A VLSI Chip Set for a Multiprocessor Workstation—Part I: An RISC Microprocessor with Coprocessor Interface and Support for Symbolic Processing

DAVID D. LEE, MEMBER, IEEE, SHING I. KONG, MARK D. HILL, MEMBER, IEEE, GEORGE S. TAYLOR, DAVID A. HODGES, FELLOW, IEEE, RANDY H. KATZ, SENIOR MEMBER, IEEE, AND DAVID A. PATTERSON, SENIOR MEMBER, IEEE

Abstract **—This two-part paper describes two key components used in building a 40–70 MIPS multiprocessor workstation. In the first part, VLSI implementation of the central processing unit (CPU) chip, based on reduced instruction set computer (RISC) architecture and with support for LISP is described. The 1.3-cm^2 CPU chip uses a direct-mapped 512-byte on-chip instruction cache, and 138 40-bit registers organized in eight overlapping windows to achieve 10 MIPS per processor peak performance with a 10-MHz, four-phase clock.**

The second part of the paper [1] describes the memory management unit and cache controller (MMU/CC) chip. System-level design issues such as multiprocessor cache coherency and synchronization among chip sets are also considered in the second part. Both chips are implemented in a 1.6-μm double-layer-metal CMOS technology, and are being used in a multiprocessor workstation (SPUR) successfully executing a UNIX-like network-based operating system called Sprite as well as many applications including LISP programs.

I. Introduction

SPUR (Symbolic Processing Using RISC's) is a multiprocessor workstation developed at the University of California at Berkeley as a testbed for research on parallel processing, particularly in LISP [2]. The SPUR workstation, shown in Fig. 1, can have 6 to 12 identical processors, each of which consists of a 128-kbyte cache, a central processing unit (CPU), a floating-point coprocessor, and a memory management unit and cache control (MMU/CC) that assures cache coherency among multiple processors. A photograph of a fully populated SPUR processor board is shown in Fig. 2. This paper describes the VLSI implementation on the CPU chip, a 32-bit RISC multiprocessor.

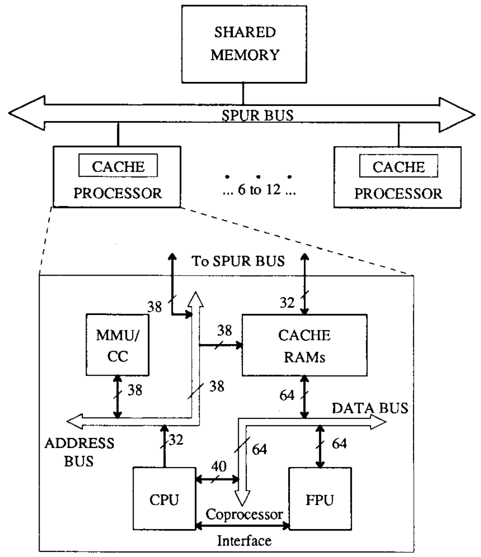


Fig. 1. SPUR multiprocessor workstation.

The SPUR CPU supports a multilevel cache scheme that includes a prefetching on-chip instruction cache, a coprocessor interface, and support for fast execution of LISP programs through a tagged 40-bit architecture. The coprocessor interface uses 27 pins to implement a low-overhead interface supporting concurrent CPU and FPU operations.

Manuscript received April 10, 1989; revised July 6, 1989. This work was supported by DARPA under Contract 482427-25840, California MICRO, Texas Instruments, National Semiconductor, Cypress Semiconductor, Tektronix, and Hewlett-Packard.

D. D. Lee was with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA. He is now with Xerox Palo Alto Research Center, Palo Alto, CA 94304.

S. I. Kong was with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA. He is now with Sun Microsystems, Mountain View, CA 94043.

M. D. Hill was with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA. He is now with the Computer Sciences Department, University of Wisconsin, Madison, WI 53706.

G. S. Taylor was with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA. He is now with MIPS Computer Systems, Sunnyvale, CA 94086.

D. A. Hodges, R. H. Katz, and D. A. Patterson are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720

IEEE Log Number 8930831.

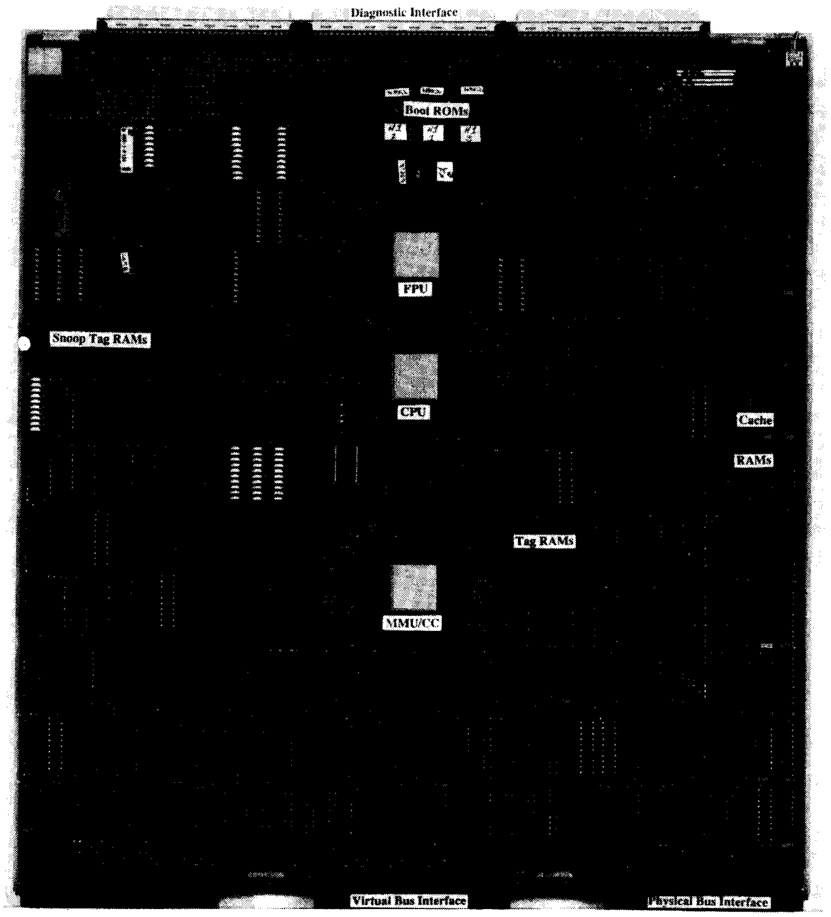


Fig. 2. SPUR processor board.

