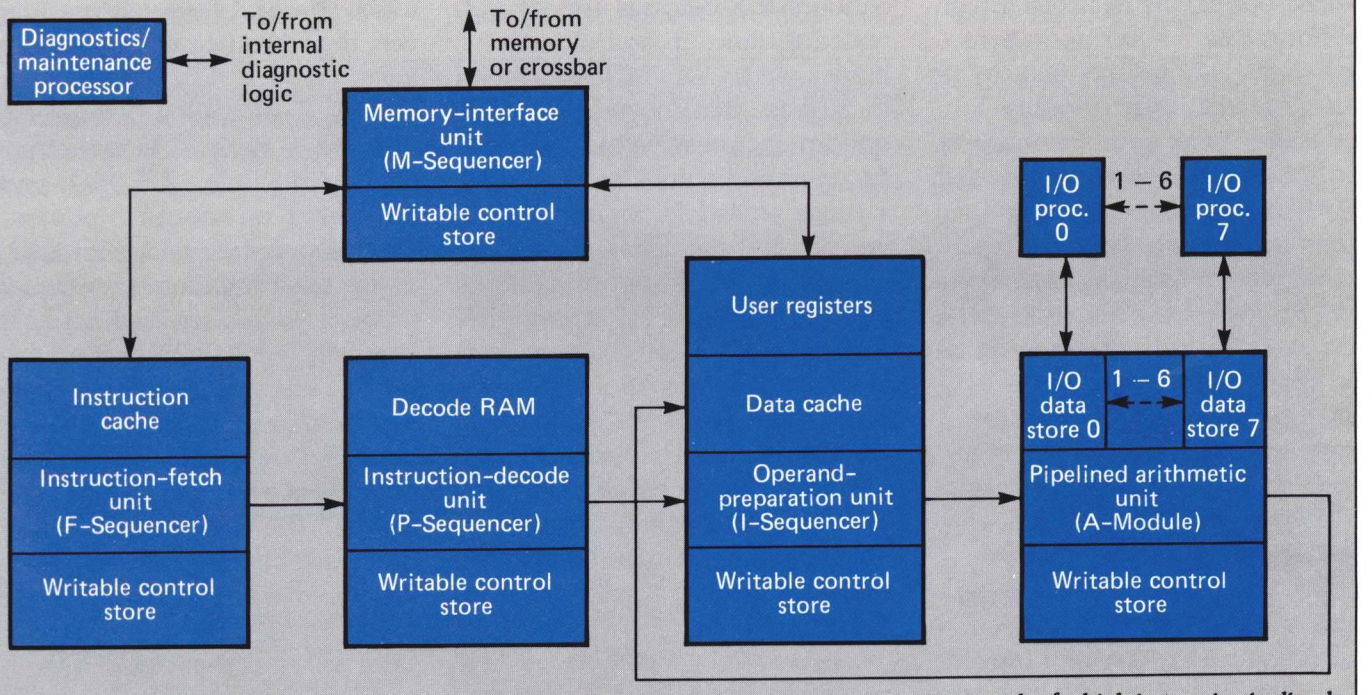


Fig. 3 ***The internal logical structure of the S-1 Mark IIA Uniprocessor. It consists of five distinct microengines that operate in parallel. Four of the microengines control the instruction pipeline, consisting of instruction-fetch, instruction-decode, operand-preparation, and arithmetic segments, each of which in turn is pipelined. One microengine controls memory transactions. All five microengines are controlled with writable microcode. The Uniprocessor also contains private cache memory.***

cache memories (one for instructions, one for data) to increase total cache bandwidth. The Mark IIA allows advance computation of simple operations in early pipeline stages; this technique minimizes idling of pipeline stages because a computation (particularly, an operand-address computation) depends on some previous result. The Mark IIA includes refined control mechanisms to coordinate the operation of multiple pipeline stages controlled by the independent programmable microengines.

The S-1 Mark IIA also employs vector operations to achieve high performance. Vector operations use multiple functional units in the pipelined arithmetic module, to achieve a peak computation rate on the S-1 Mark IIA Uniprocessor of 400 million floating-point operations per second. Any fatal error encountered during a vector operation results in a precise interrupt, so the exact location of the error can be determined by the error-handling routine; this feature is regrettably rare on existing high-performance vector processors.

