A 32-Bit NMOS Microprocessor with a Large Register File

ROBERT W. SHERBURNE, JR., MANOLIS G. H. KATEVENIS, MEMBER, IEEE,
DAVID A. PATTERSON, AND CARLO H. SÉQUIN, FELLOW, IEEE

Abstract—**Two scaled versions of a 32-bit NMOS reduced instruction set computer CPU, called RISC II, have been implemented on two different processing lines using the simple Mead and Conway layout rules with lambda values of 2 and 1.5 μm (corresponding to drawn gate lengths of 4 and 3 μm), respectively. The design utilizes a small set of simple instructions in conjunction with a large register file in order to provide high performance. This approach has resulted in two surprisingly powerful single-chip processors.**

I. Introduction

EVER more powerful and more complex processors are being integrated onto a single silicon chip. As chip functionality increases, the rising complexity confronting the designer becomes a serious concern. Tens of man-years are typically required to complete a 32-bit microprocessor design from the architectural specifications down to the circuit and layout levels. If the elapsed design time stretches over more than a few years, the fabrication technology will have changed so much that the original assumptions made regarding chip constraints no longer apply.

To minimize actual elapsed design time, chips are partitioned into modules, which are constructed in parallel by several design teams. Each module is optimized within the constraints assigned by a project leader or manager. This divide-and-conquer technique, the traditional approach in industry, has the disadvantage that no design team is familiar with the chip as a whole. This makes global or intermodule optimization difficult, if not impossible.

The overall organization of the system and the choice of a particular microarchitecture set general performance limits, and the shortcomings of poor decisions at the architectural level can be only partially compensated by sophisticated circuit design and layout optimization. The consequences of design decisions should thus be evaluated across multiple levels of the specification/implementation hierarchy. The difficulty of this process increases very quickly with increasing complexity of a system.

In view of constantly changing chip technology and constraints, design decisions must be reevaluated with each new computer. The switch in implementation technology from boards filled with SSI and MSI parts to a single MOS VLSI chip has a strong impact on the design tradeoffs. Most importantly, a single-chip CPU is governed by much more stringent constraints than a mainframe CPU: there are hard limits on the number of transistors, chip area, total power dissipation, and interconnection pins at the chip periphery. These limited resources must be balanced very carefully between the various functional modules in the CPU because additional hardware assistance given to one function will typically slow down other functions.

VLSI processor design must start with a critical analysis of the importance of different functions for the selected application area and their relative cost of implementation. Specifically, one must start with an evaluation of the contribution of each proposed instruction to the overall performance of the computer system when executing an anticipated mix of computational tasks.

II. Reduced Instruction Set

An analysis of the microcode of a minicomputer with some 300 instructions (DEC VAX-11/780) and of the dynamic instruction count in a typical job mix shows that 20 percent of the instructions, accounting for about 60 percent of all microcode, are used less than 1 percent of the time [1]. Replacement of these instructions by software routines does not significantly degrade performance. When a single-chip CPU implementation is considered, the resulting reduction in the size of the microcode has considerable benefits. Less complicated circuitry means shorter delays and a potentially faster machine cycle which benefits all instructions. Furthermore, a reduction in the area and the number of transistors in the control section frees up chip resources for the support of other functions. Last but not least, reducing the complexity and internal state of the machine simplifies the design and testing tasks, resulting in an earlier market entry.

Recently, several NMOS, single-chip VLSI microprocessors have been designed with this idea in mind. The RISC I [2], RISC II [3], and MIPS [4] architectures employ low-level instructions, each executing in one machine cycle. This regular execution timing simplifies the implementation of pipelining and retains conceptual simplicity. The

Manuscript received March 12, 1984; revised May 9, 1984. This work was supported by the Defense Advanced Research Projects Agency, U.S. Department of Defense, under ARPA Order 3803, monitored by the Naval Electronic Systems Command under Contract N00039-81-K-0251

The authors are with the Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

0018-9200/84/1000-0682$01.00

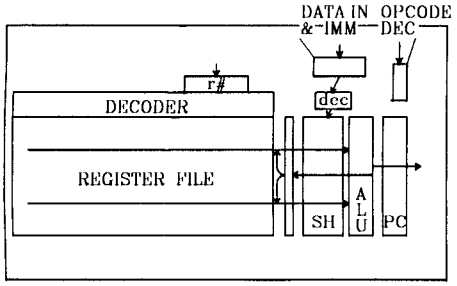


Fig. 1. Allocation of chip area resources in RISC II.

resulting chips look quite different from commercial single-chip microprocessors, which often dedicate up to half their die area to the control for the datapath. In the reduced instruction set computer (RISC) less instruction decoding and control logic is necessary, making possible the use of small, fast, programmed logic arrays (PLA's). The delays through the instruction decoder are reduced so that they are no longer in the critical path that limits processor performance. The area freed up by the reduced control circuitry is used for an expanded register file (Fig. 1) to provide fast access to operands.

The addition of a large local memory is not the only way the freed-up chip area can be used. For different application domains, other functions may play an even more important role in enhancing overall systems performance. For example, in a RISC implementation of a Smalltalk processor [5] extra tag checking logic is included on the chip to improve the performance of variable length integer arithmetic and of garbage collection.