TABLE I
CHIP STATISTICS

Number of Transistors	115,214
Number of PLA's	13
Die Size	11.5mm x 11.5mm
Package	208-pin PGA (40 pins for power supply)
Process	Double-Metal 1.6μm N-Well CMOS
Operating Frequency	10MHz, with 5V supply and at 25°C
Power Dissipation	0.8W at 10MHz with 5V Supply

The chip, implemented in 1.6-μm double-metal CMOS technology, contains 115K transistors. The chip statistics are summarized in Table I, and a chip photomicrograph is shown in Fig. 3. An on-chip clock generator, based on a charge-pump phase-locked loop with tapped delay line, provides an accurate phase relationship with the board clock and also with clock phases of the other chips [3]. Nominal operating frequency with a four-phase nonoverlapping clock (18-ns nominal per phase and 7-ns nonoverlap time between phases) is 10 MHz (12.5 MHz max, with

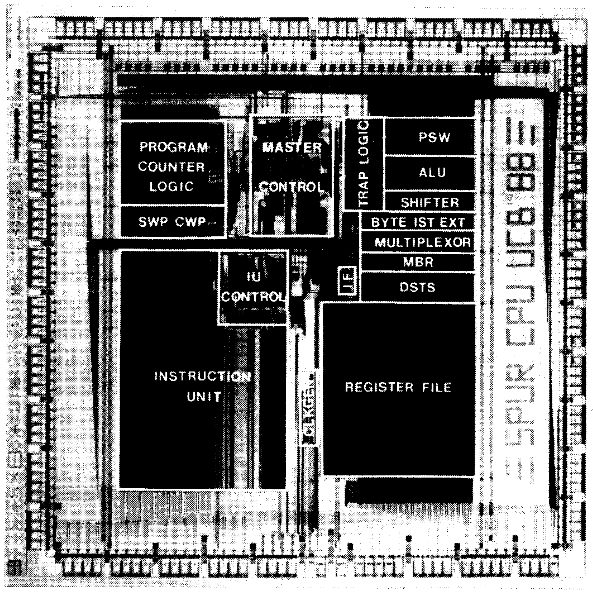


Fig. 3. Chip photomicrograph.

a 5-V supply at 25°C). The dual-ported register file and the on-chip instruction cache, operating at different points in each cycle, require four-phase nonoverlapping clocking. A SPUR uniprocessor running LISP programs (Gabriel benchmarks) at 10 MHz can provide 2× performance improvement on the average over the Symbolics 3600 or VAX 8650, according to simulation [4]. A multiprocessor SPUR workstation with 6 to 12 processors is predicted to yield a sustained throughput of 40 to 70 MIPS, respectively. A prototype multiprocessor workstation configured with three processors has been developed. It runs a UNIX-like network-based operating system called Sprite as well as many applications.

The organization of the paper is as follows. Section II gives an overview of the CPU architecture and execution pipeline. Section III focuses on the hardware required to implement various features of the SPUR CPU architecture. Section IV describes the design, verification, and testing methodologies of the full-custom SPUR CPU chip. Finally, the summary and conclusion are given in Section V.

II. An Overview of the SPUR CPU Architecture

The SPUR CPU is a third-generation RISC microprocessor developed at the University of California at Berkeley. It is specifically designed to be used in the SPUR multiprocessor workstation. The architecture is similar to those of previous RISC projects at U.C. Berkeley [5], [6], but significant new features have been added. These include coprocessor interface to support floating-point computation, an efficient interface to the MMU/CC, and run-time hardware tag checking for fast execution of LISP programs. The instruction set of the SPUR CPU is carefully chosen such that an efficient implementation of the single-cycle execution of all instructions is possible.

Like previous RISC processors, the SPUR CPU is a load–store machine. Memory is accessed only through load and store instructions. All other instructions are register-to-register or immediate-to-register oriented. There are four generic instruction types: register-to-register, store, compare-and-branch, and call-jump. Load and return instructions are special cases of register-to-register in which $R_{s1}+R_{s2}$ or $R_{s1}+$immediate is used as an effective address. The R_d field specifies the register to be loaded for the load instruction type and is not used for the return instruction type. All instructions (40 integer and 20 floating point) are 32 bits wide and use fixed formats. The seven instruction formats are shown in Fig. 4. For simplicity of decoder, opcode and register specifiers are in the same positions in all formats. The three-register format (RRR) is used for load, register-to-register operations, special register operations, and coprocessor operations. The two-register and one-immediate (RRI) format is used for load and register-to-register operations. Compare-and-branch instructions have three slightly different formats depending on the field specifying the condition.

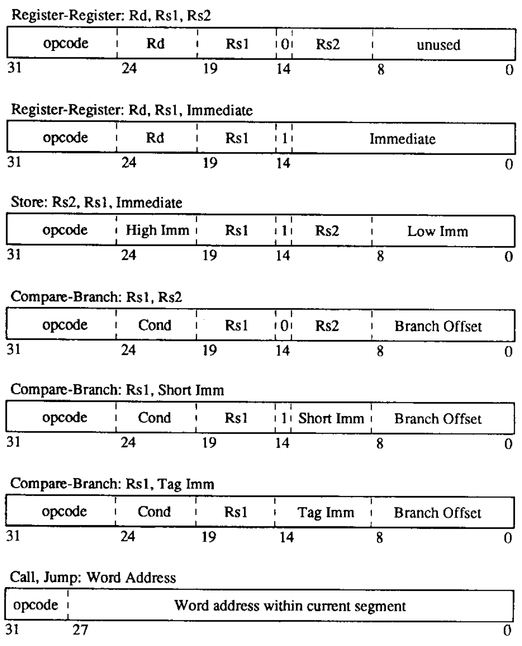

Register-Register: Rd, Rs1, Rs2
opcode | Rd | Rs1 | 0 | Rs2 | unused
31 24 19 14 8 0
Register-Register: Rd, Rs1, Immediate
opcode | Rd | Rs1 | 1 | Immediate
31 24 19 14 0
Store: Rs2, Rs1, Immediate
opcode | High Imm | Rs1 | 1 | Rs2 | Low Imm
31 24 19 14 8 0
Compare-Branch: Rs1, Rs2
opcode | Cond | Rs1 | 0 | Rs2 | Branch Offset
31 24 19 14 8 0
Compare-Branch: Rs1, Short Imm
opcode | Cond | Rs1 | 1 | Short Imm | Branch Offset
31 24 19 14 8 0
Compare-Branch: Rs1, Tag Imm
opcode | Cond | Rs1 | Tag Imm | Branch Offset
31 24 19 14 8 0
Call, Jump: Word Address
opcode | Word address within current segment
31 27 0

Fig. 4. SPUR instruction formats.