Status and plans. The S-1 Mark I was developed to be a prototype for evaluating the S-1 Native Mode and its advanced hardware and to provide the necessary computational resources for the development of the S-1 Mark IIA hardware and software. Only one S-1 Mark I has been produced; it began operating late in 1977. Constituted of 5350 ECL-10K integrated circuits, it was designed to execute floating-point arithmetic only in microcode emulation and also contained a severely reduced instruction-preparation pipeline. On a small set of floating-point-intensive scientific benchmark codes written in Pascal, the S-1 Mark I has been observed to compute between 0.3 and 0.5 times as fast as the CDC 7600, although the judicious use of hand-coded routines in crucial inner loops on the CDC 7600 was found to increase that machine's overall performance relative to the Pascal-programmed Mark I to a speed advantage of fivefold, for our 16 000-line physical simulation code. Conversely, the maximum execution rate of the Mark I (10 million instructions per second), when combined with the powerful addressing modes and field-manipulating features of the S-1 Native Mode instruction-set architecture, permits it to execute a variety of non-floating-point-intensive codes significantly more rapidly than does the CDC 7600.

The second generation S-1 Uniprocessor, the Mark IIA, executes the same instruction set (the S-1 Native Mode) as the Mark I, but it has extensive hardware floating-point and vector operation capabilities. Its performance is expected to be comparable to that of the Cray-1 on scientific problems expressed in high-level languages such as Ada, Pascal, and FORTRAN, for just those applications in which the single-word floating-point format of the S-1 architecture is as useful as the substantially higher precision floating-point format of the Cray-1. The Cray-1 will assuredly retain primacy in high-precision, vector-intensive data processing relative to near-term S-1 Uniprocessors, since this type of computing capability cannot be justified for present or readily foreseen Navy applications, most of which stress relatively low precision, very high throughput data processing.

Table 1 shows the performance of the S-1 Mark IIA Uniprocessor compared to the CDC 7600 and the Cray-1 on several important DOE benchmark miniprograms. These miniprograms are representative of the full set used at LLL to compare the performance of advanced scientific computers; they accurately and concisely characterize the computation-intensive portions of extensive scientific code at LLL. The S-1 Mark IIA Uniprocessor computes these benchmarks at roughly the same speed as the Cray-1 and almost twice as fast as the CDC 7600. The CDC 7600 rate was measured using an optimizing compiler first available in 1974. The Cray-1 rates are based on actual performance measurements made in February, 1979, with a moderately mature optimizing and vectorizing compiler supplied by Cray Research, Inc. Although the Cray-1 executes more instructions per second than the Mark IIA, many Cray-1 instructions are expended in overhead

computations. The S-1 results assume the use of 36-bit floating-point numbers, since high-precision arithmetic is often not necessary in LLL applications; however, neither the CDC 7600 nor the Cray-1 provides a low-precision floating-point format. For applications requiring high precision, the Mark IIA supports operations on the 72-bit floating-point format at roughly half the speed of operations on the 36-bit floating-point format. For low-precision applications, the Mark IIA supports operations on the 18-bit floating-point format at approximately twice the speed of operations on the 36-bit floating-point format.

The S-1 Mark IIA Uniprocessor is constituted of ECL-100K MSI circuits in performance-critical areas and ECL-10K circuits elsewhere. All Mark IIA circuits are standard, commercially available products. The transistor population of the Mark IIA's arithmetic unit alone is greater than that of the entire central processing unit of the Cray-1; gate circuit densities within this arithmetic unit are about 20 times greater than those in the Cray-1 central processing unit. The Mark IIA is in development at the present time; it is being packaged in the folded form shown in Fig. 4 (shown unfolded on the cover).

The S-1 Mark III is in an early design phase. Like the Mark I and Mark IIA Uniprocessors, the Mark III executes the S-1 Native Mode, but it is to be implemented completely in commercially available ECL-100K LSI circuits. While it will not achieve a large performance gain over the Mark IIA, it will be physically more compact because of its order-of-magnitude greater logic gate density.