The instruction set of RISC II is presented in Table I. Arithmetic operations are limited to simple add and subtract instructions. Logical operations and arbitrary left and right shifts are also provided. Multiplication, division, and floating point operations are performed by subroutines or with the use of an off-chip coprocesssor, depending on the needs of the system. Data loads and stores support byte, half-word, and full-length 32-bit formats. Also included are conditional jumps, calls, and returns, as well as a few miscellaneous instructions dealing with interrupts or with the composition of a full-length 32-bit address in a register. In all, there are only about 30 instructions in RISC II, in stark contrast to other machines with more than ten times as many instructions.

Instruction execution follows a 3-address, register-to-register format. In each machine cycle two operands are fetched in parallel, are operated upon by the arithmetic/logic unit (ALU) or by the shifter, and the result is written back to the register file in a pipelined manner during the next instruction execution. As a result of the small set of simple instructions and their regular execution pattern, the control logic is almost invisible. The instruction decoder is shown in the upper right corner of Fig. 2. It consists of a generalized NOR decoder with only 8 inputs, calculating 36 different product terms. A few of these product terms are replicated as necessary in order to provide the desired ordering of output terms to the datapath modules. The second level of NOR gates, in which product terms are summed, is very small: an average of 1.5 minterms per product term are utilized for the 30 different decoder outputs. This approach thus yields a very compact and fast instruction decoder. The other fields in the instruction format, the source and destination addresses, are decoded in separate small circuit modules close to the area on chip where the generated signals are actually used.

III. Increased Local Memory

A fundamental limit of computer performance is reached when memory traffic equals the available input/output (I/O) bandwidth of the processor chip. The I/O bandwidth limit for a given technology is set by area and power constraints. Only a limited number of I/O pads with their associated driver circuits can be accommodated on a single chip. The speed of I/O drivers is bounded by a delay–power product, since the off-chip loading is primarily capacitive. Multiplexing the pads for several I/O transactions per cycle, requires a faster settling time and hence greater power dissipation.

Memory traffic consists of two classes of information: instructions and data. Several options are available for reducing either component. At the architectural level, the set of machine instructions may be designed to include powerful constructs equivalent to many simple instructions. This has been done traditionally for large mainframe computers to increase the effective bandwidth at which instructions can be delivered to the CPU. RISC implementations, on the other hand, choose a less strongly encoded instruction format which leads to a simple and small control section. Instead, they make a special effort to minimize data memory traffic. Much of the chip area is devoted to a large register file that can store many of the frequently used operands on chip, thereby reducing data traffic through the chip periphery.

A register-based machine can store frequently used operands in a fast, multiple-port register file. Register allocation is performed by the compiler. It is performed independently for each subroutine; thus register contents may have to be swapped off-chip in order to make room for the next procedure [Fig. 3(a)]. For many machines, this makes the subroutine call and return a very time-consuming operation resulting in a significant performance degradation when executing typical high-level language programs.

In order to reduce this overhead associated with procedure calls, the RISC II microprocess provides a lot of local memory, organized in multiple, overlapping register banks, also called *windows*. Each bank supports a different level of the dynamic procedure calling hierarchy [Fig. 3(b)]. The active procedure has access to a total of 32 registers. Ten of these registers are local to the present procedure level. There are also six *high* and six *low* registers which are shared by adjacent procedure levels; they are used primarily for passing parameters and results between procedures without the need for explicit data movements. In addition, ten global registers are accessible from any procedure level. Operands or parameters that do not fit into this fixed register framework are stored in main memory. The overlapping banks are implemented as a contiguous block