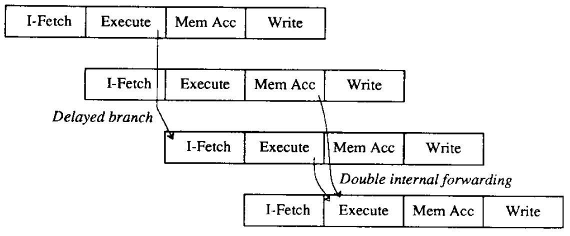


Fig. 5. SPUR CPU pipeline.

The CPU registers are organized in eight overlapped windows (128 registers) and ten global registers accessible from any window (total 32 registers visible from one window). The overlapped window scheme considerably reduces the register save and restore overheads between procedure calls. Unlike previous Berkeley designs, the registers are 40-bit registers with 32 bits for data and an 8-bit tag used for run-time type checking and garbage collection. The 8-bit tag consists of a 6-bit object's type tag and a 2-bit generation number. LISP is supported with three types of hardware tag checking with traps to a software trap handler: data type checking for general computations, pointer type checking for list operations, and generation number checking for garbage collection based on the generation scavenging algorithm [7].

The on-chip instruction cache provides the effect of an extra memory port, allowing simultaneous data memory reference and instruction fetch by the execution unit (EU). This leads to a four-stage pipeline (Fig. 5) that eliminates pipeline stalling whenever a load instruction is executed. Consequently, the CPU can issue and complete one instruction per cycle (peak performance rate of 10 MIPS per processor) as long as there are no instruction or external data cache misses. Branch conflict in the pipeline is resolved by a single-cycle delayed branch with one instruction in the delayed slot. Data conflicts are resolved by hardware internal forwarding logic.

To facilitate high-precision floating-point computations and other possible coprocessing capabilities, the SPUR CPU incorporates a parallel interface to coprocessors. The floating-point coprocessor interface (27 pins) implemented in the current version of the CPU chip supports concurrent CPU and FPU operations. The FPU tracks CPU instructions issued by the instruction cache in the CPU via 22 pins carrying opcode and register specifiers. The CPU sends two control signals to the FPU, and the 3-bit FPU status is sent to the CPU. The CPU treats all FPU instructions as illegal instructions when the FPU is disabled. When the FPU is enabled, all FPU instructions except FPU load and store are treated by the CPU as NO_OP. For FPU load and store, the CPU computes the effective memory address and the FPU reads and writes the data directly from the external cache.

In the SPUR instruction set, a number of special load (seven) and store (three) instructions are dedicated to cache control and virtual memory management. Although these instructions look almost identical to the CPU, appropriate cache operations are provided by the CPU to the external MMU/CC through the MMU/CC interface. The interface consists of a 4-bit cache opcode, two bits indicating the mode of operations (user versus kernel and physical versus virtual), and nine other status bits of both the CPU and the MMU/CC.

The unusual conditions that the CPU may face at run time can be divided into four groups. Unusual conditions detected inside the CPU are called CPU exceptions: integer overflow, tag checking, window overflow and underflow, and so on. Unusual conditions caused by the FPU are called floating-point exceptions. All other unusual conditions occurring outside the CPU are called faults and interrupts. Faults occur in response to the execution of an instruction, while interrupts are asynchronous events that come from outside the processor (e.g., an I/O interrupt). The CPU responds to exceptions, faults, and interrupts by taking a vectored trap. There is a priority ordering for cases when more than one unusual condition occurs at the same time. Traps can be disabled or enabled selectively by controlling the 8 bits in both kernel and user processor status words (KPSW and UPSW).

III. Hardware Implementation of the SPUR CPU

The major functional blocks are shown in Fig. 6 and outlined in the chip photomicrograph (Fig. 3). They are the execution unit (EU) and the instruction unit (IU). The EU is further divided into the upper data path, the lower data path, and the control. The 30-bit upper data path contains pipelined program counters and special registers. It is used for instruction address calculations and special register references. The 40-bit lower data path is for general computation on the tagged registers.

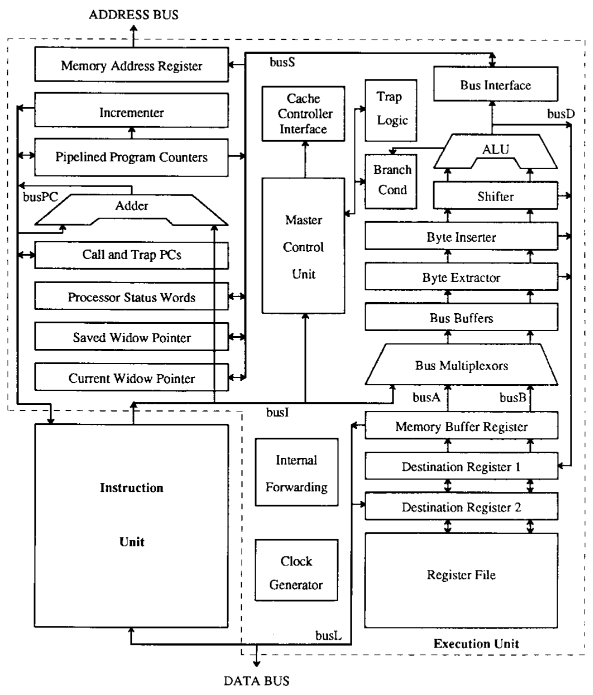


Fig. 6. SPUR CPU block diagram.

A. The Register File

The SPUR CPU has a total of 138 general-purpose registers organized in eight overlapped windows and ten global registers. Thirty-two registers are visible to the compiler at any one time: ten globals, ten locals, six overlapped with caller window, and six overlapped with callee window. Each register is 40 bits wide having a 6-bit tag, 2 bits for generation number, and 32 bits for data. The six-transistor (6T) CMOS SRAM cell is used for the register cell (single-ended access for two reads per cycle and differential access for a write) [8]. The layout of a SRAM cell is constrained by the pitches of the data-path bit slice and the register decoders (two decoders per register). The result is a large but fast SRAM cell that does not require a sense amplifier.