We are moving as rapidly as possible toward using the

Table 1 *Comparison of the performances of the S-1 Mark IIA Uniprocessor, the Cray-1, and the CDC 7600. Data on Cray-1 and CDC 7600 taken from Ref. 5.*

Mini-program	Miniprogram function	Computation rate, MFLOPS[a]				
		S-1 Mark IIA		Cray-1		
		Scalar[b]	Vector[c]	Scalar[d]	Vector[e]	CDC 7600
1	Hydro excerpt	9.1	59	9.3	71	5.3
2	Unrolled inner product	11	74	8.8	47	6.6
3	Inner product	8.0	65	4.4	62	4.6
5	Tridiagonal elimination	7.5	7.5	7.6	7.6	4.0
7	Equation-of-state excerpt	13	46	12.6	80	7.3

[a]MFLOPS stands for millions of floating-point operations per second.
[b]Assumes no use of vector capability.
[c]Assumes full vectorization.
[d]Obtained by turning off compiler vectorization.
[e]Obtained by turning on full compiler vectorization.

technology of very large scale integrated circuits. The first generation presently planned to follow the Mark III will express the entire Mark III architecture on several VLSI chips, at a performance level at least as great as that of the Mark III.

S-1 Design System

The capabilities offered by semiconductor technology for the implementation of advanced computer designs are rapidly outpacing the capabilities developed for articulating the conception of those designs. To make best use of rapidly improving semiconductor technology, we have developed the SCALD (Structured Computer-Aided Logic Design) System.[6]

SCALD is a graphics-based system for designing digital logic. It inputs a high-level description of a digital system and outputs magnetic tapes that are used by commercial automatic wire-wrap machines to build the hardware.

The main advantage of using SCALD is a drastic reduction in the amount of time required to design a large digital system. This reduction occurs because the designer can express his design in the same general level in which he thinks about it, freeing him from the task of actually drawing out all of the details of the logic and creating a wire list specifying its interconnection. Designs expressed in this high-level notation become much more comprehensible for all those who have to work with them—for computers, for computer designers, and for maintenance engineers. By reducing the amount of clerical work required of digital logic designers, SCALD reduces the number of designers required to execute a design project and the

Fig. 4 *The S-1 Mark IIA Uniprocessor. The package consists of identical pages. The pages unfold to expose all wire-wrap pins for maintenance. Ambient air blows up through the centers of the pages to cool the integrated circuits, which are mounted on the inside. Commercially available power supplies are mounted in the cabinet base.*

communication overhead per designer, thus increasing each designer's productivity and further reducing the total designer requirements of the project. Manpower savings well in excess of an order of magnitude may be realized; such savings have actually been demonstrated in practice during both the S-1 Mark I and Mark IIA design efforts.

SCALD allows designs to be recompiled rapidly when new integrated circuits become available; such circuits may simply take the place of low-level modules. Thus, a designer can quite effectively use a previous design to reduce his design time on a new project, thereby taking maximum advantage of the exponential rates of advance in component density and cost-effectiveness currently characterizing the semiconductor industry. In practice, considerable work may still be required to update a design to incorporate recent technology advances, but the required effort is likely to be much less than if the design were not expressed hierarchically.

SCALD also facilitates designing with very high accuracy, because SCALD performs design verification procedures that cannot be done by a human. Not only can SCALD verify syntactic details of the design (for example, that every gate input is connected to some output), but it can also verify that transmission lines are effectively free from signal reflections; it can certify that the logic networks defined by the designer do not contain timing errors, and it can demonstrate by simulation that the logical operation of the design is correct.

Historically, logic design has lagged far behind program design in terms of the ideals of structured design: that arbitrary modules be specified, each in terms of a few other modules, relatively independently, and that they communicate through well-defined interfaces. Logic is still typically hand-drawn by draftsmen; the specification language consists of drawings of the primitive logical elements available from integrated-circuit manufacturers and the physical connections between those logical elements. On the other hand, typical modern programming systems readily support the design of arbitrary modules (that is, routines), each in terms of a few other routines, and allow the specification of tightly structured interfaces between those routines. SCALD simply expresses these performance-proven software-engineering concepts in the world of hardware design.

SCALD consists of a set of computer programs. The Graphics Editor[7] enters drawings directly into a suitable computer, the Macro Expander compiles them, and the Router embeds them in a physical packaging system.

The Graphics Editor allows the designer to edit drawings at a graphics terminal and to print them out. The designer may create a library of shapes (macro bodies) that are generally abstractions of digital logic functions, though some may represent physical parts available from manufacturers. Each macro body is linked by name to a set of drawings, its macro definition. A macro is defined only once but may be used in the drawings any number of times. The designed system is then made up by connecting these macro bodies by lines indicating information flow. A single line in a drawing represents one or more signals (a signal vector) and may be named. Macro bodies have parameters, including parameter signal vectors. Names on signal vectors include a timing notation that allows SCALD to verify automatically (using real or estimated delays of wires and integrated circuits) that stated timing constraints will actually be satisfied

by the digital logic when implemented in the specified physical package.