TABLE I
INSTRUCTION SET OF RISC II

Instruction	Operands	Comments	
add	Rd,Rs,S2	Rd ← Rs + S2	integer add
addc	Rd,Rs,S2	Rd ← Rs + S2 + carry	add with carry
sub	Rd,Rs,S2	Rd ← Rs - S2	integer subtract
subc	Rd,Rs,S2	Rd ← Rs - S2 - borrow	subtract with borrow
subi	Rd,Rs,S2	Rd ← S2 - Rs	integer subtract
subci	Rd,Rs,S2	Rd ← S2 - Rs - borrow	subtract with borrow
and	Rd,Rs,S2	Rd ← Rs & S2	bitwise logical AND
or	Rd,Rs,S2	Rd ← Rs | S2	bitwise logical OR
xor	Rd,Rs,S2	Rd ← Rs xor S2	bitwise logical EXCLUSIVE OR
sll	Rd,Rs,S2	Rd ← Rs shifted by S2	shift left
srl	Rd,Rs,S2	Rd ← Rs shifted by S2	shift right logical
sra	Rd,Rs,S2	Rd ← Rs shifted by S2	shift right arithmetic
ldw	Rd,(Rx)S2	Rd←M[Rx+ S2]	load word
ldhu	Rd,(Rx)S2	Rd←M[Rx+ S2] (align, zero-fill)	load half unsigned
ldhs	Rd,(Rx)S2	Rd←M[Rx+ S2] (align, sign-ext)	load half signed
ldbu	Rd,(Rx)S2	Rd←M[Rx+ S2] (align, zero-fill)	load byte unsigned
ldbs	Rd,(Rx)S2	Rd←M[Rx+ S2] (align, sign-ext)	load byte signed
stw	Rm,(Rx)S2	M[Rx+ S2]←Rm	store word
sth	Rm,(Rx)S2	M[Rx+ S2]←Rm (align)	store half
stb	Rm,(Rx)S2	M[Rx+ S2]←Rm (align)	store byte
jmpx	COND,(Rx)S2	if COND then PC←Rx+ S2	cond. jump, indexed, delayed
jmpr	COND,Y	if COND then PC←PC+ Y	cond. jump, PC-rel., delayed
callx	Rd,(Rx)S2	Rd←PC; PC←Rx+ S2; CWP--	call indexed, change window
callr	Rd,Y	Rd←PC; PC←PC+ Y; CWP--	call PC-rel., change window
ret	(Rx)S2	PC←Rx+ S2; CWP+ +	return, change window
ldhi	Rd,Y	Rd<31:13>←Y; Rd<12:0>←0	load immediate high
gtlpc	Rd	Rd←lastPC	to restart pipeline after interrupts
getpsw	Rd	Rd←PSW	read status word
putpsw	Rm	PSW←Rm	set status word
reti	(Rx)S2	PC←Rx+ S2; CWP+ + ; en.interr.	return from interrupt
calli		hardwired, on interrupts: CWP--; R25←lastPC; PC←Intr.Vect.; disable intr.	

Rd, Rs, Rx, Rm: a register (one of 32, where R0≡0); S2: either a register or a 13-bit immediate constant; COND: 4-bit condition; Y: 19-bit immediate constant; PC: Program-Counter; CWP: Current-Window-Pointer; All instructions can optionally set the Condition-Codes.

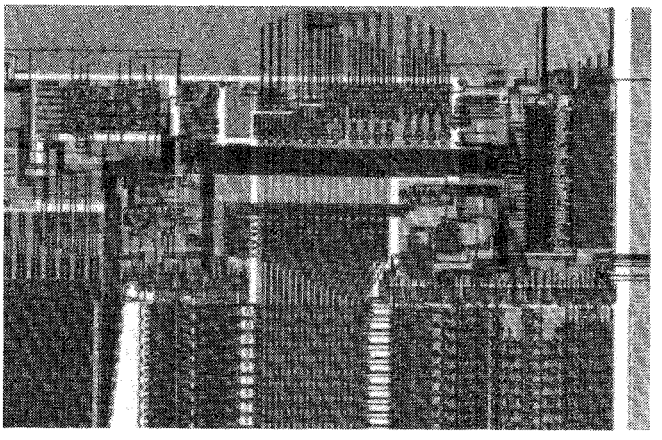


Fig. 2. Control logic area (instruction decoder in upper right).

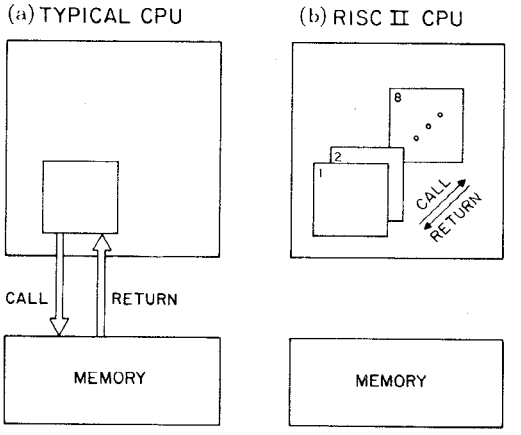


Fig. 3. Procedure call and return in register-based machines.

of 138 registers in which at any one time a group of 22 registers is singled out by a window pointer; the latter moves in steps of 16 register positions for each procedure call or return. 128 of the registers are used in this manner to implement a circular buffer of 8 register banks. When the procedure nesting depth exceeds the size of this buffer, an overflow occurs and one or two register banks [6] are swapped back to main memory to free up new register space for the called procedure. These registers are restored from main memory when an underflow occurs.

Since this multiple-bank local memory may occupy a significant portion of the chip area, it is important to ensure that it makes effective use of available resources, and the optimal size and number of these windows should

TABLE II
NORMALIZED RISC II EXECUTION TIME

WINDOWS		1	2	3	5	7	9	∞
Normalized RISC II Register Cycle Time		1.00	1.22	1 41	1.73	2 00	2.24	∞
TOWER 19% Dynamic Call&Return	(architecture)	7.08	3 02	2.52	1 38	1.10	1.02	1.00
	(system)	7.08	3.68	3.55	2.39	2.20	2.28	∞
PUZZLE 0 7% Dynamic Call&Return	(architecture)	1.17	1.02	1 00	1 00	1.00	1.00	1.00
	(system)	1.17	1.24	1 41	1 73	2 00	2.24	∞

(Architecture; Based on machine cycle time.)
(System: Scaled with register file cycle time.)

be determined carefully. Architectural simulations [7] indicate that over/underflows occur in only a few percent of all procedure calls if the register file contains eight windows, and that most scalars can be accommodated in local memory with the specific numbers of registers per window mentioned above. But concentrating on architectural simulations alone overlooks the fact that too large a register file also leads to performance penalties. The increased parasitic capacitances of longer buses in a larger register file stretch out the basic machine cycle. A proper analysis of the tradeoffs must encompass architectural as well as circuit design issues [8].