The SPUR CPU architecture is register oriented and requires two reads and one write per cycle. The register read and write operations are time multiplexed and pipelined to minimize the critical path. Bit lines are decoded and precharged in the same phase, and the register array is accessed in the following phase by driving the

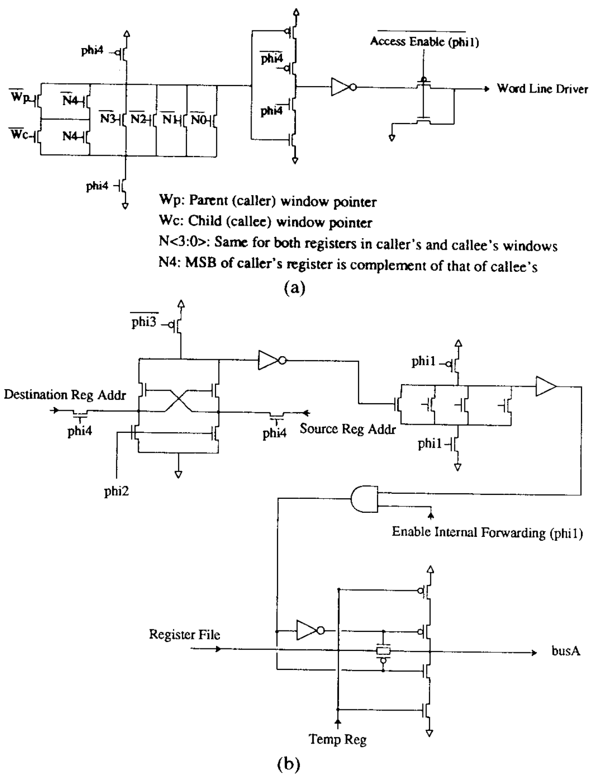

phi4
Wp
Wc
N4
N3
N2
N1
N0
Access Enable (phi1)
Word Line Driver
Wp: Parent (caller) window pointer
Wc: Child (callee) window pointer
N<3:0>: Same for both registers in caller's and callee's windows
N4: MSB of caller's register is complement of that of callee's
(a)
phi3
phi1
Destination Reg Addr
phi4
Source Reg Addr
phi2
Enable Internal Forwarding (phi1)
Register File
busA
Temp Reg
(b)

Fig. 7. (a) Overlapped window register decoder. (b) Internal forwarding logic.

word line. The access time of the register-file read is the critical path of the chip. It is measured to be under 14 ns with a 5-V supply at 25°C. For registers in the overlapped window, a special decoder shown in Fig. 7(a) is used to map two different register addresses (one from the caller's window and the other from the callee's window) to one register [5].

In the pipelined execution of the instruction stream, data interdependencies among instructions in the pipeline may arise. In the SPUR CPU, these interdependencies are detected and resolved by the hardware which forwards the results from preceding instructions to the following instructions (internal forwarding) before being written back to the register file, as indicated by the arrows in Fig. 5. In the case of a four-stage pipeline like the SPUR CPU, the data interdependencies may exist among three consecutive instructions since the write-back stage of the pipeline is delayed by two cycles after the execution stage. The result available from each instruction's execution stage, therefore, needs to be stored in temporary registers for two cycles and then forwarded to the following instructions if necessary. When both operands are registers, each register address is compared to the destination register address of the two preceding instructions. This may result in double internal forwarding, in that both operands are results of two preceding instructions and hence supplied from the temporary registers.

The hardware internal forwarding logic is in the critical path of the register-file access, and it must be implemented without slowing down the cycle time. Like decoding and accessing the register array, it is also pipelined. Address comparisons are done in parallel with the decoding of the register file, and internal forwardings are made if necessary while the register file is accessed. Four address comparisons are necessary to detect all possible data dependencies. The address comparator must be fast to keep the cycle time short, and it must be compact to fit in the area between register decoders and temporary registers, as seen in Fig. 3 (block IF). Bit-wise comparison is done using a dynamic XOR, shown in Fig. 7(b), and then the outputs are fed into the domino circuit for an address match. Since this XOR does not require complementary inputs, the routing and area required are significantly reduced. A special multiplexor, shown in Fig. 7(b), is used to minimize the signal delay through the internal forwarding logic that lies between the register file and the functional unit. If internal forwarding is necessary, the bus from the register file is disconnected by the transmission gate, and the bus to the functional unit is driven by the temporary register. The access time of register-file reading (14 ns) includes the delay through the internal forwarding logic.

B. The Instruction Unit

The SPUR IU contains 128 instructions (512 byte), is direct-mapped (16 blocks of eight instructions per block), and is addressed by virtual addresses. A novel feature of the SPUR instruction cache is a valid bit associated with each instruction word in the cache so that any subset of instructions within a block may be valid. The SPUR IU uses this flexibility to reduce demand miss time by loading only the fetched instruction rather than the entire block and to permit instruction prefetching to load the rest of a block in parallel with subsequent instruction fetches [9], [10]. If subsequent prefetches are successful, the miss penalty is just two cycles (loading the first instruction) for the entire block containing the missed instruction.

The IU can operate in three different modes: 1) disabled, 2) enabled without prefetching, and 3) enabled with prefetching, controlled by two bits in the KPSW. In the disabled mode, the IU fetches every instruction requested by the EU from the external cache. The disabled mode is useful for initial chip testing and for allowing chips with stuck-at-type errors in the cache or tag array to function correctly, albeit more slowly. In enabled-without-prefetching mode, the IU will cache instructions upon demand misses but will not initiate any prefetches.

The normal mode is the enabled with prefetching. After the missed instruction is cached, prefetches are made to subsequent words within the block until another demand miss occurs or prefetch is blocked by the EU's external data access. These prefetches are "free," as they never interfere with external cache accesses, such as instruction fetch and external data reference, by the EU because prefetch has the lowest priority. If prefetch causes an external cache miss, the cache controller simply ignores the request.

The instruction unit is controlled by two finite-state machines: one controls the fetching and the other controls the prefetching of instructions. Two finite state machines and other random control logic are partitioned into the total of six PLA's considering the timing constraints. The same 6T SRAM cell used in the register file is used in the IU, except it is accessed differentially for both read and write. The data portion of the cache is an array of 128 words. Associated with each entry are a 32-bit instruction word and a valid bit. The tags are stored in a separate array (16 24-bit words) whose access time is significantly faster than that of the data array. This allows the tag comparison to be done while the instruction is being read out from the data array. Bit-wise comparison using an XOR gate is used for tag comparison and is followed by dynamic logic to determine a hit. The effective access time of the instruction cache including hit logic is under 12 ns (5-V supply at 25°C) without using a sense amplifier.