Figure 5 shows a sample mid-level drawing from the Mark IIA design; it represents several thousands of integrated circuits. The drawing shows the Mark IIA data cache and register file, operand queues, alignment network, arithmetic module, and connections between those elements. This drawing represents the described portion of the machine accurately, in that hardware is automatically built using the drawings as a specification, but it is lacking in detail and requires definitions of its submodules for completeness.

The Macro Expander expands the design to individual integrated circuits by iteratively substituting the appropriate macro definition for each macro body in the drawings. The Macro Expander also verifies that designer-specified timing constraints are satisfied. The Macro Expander is largely technology-independent and is coded in transportable Pascal.

The Router reads an interconnection list produced by the Macro Expander and produces magnetic tapes that permit the design to be implemented by automatic and semiautomatic commercial machines. Extensive maintenance and debugging documentation is produced by the Router, which is also coded in transportable Pascal.

SCALD was used to design the S-1 Mark I and the S-1 Mark IIA.

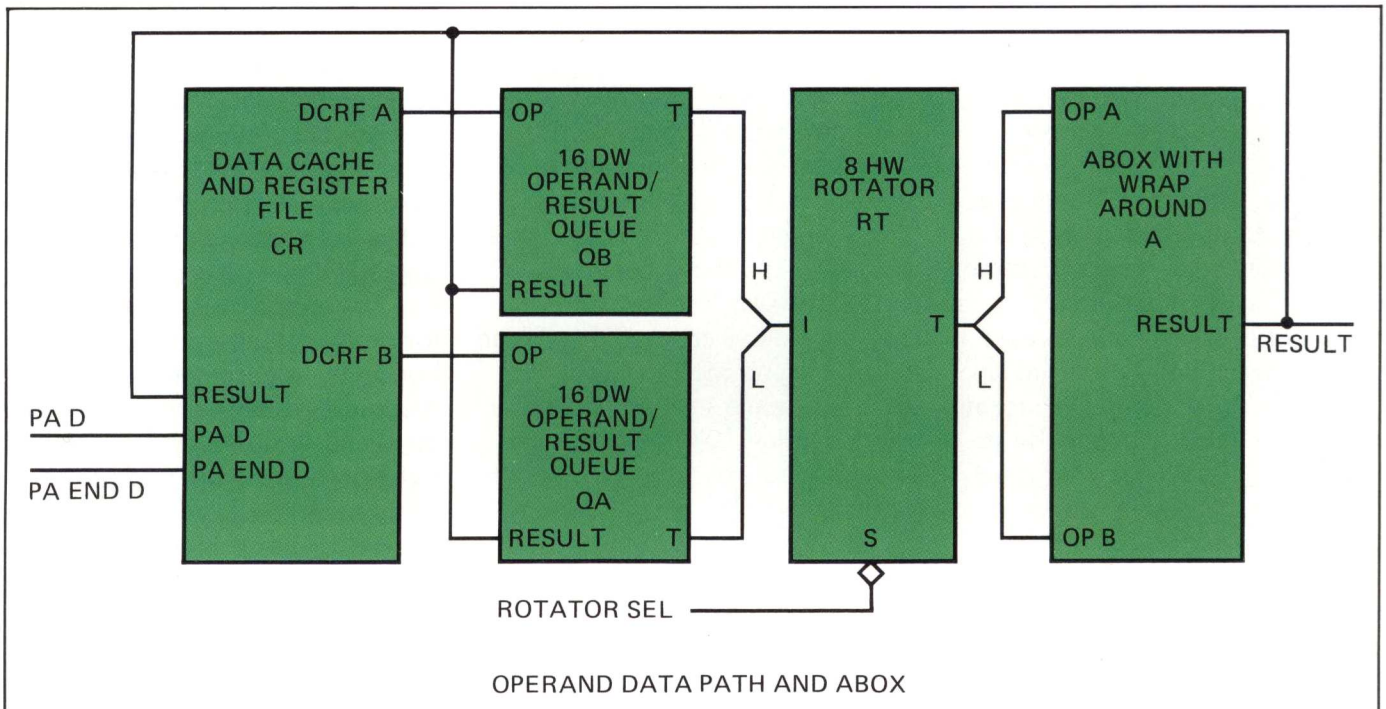


Fig. 5 *A typical SCALD drawing from a middle level of the S-1 Mark IIA Uniprocessor design. This whole drawing defines a single block in a more general diagram at the next higher level. Each of the blocks in the drawing shown here is defined by a more detailed drawing at the next lower level. Thus a hierarchy is established that stretches from the most general abstraction down to the individual components of the computer. Many of these drawings can be used over and over again in the design; they are drawn once and then simply recalled by SCALD as needed. SCALD also generates a wiring list, checks the design for mistakes and timing errors, and produces taped instructions for automatic wiring machines.*

