R4200: A High-Performance MIPS Microprocessor for Portables

Barbara Zivkov, Brian Ferguson, Mayank Gupta

MIPS Technologies, Inc., Silicon Graphics, Inc.
2011 N. Shoreline Blvd., Mountain View, CA, 94039

Abstract

The MIPS R4200 is a low-power, low-cost RISC processor targeted at portable computing and embedded applications. Fully compatible with the MIPS R4000 family of high-performance microprocessors [1], the R4200 will power notebook, portable and low-cost desktop systems running either Windows NT or Unix. The R4200 is specifically designed for low power consumption and achieves low cost through small, high-yield die while retaining high levels of performance.

1: Introduction

The motivation for the development of the R4200 MIPS RISC microprocessor was driven by the industry trend toward more compact portable computers and "green PC" applications. Because RISC technology has been embraced in the Unix-based workstation marketplace, new RISC microprocessor development has typically pushed the performance envelope, resulting in more-expensive chips consuming greater amounts of power. The R4200 microprocessor, on the other hand, provides integer performance better than the R4000PC at only a fraction of its power consumption and cost, making it possible to bring RISC performance levels to a much wider range of products.

The R4200 designers initially considered modifying the existing R4000 design but soon realized that to meet our aggressive cost, power and performance goals this would not be feasible. Therefore we decided to do a completely new design which allowed us to optimize critical blocks of the design to achieve our objectives. Meeting our goal of decreasing the power dissipation while maximizing die yield provided challenges in all aspects of the design: the R4000-compatible micro-architecture that delivers the performance, the logic and circuits which implement the micro-architecture, and the layout of the die itself

Design trade-offs which are often decided in favor of increasing performance were more carefully weighed in terms of performance bought at the expense of increased power and die area. Increased die area results not only in greater power consumption but in a higher chip cost.

The following sections discuss the implementation of the R4200, with particular emphasis given to the design trade-offs that faced the engineering team and the manner in which they were resolved. A block diagram of the R4200 is shown in Figure 1: R4200 processor block diagram.

1.1: Features

The R4200 is a highly-integrated (1.4 million transistors) 64-bit RISC microprocessor that provides high integer performance at a fraction of the power and cost of comparable microprocessors. See Table 1: Performance/implementation data for details. Its features include:

- Integrated floating-point and integer datapaths
- On-chip 16-KB instruction and 8-KB data caches
- 32-double-entry joint TLB and 2-entry instruction micro-TLB
- Flush buffer
- Kernel-, supervisor- and user-level R4000/R4400 compatibility [2]
- R4000/R4400-compatible system interface
- Reduced power mode and instant-on/off capability
- Low-cost plastic and R4000/R4400 pinout-compatible packages available
- 55 SPECint92 and 30 SPECfp92 (estimate)
- 1.5 W power dissipation (estimate)
- 3.3 V supply voltage