Some results of such a study are presented below. Table II gives the relative execution times of two *C* programs. *Tower of Hanoi* and *Puzzle*, versus the number of windows on the chip. Both benchmarks nest to a depth of twenty, but *Tower* has a very high rate of procedure calls and returns (19 percent) and thus makes intensive use of the multiple register banks. Performance improves noticeably up to the use of about seven windows. In *Puzzle*, on the other hand, only 0.7 percent of all instructions executed are calls and returns; the program performs adequately with only one or two windows. The first entry in each field (architecture) is solely based on architectural studies of procedure behavior [7] and register file management overhead for RISC [6]. The second set of numbers (system) also takes into account the lengthening of the machine cycle with increasing size of the register file [8]. They clearly show a point of diminishing return in the number of register windows. When considering more sophisticated schemes that reduce the swapping overhead by saving and restoring only the registers actually used by a procedure, the optimum number of windows shifts to even lower values. These studies show that seven windows (plus one to handle interrupts) are a practical upper bound for the size of the register file—even for very procedure call intensive programs.

IV. PIPELINING AND DATAPATH TIMING

In a single-chip design, datapath speed is a critical factor in system performance. Careful selection and orchestration of the resources in the datapath are thus the prime task of the microarchitect. Standard ways of improving performance include the use of pipelining in the datapath (requiring more buses) or enhancing the computational power of its modules (requiring more circuitry). Because chip area is limited, only a few of the many possible techniques for improving performance can be implemented simultaneously. Careful analysis is necessary to ensure that overall system throughput, not just the speed of a few instructions is maximized.

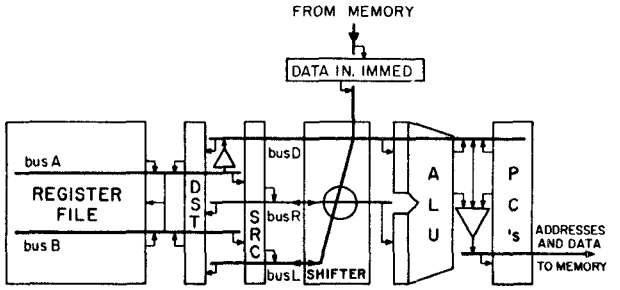


Fig. 4.

The RISC II datapath is illustrated in Fig. 4. Two 32-bit buses carry the data from the dual-port register file to the ALU. These buses also carry the result of the ALU and its complement back to the register file. An additional bus is incorporated into the shifter, which facilitates alignment of load and immediate data. A separate bus is also used in order to interface with off-chip data and instruction memories.

Each instruction stretches over three machine cycles as shown in Fig. 5. The first cycle consists of the instruction fetch and decode. Because of the reduced instruction set, decoding requires only 25 percent of the cycle time, leaving the remaining 75 percent for memory access. During the second cycle, a dual-port operand read and the ALU or shift operations are performed; the result is temporarily stored in the DST register. At the end of the third cycle, this result is written into the register file. By delaying the write operation, the basic cycle can be shortened from *read-modify-write* to *read-modify*. Another advantage of delaying the write operation is that it allows more time for detecting and processing interrupt requests, thus facilitating restartability of the system.

Overall execution is pipelined (Fig. 6), so that at any one time there are three instructions being processed. The operand fetch is performed in parallel with the instruction fetch for the next cycle, while the operand write of the previous instruction is performed simultaneously with the ALU or shift operation of the current instruction and the decoding of the next one. This pipelining readily follows from the regularity of the RISC instruction execution.

Although the timing sequence of Fig. 6 ensures that there is no contention for datapath resources, performance may still be degraded by data dependencies among consecutive instructions. Since the ALU output from one instruction is not written into the register file until the end of the following cycle, the result is not accessible during the operand fetch of that instruction. The correct data can only be found in the register file after an extra cycle of delay.

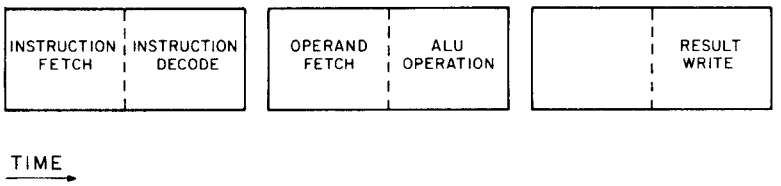


Fig. 5. RISC II instruction execution.