C. The Data Path

The data path is divided into two parts: the upper data path for program counter logic and special registers, and the lower data path for general computations on tagged registers. Functional units in the lower data path include a byte extractor, a byte inserter, a simple shifter that shifts up to 3 bits, and an ALU. The ALU provides XOR, OR, AND, ADD, and SUBTRACT operations and comparison for two 32-bit operands. The upper data path consists of a number of program counters to hold instruction addresses in the pipeline, an address incrementer and adder, and special registers such as window pointers and processor status words. All registers and counters are made of pseudo-static latches, such that each register is refreshed once per cycle. This is necessary because an indefinite pipeline stall is possible due to a long external cache miss.

In the SPUR CPU, compare-and-branch instructions are executed in one cycle with one delay slot. A separate adder in the upper data path calculates the target addresses for all the compare-and-branch instructions while the ALU is used for the comparison. Two different adder designs are employed. The 32-bit ALU uses four 8-bit carry lookahead adders implemented in domino logic, and evaluates the carry within 11 ns (5-V supply at 25°C). The 30-bit address adder is more compact because it uses a Manchester carry chain which has a carry propagation delay of 13.5 ns.

The upper 8-bit slice of the lower data path is for tag-related operations. Operations on the tag and the data are logically independent, that is, no information moves between the two parts by carry propagation or any other implicit mechanism. For operations, the 6-bit tag type is checked in parallel with the data operation. If there is a tag mismatch and the tag-trap enable bit is set in the UPSW, the CPU traps to the software. Generation tag checking (2 most significant bits) is done when a special store instruction (ST_40 R_{S1}, R_{S2},Immediate) is executed. A generation tag exception may occur if the object (R_{S2}) with a higher (younger) generation number is stored into the object (R_{S1}) with a lower generation number [7]. The read_tag and write_tag instructions move a tag to and from the data portion of a register using the byte-extractor and the byte-inserter, respectively, so that any arithmetic or logical operations may be performed on it.

TABLE II
CRITICAL PATH TIMING

phase	operation	critical path†
phi1	Register file - read	14.0 nsec
phi2	Instruction Cache - fetch	12.0 nsec
phi3	ALU - 32b carry propagation	11.0 nsec
phi4	Address adder - 30b carry propagation	13.5 nsec

† with 5V supply and at 25 °C

To reduce the chip area and improve the circuit speed, domino logic [11] is heavily used in the design. Potential charge-sharing problems are prevented by careful layout of the critical nodes (parasitic capacitance on critical node is intentionally made much higher than adjacent nodes possibly sharing charge). The SPUR CPU has seven major buses to provide communications both externally and internally. Some of these buses have high capacitive loadings, and hence precharging is used to improve the speed of data flow through the highly capacitive buses. The high-capacitance bus is precharged to high before being used and discharged conditionally through a strong NMOS pull-down network. This not only reduces the signal delay through the bus but also minimizes the chip area required for a strong, large driver. Some logic function may be included in the pull-down network as well, further saving the chip area. The four-phase clock made it easier to implement these dynamic circuits since precharging can be hidden and well separated from the evaluation phase. Precharging current can also be somewhat distributed over several phases, reducing the current spikes. Critical paths of the data path, register file, and instruction cache are summarized in Table II.

D. The Control

Four-phase clocking and a uniform four-stage pipeline for all SPUR integer instructions make the control section of the CPU relatively simple. The SPUR CPU uses internal instructions to handle pipeline interrupts, rather than requiring complex sequences for those exceptions. These internal instructions are issued by the control unit when a cache miss (miss) or trap occurs (trap_call and read_pc). The use of these internal instructions further simplifies the control design because they are handled by the EU as if they are regular user instructions.

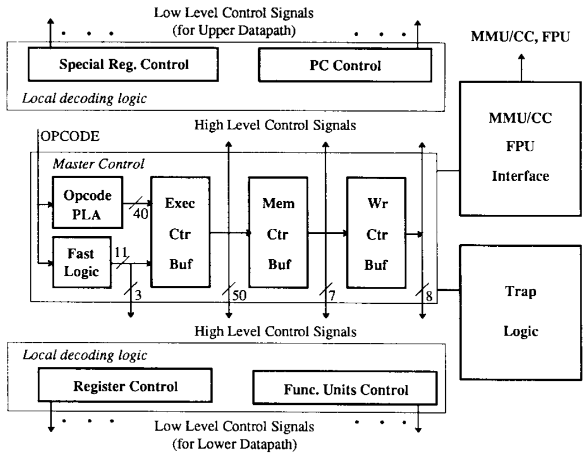


Fig. 8. Block diagram of master control.

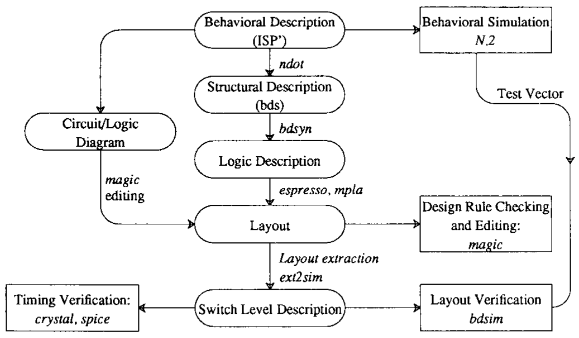
The control can be divided into three parts: master control, trap logic, and the interface to the MMU/CC. The latter two are separated out from the master control to simplify the control design. Trap logic detects all unusual conditions during the pipelined execution of an instruction. All traps are taken during an instruction's third pipeline stage, and hence only one instruction can cause a trap in any cycle. The MMU/CC interface logic generates cache opcodes according to the current instruction and the status of the CPU. It also buffers signals to and from the MMU/CC.

A block diagram of the master control is shown in Fig. 8. A centralized master control unit controls the processor sequencing and decodes the opcode into high-level control signals. Local random logic blocks then decode the high-level signals into low-level signals. They also provide buffering of the low-level signal according to the loading requirement. All signals controlling the data path are individually optimized to have equal delays relative to the clock edges. The separation of master control and local decoding/buffering significantly reduces the amount of routing between those two blocks, particularly in CMOS design where complementary signals are required in controlling the data path.