1063-6390/94 $3.00 © 1994 IEEE

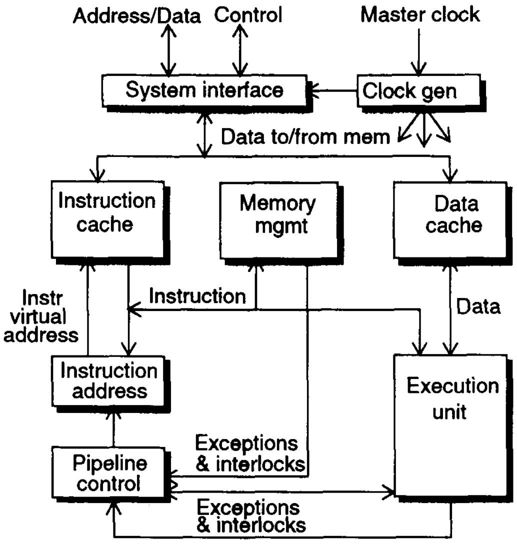


Figure 1: R4200 processor block diagram

1: CPU pipeline

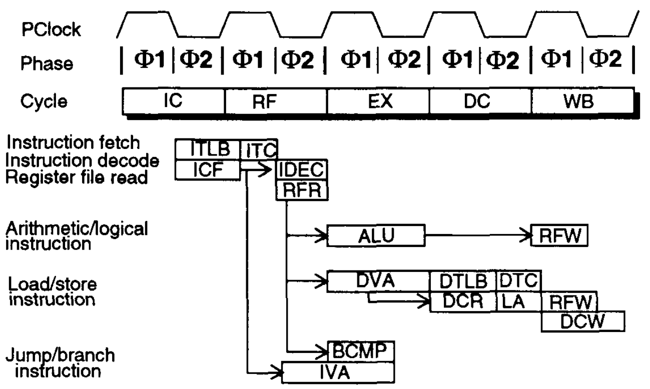
The R4200 processor has a single five-stage execution pipeline. Each pipeline stage takes one pipeline clock (PClock) to complete. PClock normally runs at twice the frequency of the external MasterClock. The execution of each instruction has a minimum latency of five PClocks and a new instruction can start every PClock. When the pipeline is not stalled, the processor has a throughput of one instruction per PClock. The pipeline performs in-order issue, in-order execution and in-order completion. Figure 2: Pipeline activities shows the activities during each pipeline stage for each instruction type.

The R4200 implements a single five-stage execution pipeline since this represents the most efficient trade-off between power, die area and performance compared with either a superpipelined or superscalar design [3]. Both of these techniques would have given performance advantages but the cost in terms of power and area was too great for the design objectives of the R4200.

A superscalar design achieves higher performance by having a number of execution units operating in parallel on different instructions [3]. This approach requires more die area while adding considerably to the design complexity. The instruction-dispatching logic becomes more complex since the ability to issue more than one instruction per pipeline clock requires extra structural and data hazard checking. The register file becomes more complex due to the need for extra read and write ports to service the multiple execution units. Indeed, the R4200 reduced parallelism beyond that of typical non-superscalar designs by merging the integer and floating point units, including the register files, into a unified execution datapath, further reducing hardware. This is discussed in a companion paper.

A superpipelined design achieves higher performance by further pipelining functional units, allowing a higher clock frequency. This requires more hardware and more complex control logic than traditional pipelining and increases the average cycles per instruction due to longer branch and load delays[4]. For a low-power and low-cost design, a shorter pipeline is better since it results in a lower clock frequency, and, therefore, lower power dissipation and simpler hardware. The performance loss due to the lower clock speed is offset somewhat by the improved cycles per instruction due to shorter branch and load delays; therefore, a traditional five stage pipeline is more optimal for a low cost, low power processor.

The branch instructions in the MIPS architecture include one architecturally-specified delay slot which defines that the instruction immediately following the branch executes regardless of whether the branch is



Cycle	Phase	Mnemonic	Description
IC	Φ1		
	Φ2	ICF	Instr cache fetch
		ITLB	Instr micro-TLB read
RF	Φ1	ITC	Instr cache tag check
	Φ2	RFR	Register File Read
		IDEC	Instruction DECode
		IVA	Instr Virt Addr calc
EX	Φ1	IVA	Instr Virt Addr calc
		BCMP	Branch CoMPare
	Φ1,Φ2	ALU	Arith/Logic operation
		DVA	Data Virt Addr calc
DC	Φ1	DCR	Data Cache Read
		DTLB	Data joint-TLB read
	Φ2	LA	Load data Alignment
		DTC	D-cache tag check
WB	Φ1	RFW	Register File Write
	Φ1,Φ2	DCW	Data Cache Write

Figure 2: Pipeline activities

taken. The R4200 processor has no additional branch delay penalty for taken branches other than the defined delay slot. The performance advantage of having a single branch delay slot as opposed to two is around 8% from our simulations of the SPECint92 benchmark suite. The one instruction branch delay slot is achieved by having an extra adder in addition to the main ALU specifically for the purpose of calculating the branch target virtual address. The branch instruction determines in phase 1 of the EX stage if the branch is taken. If the branch is not taken then the PC of the instruction in the RF stage is incremented by four and used to access the instruction cache. If the branch is taken then the sign-extended 16-bit immediate field of the branch instruction is added to the PC of the instruction in the RF stage to form the branch target address.