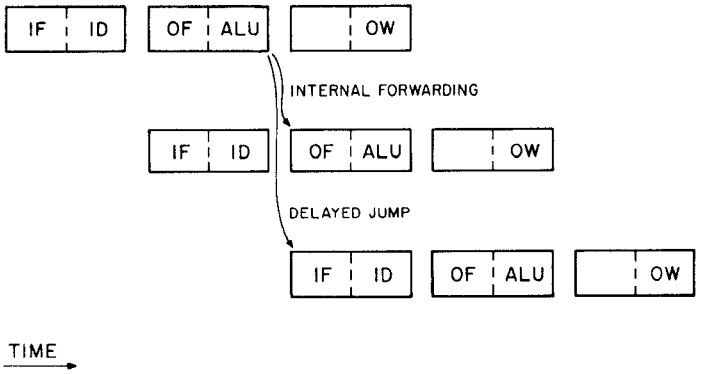


Fig 6. Instruction pipelining and dependencies.

Another type of dependency exists with jump or branch instructions. In this case the ALU phase performs the target address calculation, but the result arrives too late to be used for the next instruction fetch. A single cycle of delay must be inserted after the jump instruction as well.

In order to minimize performance loss from these pipeline dependencies, two strategies are adopted in RISC II. First, all data dependencies are eliminated by allowing the data temporarily stored in the DST register (Fig. 4) to be accessed during operand fetching (*internal forwarding*). These dependencies are detected by register address comparators on the chip; overhead for this logic is less than 1 percent of the chip area. Jump dependencies are eliminated through *delayed jump* execution and by program reordering. The instruction immediately following the jump instruction is always executed *before* the branch is actually taken. If possible, some useful instruction is inserted by the compiler; in the remaining cases a NOOP (*no o*peration) is inserted. Normally, more than 50 percent of the slots after branch instructions can be put to use.

Because the RISC II CPU has a single I/O bus, operation of the pipeline must be temporarily suspended when external data memory is accessed (Fig. 7). Data loads follow the same timing as instruction fetches, with an alignment operation replacing instruction decoding. This data is also stored in the DST register so that dependencies may be overcome by internal forwarding.

This data I/O wait-state could be eliminated with an additional level of pipelining, but the necessary changes would be costly. First, two I/O operations (instruction and data) would have to be performed in each cycle, necessitating two separate I/O buses. Second, a dual-port write would be required in the register file when executing data loads. Third, a separate wait-state cannot be avoided when dependencies on the loaded data must be handled. Because

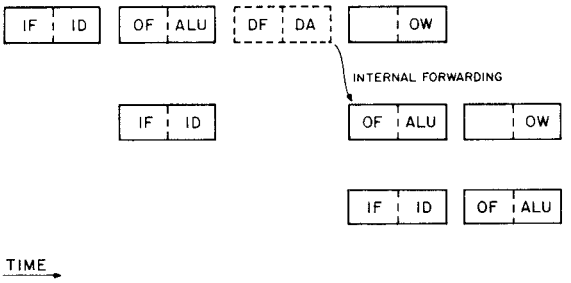


Fig 7. RISC II data I/O execution.

of these costs and because of the rarity of external data fetches in RISC II, such a four-way pipeline scheme was not pursued.

V. Circuit Design and Cell Layout

Because a large portion of the die is taken up by the register file, the design of a compact bit cell is crucial. For RISC II the 3-bus approach of RISC I was thus abandoned in favor of a 2-bus register file. We started with a standard 6-transistor static cell. A second wordline was added to allow dual-port reading (Figs. 9 and 10). The dual-port read is performed with single-ended sensing since differential sensing would require an additional pair of bitlines. Normally, these bitlines could pass over the depletion load devices without enlarging the cell, but with the implementation technology used, the butting contacts block the channel where otherwise a second set of metal buses could have been placed.

Fig. 11 shows this dual-port static cell with its associated precharge, sensing, and decode logic. The bitlines are precharged prior to reading; during the read, other precharged buses are driven throughout the datapath. The static NOR

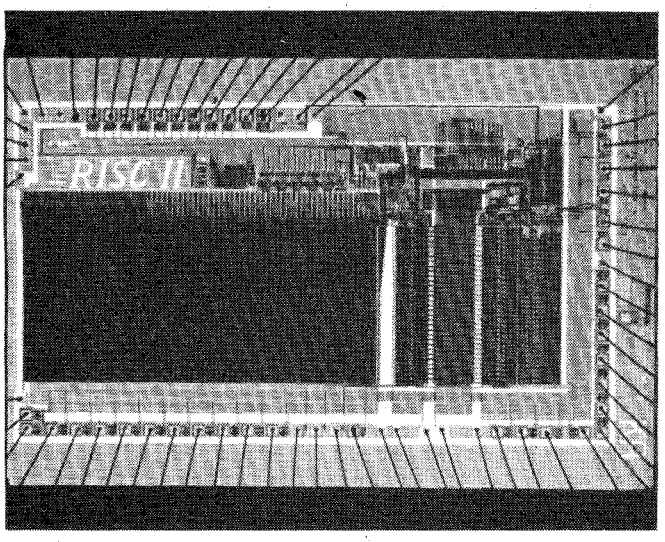


Fig. 8. RISC II chip photomicrograph.