Most of the control logic in the SPUR CPU is implemented in static PLA's. The largest PLA is the one that decodes the opcode. It has 69 product terms with 40 outputs. The simulated propagation delay through this PLA is about 15 ns, well below the required timing of two phases or 50 ns. All PLA outputs are evaluated once per cycle and need to be held in registers until the next cycle.

IV. Design, Verification, and Testing Methodology

Methodologies employed in the SPUR CPU design have been influenced by the following two themes of the SPUR project: 1) an overall system-wide rather than local optimization, and 2) designing a chip for a working system rather than an experimental prototype chip. Consequently, methodologies became very important since the chip being designed must meet all the functional requirements set for the system design as well as performance goals.



Behavioral Description (ISP')
Behavioral Simulation N.2
ndot
Structural Description (bds)
Test Vector
Circuit/Logic Diagram
bdsyn
Logic Description
magic editing
espresso, mpla
Layout
Design Rule Checking and Editing: magic
Layout extraction ext2sim
Timing Verification: crystal, spice
Switch Level Description
Layout Verification bdsim

Fig. 9. Design methodology.

A. Design Methodology

The design strategy incorporated both top-down and bottom-up approaches. The top-down flow was as follows: architecture definition, instruction set design, microarchitecture design, and a detailed functional/behavioral description of the hardware. The bottom-up flow was the circuit design of basic components, the layout of basic cells, the assembly of major blocks using those cells, and the global placement and interconnections. Both approaches were taken in parallel from the beginning, in order to achieve the highest performance for given technology and system design goals. For instance, many microarchitecture decisions were made after the feasibility of a given hardware resource was carefully considered. Division of design tasks followed the same hierarchical boundaries of design abstractions: architecture and instruction set design, microarchitecture design, and VLSI implementation. One- or two-person groups were formed to take responsibility for each design level. Close interaction among different groups was necessary to make clean interfaces among themselves and design specifications.

Most of the CAD tools used in designing the SPUR CPU chip were developed at Berkeley, except those for the behavioral level design. The detailed design started by describing the behavior of the chip and its interactions with other components within the system. The functional behavior was written in ISP', a hardware description language, and simulated using the N.2 simulator [12]. Most parts of the control design were done using a set of CAD tools that automatically synthesizes the behavioral description of combinational logic into a PLA [13], [14]. Other parts of the control logic (sequential) and data paths were designed manually but aided by another set of tools. These two approaches are diagrammed in Fig. 9. For the automated synthesis path, only those parts of the hardware description containing combinational logic can be synthesized. For the manual part, logic and circuit design were done first for each block followed by layout. Layout was done using an interactive layout editor, Magic [15], with background design rule checking and hierarchical extraction. The extracted layout, a switch-level description of the chip, was simulated using *bdsim*, a switch-level logic simulator [13].

Timing analysis was done before layout, to make early trade-offs among the many alternatives, and again after layout, to perform exact timing analysis with all parasitics correctly annotated. To estimate critical paths more accurately, and thus to determine the minimum cycle time, a test chip containing a register file with internal forwarding was fabricated and tested [16]. The measured critical path (register-file read) was below 18 ns, and this encouraged us to set the cycle time goal at 100 ns.

B. Verification Methodology

The verification methodology was constructed following a bottom-up approach. As each individual module was designed, a small set of handwritten test vectors was used to perform switch-level simulations on the extracted layout. Once individual modules were verified, they were connected and simulated together until the integration reached the highest level division: execution unit and instruction unit. Test vectors up to this point were small and easy to generate by hand, since the test sequences required to verify operations on these units separately were relatively simple. After all major blocks were integrated, the verification effort was directed at both functional and switch levels.

Functional simulations were performed not only on each major component, to verify its internal functions, but also on the external system level, to verify interactions among major chip sets. The diagnostics for the functional simulation were coded in SPUR instructions, and an instruction-level simulator called *Barb* was written to debug the diagnostics. The diagnostics were intended to be stored in a start-up ROM on the processor board. The N.2 system provided simulated memories that could be used to model ROM or other types of memory. Therefore the diagnostics were assembled and loaded into the simulated memory. When the N.2 simulation was started, it was forced to go through a series of start-up sequences, making the CPU begin fetching instructions from the simulated ROM containing the diagnostics. The diagnostics were then executed to completion or failure. The same ROM image was used to program EPROM's to be used for on-board testing of the chip.

Running extensive simulations on the hardware description verified many design ideas and functionalities, but it was still necessary to extract and simulate the layout of the entire chip. The extracted description is almost guaranteed to accurately model the real chip. However, developing tests and examining their results for a complete switch-level simulation would be very difficult. To minimize the required work, the functional simulation was used to drive the switch-level simulation while automatically verifying that the two match at every clock cycle. Fortunately, the N.2 simulator provides a "tracing" capability that logs all changes to a specified set of signals into a file. By tracing all inputs and outputs of an N.2 module, it is possible to obtain a set of switch-level test vectors automatically. These vectors along with expected results on output nodes are fed into the switch-level simulation. The switch-level simulator *bdsim* sets the input nodes according to the timing and vectors specified and verifies the output nodes with the expected results. Any unusual condition is recorded so as to be used in debugging.

TABLE III
DIAGNOSTICS

diagnostics	test vector length (cycles)
CPU functions	13,113 (24%)
MMU/CC interface	16,356 (29%)
FPU interface	1,543 (3%)
Lisp tags and traps	8,675 (16%)
Boot-up diagnostics	15,829 (28%)

A problem may arise because functional simulation and switch-level simulation may show different results under unusual states, such as unknown and initial states. For example, the functional simulator initializes all nodes to zero, while all nodes are initially set to unknown in the switch-level simulation. When the chip is tested, neither of these initial conditions is correct. To alleviate this problem, all internal states are initialized explicitly in the simulations.

In order to have a working system rather than merely a prototype chip, all aspects of the design had to be verified, especially the interfaces to external chips. Table III summarizes the diagnostic vectors simulated in both functional and switch-level simulations. These vectors are mainly for the CPU chip. Additional system-level verification was done using extensive functional simulation, and is further discussed in the companion paper [1].

C. Testing Methodology

Several features were incorporated into the SPUR CPU chip to increase its testability. Passive scan registers are attached to all major buses to increase observability. Any signal put on these buses can be scanned out for examination. All major blocks are connected and communicate through these buses, so the diagnostics capability is greatly improved. Many signals, like state bits of finite state machines in IU and the LSB's of the instruction address bus (busPC), are also routed out to pins to help determine the status at any time. The CPU sends out an instruction every cycle to the FPU (via busI). This enables us to monitor the instruction being executed including internal instructions. The IU and the EU can be physically separated by setting certain *diagnostic pins*. Using these features, instructions can be delivered directly to the EU, in case the instruction unit is not functional, by monitoring the instruction address (busPC⟨10:2⟩) available on pins.