The Mark I design consisted of 211 high-level drawings (drawings used only once in the design) and 144 low-level drawings (drawings used several times). Low-level drawings form an investment in the particular technology chosen for implementation, since they have a high probability of being used again in subsequent designs. In contrast, high-level drawings represent an investment in the particular architecture being implemented and may be reused to recompile that architecture periodically into current, more cost-effective implementations. A total of two man-years was expended in the Mark I design work during an elapsed period of one calendar year.

Structured logic design consists of extending to logic design the essential power of the concepts and tools developed for simplifying the programming task; the savings in human labor expended in designing digital systems are potentially as great as those resulting from the use of compilers. Our experience has shown that SCALD has made the S-1 Mark I and Mark IIA designs more understandable, thus reducing the design efforts, enhancing design correctness, and facilitating generation of final documentation. The designs themselves serve as major portions of the final documentation because they are so readily understood; thus, the need for expensive and usually inaccurate post facto documentation has been greatly reduced. Furthermore, SCALD has increased the mutability of these designs; since macros are inherently isolated, changes in one macro definition usually require minimal changes in other parts of the design. Finally, the imposition of structure on the design and the use of computational resources in the verification task has resulted in designs of an unprecedented level of accuracy.

Summary

S-1 Project effort related to the development of high-performance computing machines is directed toward three major areas: the S-1 Multiprocessor, the S-1 Uniprocessor, and the S-1 Design System. S-1 Multiprocessors are rapidly extensible to very high powers and large memory capacities at uniprocessor cost-effectiveness levels and feature ultrareliable system performance. The S-1 Uniprocessors are general-purpose, emulation-oriented machines that are powerful and highly cost-effective and have advanced maintainability features. The S-1 Design System supports highly automated, general-purpose digital systems design and provides extensive construction and debug support of advanced computer systems.

Acknowledgments

Edward Teller's encouragement and support have been crucial to the initiation and continuation of the S-I Project, one of this Laboratory's most recent initiatives in applied computer science. We gratefully acknowledge this most effectively expressed interest of Teller and that of his like-minded colleagues in this Laboratory, in the Departments of Energy and Defense, and in the Congress.

We also wish to acknowledge our indebtedness to our colleagues in the academic, industrial, and governmental sectors for their many contributions to the work of the S-1 Project.

Key words: CDC 7600; computer-aided logic design; Cray-1; floating-point parallel processing; ICs; integrated circuits; large-scale integration; LSI; multiprocessing; multiprocessor; pipelined processing; S-1 Project; SCALD; uniprocessor; very-large-scale integration; VLSI.

Notes and references

1. Such multiprocessors are described in "Tandem Non-Stop Systems," *Datapro Reports on Minicomputers*, Datapro Research Corporation, Delran, N.J. (1979), and in S. M. Ornstein *et al.*, "Pluribus—A Reliable Multiprocessor," *Proc. AFIPS 1975 Nat. Comp. Conf., Anaheim, 1975* (AFIPS Press, Montvale, N. J., 1975), vol. 44, p. 551.
2. L. M. Censier and P. A. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Trans. Computers* **C-27** (12), 1112 (1978).
3. S-1 Project Staff, *Advanced Digital Processor Technology Base Development for Navy Applications: The S-1 Project*, Lawrence Livermore Laboratory, Rept. UCID 18038 (1978).
4. J. T. Coonen, *Specifications for a Proposed Standard for Floating Point Arithmetic*, University of California, Berkeley, Electronics Research Laboratory Memorandum UCB/ERL M78/72 (1978).
5. F. McMahon, Lawrence Livermore Laboratory, private communication, October 1976.
6. SCALD is described in detail in "SCALD: Structured Computer-Aided Logic Design" and "The SCALD Physical Design Subsystem," written by T. M. McWilliams and L. C. Widdoes and published in *Proc. Ann. Design Automation Conf., 15th, Las Vegas, 1978* (IEEE, ACM, New York, 1978), p. 271.
7. D. Helliwell, *The Stanford University Drawing System*, Stanford Artificial Intelligence Laboratory, Palo Alto, California (1972).