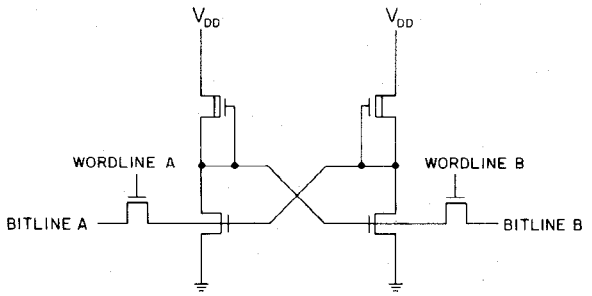


Fig. 9. Dual-port static register cell.

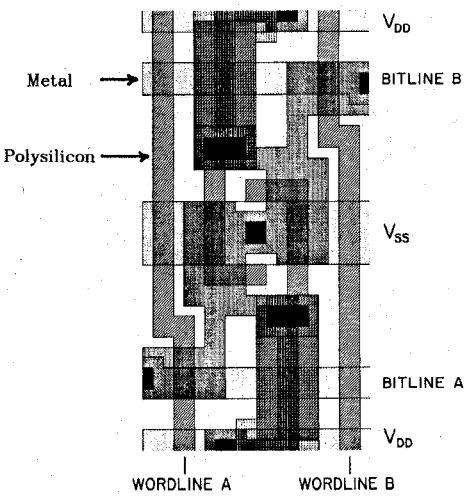


Fig. 10. Layout of dual-port static bit cell.

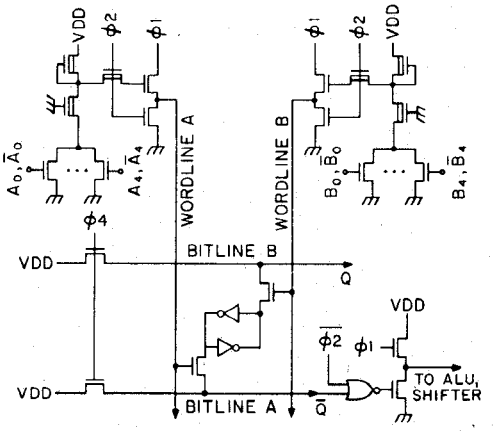


Fig. 11. Register file cell with decoders and single-ended sensing logic.

decoders utilize series depletion devices with grounded gates in order to isolate the output node from input parasitics when charging. The full 5 V signal is passed to the bootstrapped wordline drivers by use of dynamic depletion mode pass transistors. Each decoding operation requires two clock phases; decoders are shared for both reading and writing, leading to a four-phase cycle. Register write is performed by driving pairs of bitlines differentially with the write data (as is done in conventional static RAM's) and providing identical addresses to both word decoders.

Precharged buses and logic were used in most areas of the datapath. The 32-bit ALU is a dynamic ripple carry design in which the carry is buffered after every four bits. The shifter utilizes two precharged, bidirectional buses; this allows both right and left shifts to be performed using a single 32-bit, arbitrary-amount shifter.

In addition to the 8-input instruction decoder, the control logic contains circuitry for interrupt processing, for conditional branch calculation, and for internal forwarding to resolve data dependencies. The program status word includes user mode, interrupt state, and condition code information. To simplify debugging and chip evaluation, scan in/scan out logic was added to give access to the machine state, and the four-phase clock and substrate bias are generated off-chip.

VI. Chip Implementation and Evaluation

The RISC II CPU chip was designed and simulated for compatibility with a range of *silicon foundry* processes. The result is a conservative circuit design that can accommodate a wide range of process capabilities and parameter variations. For the layout, the simple and scalable Mead and Conway design rules [9] were employed. In addition, since many of our design tools impose such a restriction, all geometry was restricted to rectilinear features aligned with the coordinate system ("Manhattan geometry"). The anticipated fabrication process provides a single level of metal and one level of about 30 Ω/square polysilicon. No *buried* contacts are available; instead, contact between the polysilicon layer and the source/drain regions requires a metal patch placed over a contact cut overlapping both these two layers. This reduces layout density significantly because metal wires must be spaced some distance from these *butting* contacts. Since no exact process parameters were available at the time the circuit design had to be finalized, fairly large parameter variations for various doping concentrations, sheet resistivities, and threshold voltages were used in the SPICE2 [10] simulations of the critical circuits. The RISC II CPU required over forty

thousand transistors, yet it was designed, simulated, and tested using less than three man-years of effort.

The design was found to be functionally correct on its first silicon run. This is the result of extensive simulation at several different levels. Processor descriptions were produced at three levels. First, an ISP description was used to guarantee that we had defined a useful instruction set. A second representation was formed with a mixed-level description, called SLANG [11], spanning the register transfer and logic gate levels. This "home-brewed" language based on Lisp allowed a full description of the chip behavior using less than 300 explicit "node." All the different instructions were simulated at this level and compared against the ISP description to guarantee that our SLANG description indeed represented a processor with the desired architectural behavior. The final step was to extract transistors and their interconnections from the mask geometry specification, using MEXTRA [12]. The resulting circuit was used in a switch-level simulator run in tandem with the SLANG simulator to verify that the actual layout behaved like the logic description expressed in SLANG. Detailed circuit simulation using SPICE2 was only used for isolated critical circuit modules and for tuning the performance of the various bus systems. However, the complete circuit was subjected to the timing analyzer program *Crystal*, written by Ousterhout [13]. This program identified the critical delay paths through our circuit and helped us find design errors that might impair performance. It also identified modules that might warrant a more detailed analysis with a SPICE2 simulation. Because of the thoroughness of these simulations it is not too surprising that the design was functionally correct.