Initial testing was done on a special board made for the SPUR CPU chip. The Tektronix DAS 9100 system was connected to the board and controlled from a SUN workstation. The same vectors used in the switch-level simulations were converted into test vectors. For short testing, test vectors were downloaded to the DAS and executed. A special setup was necessary for long-cycle testing, since the DAS can hold at most 256 cycles of test vectors. Long vectors were divided into several parts to fit in the DAS capacity. The division was made at an instruction accessing memory (external cache), and the CPU was deliberately made to stall on cache miss by controlling the MMU/CC interface pin (cache busy) while the next portion of the vector was being downloaded. All signals acquired during the testing were transferred back to the SUN workstation for a cycle-by-cycle verification against the expected result. Most of the CPU functionalities were tested using the special test setup rather than the real processor board. After the debugging was done, the CPU chip was put on a SPUR processor board to test interactions with other components on the board, especially with the MMU/CC.

D. Results

First-pass silicon had a few bugs, including circuit design, layout, and timing errors, but it worked well enough to be used for initial debugging of the processor board. The layout errors discovered were well and substrate contacts misplaced onto signals rather than power-supply lines. These effectively shorted the signal to either the ground or the power line, resulting in a stuck-at type fault. This fault was not detected in switch-level simulation because the well was not explicitly drawn in the layout (Magic layout editor [15]), hence the layout extraction did not properly handle well or substrate contacts. Some of these errors were corrected by isolating the misplaced contacts from the power supply using a laser restructuring technique provided by the Information Science Institute (ISI). Either first-level or second-level metal can be disconnected by using a laser shot through the passivation layer. The second-level (topmost) metal lines with a width of 3 μm were cut successfully without affecting nearby structures. Other problems found were timing errors and glitches on signals controlling the dynamic circuits. The glitch was caused by excessive ringing on clock lines. The long clock lines (10 mm) had parasitic inductance and capacitance large enough to cause a substantial ringing, which triggered a hazardous glitch.

Several electrical-rule checks were performed to avoid repeating the same errors for the second pass. However, there was still another layout error discovered after the second fabrication. A portion of metal wire was missing, leading to a disconnected signal. A focused-ion-beam (FIB) IC development system, provided by the Seiko instrument company, was used to fix the problem. Two holes were drilled on separated wires through the passivation layer to reach the metal lines, using an ion beam, and connected using FIB chemical vapor deposition (CVD) metal film deposition between the two points. The collection of revised and repaired chips was found to be functional. These chips are currently used in the multiprocessor workstation configured with three processors, successfully executing the operating system (Sprite) as well as many applications including LISP programs. The nominal operating frequency of the chip is 10 MHz, while the maximum operating frequency is 12.5 MHz (80-ns cycle time), with a 5-V supply at 25°C.

V. Summary

The SPUR CPU is a single-chip RISC microprocessor designed for a multiprocessor workstation. It supports a multilevel cache scheme including a prefetching on-chip instruction cache, a coprocessor interface, and support for the fast execution of LISP through a tagged 40-bit architecture. In order to build a working computer system based on the SPUR CPU chip, reliable and efficient methodologies were necessary throughout the design. The chip, fabricated in a 1.6-μm double-metal CMOS process, works well in a multiprocessor system prototype, meeting both functional and performance goals set at the initial stage of the design. It runs at 10 MHz consistently for all programs and dissipates less than 0.8 W (5-V supply) of power.

Acknowledgment

The authors wish to thank all members of the SPUR project who made it an exciting project to work on, and acknowledge the technical contributions of R. Duncombe, W. Koh, K. Lutz, and J. Mak. Wafer fabrication was by Hewlett-Packard via MOSIS.

References

[1] D.-K. Jeong *et al.*, "A VLSI chip set for a multiprocessor workstation—Part II: A memory management unit and cache controller," this issue, pp. 1699–1707.

[2] M. D. Hill *et al.*, "Design decisions in SPUR," *IEEE Computer*, vol. 19, no. 10, pp. 8–24, Nov. 1986.

[3] D. K. Jeong *et al.*, "Design of PLL-based clock generation circuits," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 2, pp. 255–261, Apr. 1987.

[4] G. S. Taylor *et al.*, "Evaluation of the SPUR Lisp architecture," in *Proc. 13th Int. Symp. Computer Architecture* (Tokyo, Japan), June 1986.

[5] M. G. H. Katevenis, "Reduced instruction set computer architectures for VLSI," Univ. of Calif., Berkeley, Tech. Rep. UCB/CS Division 83/141, Oct. 1983.

[6] J. M. Pendleton *et al.*, "A 32-bit microprocessor for Smalltalk," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 5, pp. 741–749, Oct. 1986.

[7] D. Ungar, "Generation scavenging: A non-disruptive high performance storage reclamation algorithm," in *ACM Software Engineering Notes/SIGPLAN Notices Software Engineering Symp. Practical Software Development Environments* (Pittsburgh, PA), Apr. 1984.

[8] R. W. Sherburne, Jr., M. G. H. Katevenis, D. A. Patterson, and C. H. Sequin, "A 32b NMOS microprocessor with a large register file," *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 682–689, Oct. 1984.

[9] J. R. Goodman, "Using cache memory to reduce processor memory traffic," in *Proc. 10th Int. Symp. Computer Architecture* (Stockholm, Sweden), June 1983, pp. 124–131.

[10] M. D. Hill and A. J. Smith, "Experimental evaluation of on-chip microprocessor cache memories," in *Proc. 11th Int. Symp. Computer Architecture* (Ann Arbor, MI), June 1984, pp. 158–166.

[11] R. H. Krambeck, C. M. Lee, and H. S. Law, "High-speed compact circuits with CMOS," *IEEE J. Solid-State Circuits*, vol. SC-17, pp. 614–619, June 1982.

[12] *N.2 Simulator User's Manual*, ENDOT, Inc., Cleveland, OH, 1985.

[13] R. B. Segal, "BDSYN: Logic description translator, BDSIM: Switch-level simulator," Electron. Res. Lab., Univ. of Calif., Berkeley, Memo. M87/33, May 1987.