The first chips were fabricated with a 4 μm gate length process (lambda = 2 μm) using the MOS Implementation Service (MOSIS) at the Information Science Institute at the University of Southern California. The chip size was constrained by a package cavity size of about 11 mm, and this constrained the total length of the datapath. Within this limit we were able to integrate the desired set of eight register banks. Unlike the RISC I chip, this design was also operating close to the expected performance. The chips with the 4 μm gates worked with the external clocks set to a cycle time of 500 ns, which was 5 percent above the nominal target value. This must be attributed to the availability of the *Crystal* timing analyzer.

A few months later, a 3 μm version was created by simply scaling down the entire mask pattern. This second batch of chips was implemented at the Xerox PARC facilities. Even though these chips went through the scaling procedure and were run on a different fabrication line with a different process, no new simulations were carried out. The hope was that our original design had wide enough margins to tolerate a 4:3 reduction. The scaled-down chips indeed worked and ran 50 percent faster than the 4 μm version, i.e., with a cycle time of 330 ns.

The 4 μm RISC II processor operating at an 8 MHz clock rate, corresponds to a peak rate of 2 million instructions per second (MIPS). The chip consumes 1.25 W. The 3 μm version exhibits a peak throughput a 3 MIPS, with a 50 percent increase in power dissipation. These results were obtained at room temperature, using a single 5 V supply and no substrate bias. These specifications are summarized in Table III.

TABLE III
SPECIFICATIONS OF THE RISC II CPU IMPLEMENTATIONS

SPECIFICATIONS	CHIP A	CHIP B
DRAWN GATE LENGTH	4 μm	3 μm
DIE SIZE (mil^2)	228 x 406	171 x 304
REGISTER CELL AREA	4.6 mil^2	2.6 mil^2
CLOCK RATE	8 MHz	12 MHz
POWER DISSIPATION	1 25 W	1 83 W
REGISTER FILE	138 x 32 bits	
TRANSISTOR COUNT	40706	
PIN COUNT	62	
DESIGN TIME	2.8 man-years	

Using integer *C* programs compiled with the standard UNIX compiler as benchmarks, and the above operating speeds for calibration, it turns out that both versions of RISC II can outperform popular processors such as the Motorola 68000, Hewlett-Packard 9000, Intel 80286, and the DEC VAX-11/780 [14]. We attribute this surprising performance to the good match between the RISC architecture and the needs and constraints of a VLSI single-chip processor.

VII. Summary

The RISC II CPU was designed with heavy emphasis on optimizing its architecture for the intended application area (i.e., running compiled *C* programs in a UNIX environment) and on matching the special constraints of VLSI implementation. Circuit design was kept straightforward and conservative. The layout was produced with a rectilinear subset of the Mead and Conway design rules, but the critical modules used in iterative arrays, such as the register or ALU cells, were carefully optimized for density and for low power consumption. Implementation was aimed at a "standard" 4 μm Si-gate NMOS process that was becoming readily available from several "silicon foundries."

Chip performance is not determined by implementation technology alone. Through careful studies of the needs of an application area, and proper matching of the architecture to these needs, significant performance gains were achieved which more than offset the losses due to conservative layout and implementation technology. By strongly reducing the size of the implemented instruction set, precious chip area was freed up for a large local memory consisting of multiple, overlapping register banks. This alleviated the chip I/O bottleneck by keeping frequently used operands on the chip. The simple and regular instruction set also eased the design process by reducing system complexity. The outcome of this work is a single-chip CPU that rivals board-level designs utilizing many chips.

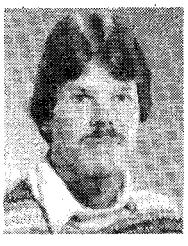
Chips have been implemented at two different gate lengths—4 and 3 μm—on two different process lines. The two implementations were functionally correct and operated close to the desired speed in first silicon. This is a result of extensive simulation at several levels, made possible by our design environment that places the necessary tools at the fingertips of the designers.

ACKNOWLEDGMENT

The authors would like to thank the MOS Implementation Service at USC/ISI for the fabrication of the 4 μm RISC II chips and thank A. Bell, J. Knight, and Xerox PARC for implementing RISC II at 3 μm gate length. We would also like to thank all our colleagues and students who have helped to create an environment in which such a project is feasible.

REFERENCES

[1] R. Supnik *et al.*, "A 32 bit microprocessor with on-chip virtual memory management," presented at the IEEE Int. Solid-State Circuits Conf., San Francisco, CA, pp. 178–180, Feb. 1984.
[2] D. A. Patterson and C. H. Séquin, "A VLSI RISC," *Computer*, vol. 15, no. 9, pp. 8–21, Sept. 1982.
[3] M. G. H. Katevenis, R. W. Sherburne, D. A. Patterson, and C. H. Séquin, "The RISC II microarchitecture," in *Proc. VLSI'83, Int. Conf. Very Large Scale Integration*, Trondheim, Norway, Aug. 1983, pp. 349–359.
[4] C. Rowen, S. A. Przybylski, N. P. Jouppi, T. R. Gross, J. D. Shott, and J. Hennessy, "A pipelined 32 bit NMOS microprocessor," in *Proc. IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, Feb. 1984, pp. 180–181.
[5] D. Ungar, R. Blau, P. Foley, D. Samples, and D. Patterson, "Architecture of SOAR: Smalltalk on a RISC," in *Proc. 11th Symp. Computer Architecture*, Ann Arbor, MI, June 1984.
[6] Y. Tamir and C. H. Séquin, "Strategies for managing the register file in RISC," *IEEE Trans. Comput.*, vol. C-32, pp. 977–989, Nov. 1983.
[7] D. Halbert and P. Kessler, "Windows of overlapping registers," CS292R Final Rep., Univ. Calif., Berkeley, June 1980.
[8] R. W. Sherburne, "Processor design tradeoffs in VLSI," Ph.D. dissertation, Comput. Sci. Div., Univ. Calif., Berkeley, Apr. 1984.
[9] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.
[10] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," ERL Memo ERL-M520, Univ. Calif., Berkeley, May 1975.
[11] K. S. Van Dkye, "SLANG: A logic simulation language," Masters thesis, Univ. Calif., Berkeley, June 1982.
[12] D. T. Fitzpatrick, "MEXTRA: A Manhattan circuit extractor," ERL Memo M82/42, Univ. Calif., Berkeley, May 1982.
[13] J. K. Ousterhout, "Crystal: A timing analyzer for NMOS VLSI circuits," in *Proc. 3rd Caltech Conf. VLSI*, Pasadena, CA, Mar. 1983.
[14] D. A. Patterson, "RISC watch," *Computer Architecture News*, vol. 12, pp. 11–19, Mar. 1984.



Robert W. Sherburne, Jr., received the B.S. degree in electrical engineering from Worcester Polytechnic Institute in Worcester, MA, in 1978. He received the M.S. and Ph.D. degrees in 1981 and 1984, respectively, in electrical engineering and computer science from the University of California, Berkeley.

He is now an Assistant Professor with the Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY. His research interests include VLSI architecture, microarchitecture, and design optimization.

Manolis G. H. Katevenis (S'76–S'82–M'83) was born in Athens, Greece, in 1955. He received the Diploma degree in electrical engineering, from the National Technical University of Athens, Greece, in 1978, and the M.Sc. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1980 and 1983, respectively.

During his graduate studies he has been a Research Assistant with the Electronics Research Lab, University of California, Berkeley, and from 1981–1983 he held an IBM graduate student fellowship. His doctoral research was on reduced instruction set computer architectures for VLSI, and on the RISC II NMOS microprocessor. Since January 1984, he has been an Assistant Professor in the Computer Science Department and Computer Systems Laboratory of Stanford University, Stanford, CA.

Mr. Katevenis is a member of the Association for Computing Machinery, the Technical Chamber of Greece, and the Association of Computer Scientists of Greece.

David A. Patterson received the Ph.D. degree in computer science from the University of California, Los Angeles, in 1976, while 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, University of California, Berkeley, and was named Associate Professor in 1981. He teaches computer architecture at both the graduate and undergraduate level, and in 1982 he received the Distinguished Teaching Award from the Berkeley Division of the Academic Senate of the University of California. 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. He is currently leading the Smalltalk On A RISC (SOAR) project, whose goal is to produce a microprocessor for Smalltalk-80. His research tries to combine popular software, experimental architecture, and very large scale integrated circuits to create more efficient computer systems.

Carlo H. Séquin (M'71–SM'80–F'82) received the Ph.D. degree in experimental physics from the University of Basel, Switzerland, in 1969. His subsequent work at the Institute of Applied Physics of the University of Basel concerned interface physics of MOS transistors and problems of applied electronics in the field of cybernetic models.

From 1970 to 1976 he worked at Bell Telephone Laboratories, Murray Hill, NJ, in the MOS Integrated Circuit Laboratory on the design and investigation of charge-coupled devices for imaging and signal processing applications. He has written many papers in that field and is an author of the first book on charge-transfer devices. He spent the academic year 1976 to 1977 on leave of absence with the University of California, Berkeley, where he lectured on integrated circuits, logic design, and microprocessors. In 1977 he became a Professor with the Department of Electrical Engineering and Computer Sciences. His research interests include computer architecture and design tools for very large scale integrated systems. In particular his research concerns multimicroprocessor computer networks, the influence of VLSI technology on advanced computer architecture, and the implementation of special functions in silicon. His most recent interests lie in the area of computer graphics and solids modeling. From 1980 to 1983, he was head of the Computer Science Division in the EECS Department.

Dr. Séquin is member of Association for Computing Machinery and the Swiss Physical Society.