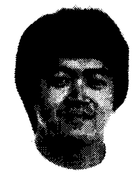
[14] OCT Tools Distribution 2.1, Electron. Res. Lab., Univ. of Calif., Berkeley, Mar. 1988.

[15] J. K. Ousterhout *et al.*, "The Magic VLSI layout system," *IEEE Design & Test Computers*, vol. 2, pp. 19–30, Feb. 1985.

[16] D. Lee, "Data path design considerations for a high performance VLSI multiprocessor," Univ. of Calif., Berkeley, Tech. Rep. UCB/CS Division 87/318, Nov. 1986.

David D. Lee (S'85–M'88) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1983, 1986, and 1989, respectively.

Before joining the SPUR project at Berkeley, he worked at IBM General Technology Division, Poughkeepsie, NY (1983–1984). In 1989 he joined Xerox Palo Alto Research Center, Palo Alto, CA, as a member research staff. His current research interests include VLSI circuits and systems design, computer architecture, and interactive display technology.



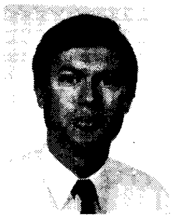
Shing I. Kong received the B.S. degree in mechanical engineering from Washington University, St. Louis, in 1982. Before undertaking graduate study at the University of California, Berkeley, he worked for the Computer System Laboratory at Washington University. In the fall of 1983, he began his graduate studies at U.C. Berkeley where he was a circuit designer for the SOAR project in 1984 and the chief designer of the SPUR CPU from 1985 to 1989. He received the M.S. and Ph.D. degrees in electrical engineering in 1985 and 1989, respectively.

Currently, he works for the advanced development group of Sun Microsystems at Mountain View, CA.

Mark D. Hill (S'80–M'87) received the B.S.E. degree in computer engineering from the University of Michigan, Ann Arbor, in 1981 and the M.S. and Ph.D. degrees in computer science from the University of California at Berkeley in 1983 and 1987, respectively.

He is an Assistant Professor in the Computer Sciences Department at the University of Wisconsin, Madison. His research interests center on computer architecture, with an emphasis on performance considerations and implementation factors in memory systems.

Dr. Hill is a member of ACM and a recipient of the National Science Foundation's 1989 Presidential Young Investigator award.



George S. Taylor received the B.S. degree in electrical engineering from Duke University, Durham, NC, in 1978. He will receive the Ph.D. degree from the University of California, Berkeley, in 1989.

He is a Senior Engineer at MIPS Computer Systems, Sunnyvale, CA, where most recently he led the design team for the R6000 ECL RISC microprocessor. Before joining the SPUR project, he designed the floating-point processor for the ELXSI 6400 minisupercomputer, wrote the design-rule checker for U.C. Berkeley's MAGIC VLSI CAD system, and developed a radix-16 divider for the Lawrence Livermore Laboratory's S-1 AAP computer project. His engineering interests include instruction set architecture, code scheduling, and VLSI packaging.

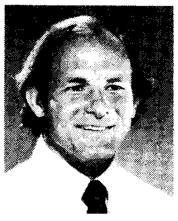
David A. Hodges (S'59–M'65–SM'71–F'77) earned the B.E.E. degree at Cornell University, Ithaca, NY, and the M.S. and Ph.D. degrees in electrical engineering at the University of California at Berkeley.

From 1966 to 1970 he worked at Bell Telephone Laboratories, first in the components area at Murray Hill, NJ, then as Head of the System Elements Research Department at Holmdel, NJ. Now he is Professor of Electrical Engineering and Computer Sciences at U.C. Berkeley, where he has been a member of the faculty since 1970. He became Chairman of the EECS Department on July 1, 1989. Since 1970 he has been active in teaching and research on microelectronics technology and design. He and H. G. Jackson are coauthors of *Analysis and Design of Digital Integrated Circuits*, a widely used textbook.

Prof. Hodges is founding Editor of the IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING. He is a former Editor of the IEEE JOURNAL OF SOLID-STATE CIRCUITS and a past Chairman of the International Solid-State Circuits Conference. With Robert Brodersen and Paul R. Gray, he received the 1983 IEEE Morris N. Liebmann Award for pioneering work on switched-capacitor circuits. He is a Fellow of the IEEE and a member of the National Academy of Engineering.

Randy H. Katz (S'74–M'80–SM'88) received the A.B. degree from Cornell University, Ithaca, NY, in 1976, and the M.S. and Ph.D. degrees from the University of California at Berkeley in 1978 and 1980, respectively.

After a year in industry, he taught at the Computer Sciences Department at the University of Wisconsin, Madison, from 1981 to 1983. He joined the EECS Department at U.C. Berkeley in 1983, where he is now a Full Professor. He is currently the Principal Investigator of a DARPA/NASA funded research project to build a high-performance I/O system for network file servers and very-high-performance computing.



David A. Patterson (M'84–SM'88) received the Ph.D. degree in computer science from the University of California, Los Angeles, in 1976.

He was first employed by Hughes Aircraft Company designing and evaluating computers. Since 1977 he has been a member of the faculty in the Computer Science Division, Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He was named Associate Professor in 1981 and Professor in 1985. He teaches computer architecture at both the graduate and undergraduate levels. Recently he has created and taught a course to introduce liberal art students to computers. This effort led to a set of books allowing these ideas to be tried at other institutions. He spent the fall of 1979 on leave of absence at Digital Equipment Corporation developing microprogram design tools. In the next year he led the design and implementation of RISC I, a 45000-transistor microprocessor that was likely the first VLSI RISC. The following year RISC II was completed, and this was the first university-built microprocessor that was accepted for publication at the International Solid-State Circuits Conference (ISSCC) as well as the second VLSI RISC. He recently finished leading the Smalltalk On A RISC (SOAR) project, which produced a 32000-transistor microprocessor that runs the object-oriented Smalltalk-80 system. These processors are the basis of the commercial RISC computer SPARC, sold by several computer and chip companies including AT&T, Sun Microsystems, Texas Instruments, and Xerox. He was most recently Principal Investigator of the Symbolic Processing Using RISC's (SPUR) project. This was an effort of six faculty and 25 graduate students to build a multiprocessor workstation based on three custom VLSI chips and a new operating system that supports both Common LISP and C as first-class citizens. He is currently working with Prof. Katz on developing input/output systems to match the increasingly higher performance of new processors.

Dr. Patterson received the Distinguished Teaching Award from the Berkeley Division of the Academic Senate of the University of California in 1982.