Figure 3: Taken branch instruction shows the pipeline operation for a taken branch instruction. The instruction cache access starts in phase 2 of the IC stage; therefore, the instruction virtual address needs to be available by the end of phase 1. This allows the outcome of the branch in the EX stage to determine whether the incremented PC or the output of the branch adder is used to address the instruction cache. The branch adder and PC incrementer operate during phase 2 of the RF stage and phase 1 of the EX stage which gives one complete PClock for the adder and incrementer. This complete PClock is useful because it allows the branch adder to be designed to minimize area and power since there is no strict timing requirement for this path. The branch adder shares a significant portion of its hardware with the PC incrementer thus further minimizing the extra hardware impact of the branch adder.

The load instruction in the MIPS architecture does

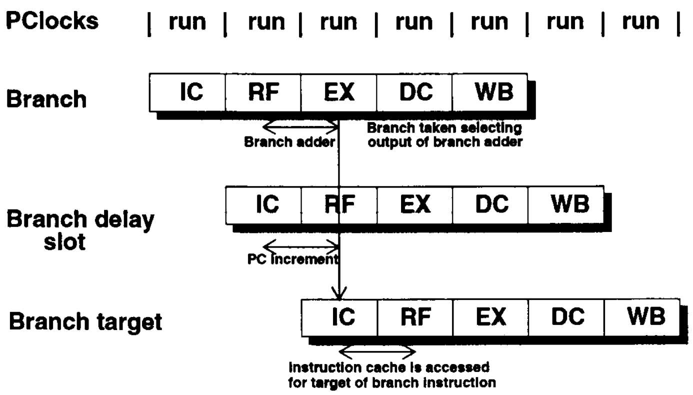


Figure 3: Taken branch instruction

not have an architecturally-defined delay slot; the instruction immediately following a load can use the contents of the loaded register. However, the target data of a load is not available until the end of the DC pipeline stage, so if the instruction in the EX stage requires this data, the hardware interlocks for one PClock. The entire pipeline stalls for one PClock while the load in the DC stage accesses the data cache and, assuming the cache hits, the data is sent to the input of the EX stage. This one-cycle stall results in a load delay of one PClock. The scheduling of this one load delay slot is desirable but not required for functional code due to the hardware interlock.

Figure 4: Scheduling of load delay slots shows the pipeline operation when the instruction using the load data (*load1 use*) is separated from the load instruction (*load1*) by one instruction and when the instruction using the load data (*load2 use*) is in the delay slot of the load instruction (*load2*)

Pipeline flow is interrupted when an interlock condition is detected or when an exception occurs.

Interlocks occur when an instruction requires more than one PClock in a particular pipeline stage to complete its operation. The interlock condition is resolved by stalling the entire pipeline. All pipeline registers hold their values until the instruction has completed execution in that stage and then the pipeline moves again by allowing the pipeline registers to load.

Exceptions occur when hardware detects a situation that requires software intervention. The hardware processes exceptions by aborting the instruction causing the exception as well as those following in the pipeline and jumping to an exception handler at a pre-defined address. When an exception is first detected the pipeline is stalled for two PClocks while the appropriate information for the exception handler is saved.

The R4200 processor ensures that during stall cycles the power dissipation is minimized by only allowing the minimum of logic needed to resolve the stall to operate. This eliminates re-accessing the caches or switching large data buses during long stall cycles.

2: Instruction and data caches

The R4200 implements a 16-Kilobyte instruction cache and an 8-Kilobyte data cache. There is no on-chip support for secondary cache. The sizes of the caches were determined from analysis of the performance of Windows NT and UNIX subject to our area and power objectives. Both caches are direct-mapped in order to minimize power, chip area and access time [5]. Simulations showed that making both the instruction and data caches two-way set associative would improve overall performance by around 7% but the cost in terms of area and power would violate our project objectives; therefore, direct-mapped caches were implemented. The overall cache and system interface organization is shown in Figure 5: R4200 cache organization.

Victim caches to eliminate conflict misses were considered but rejected. A victim cache holds cache lines recently replaced during cache misses. However, a relatively large victim cache is required to substantially improve performance, and was not considered worth the cost in terms of die area and power. Modelling showed that a fully-associative 4-line (32 words) instruction victim cache and a fully-associative 4-line (16 words) data victim cache would have given the R4200 a mere 4% performance improvement

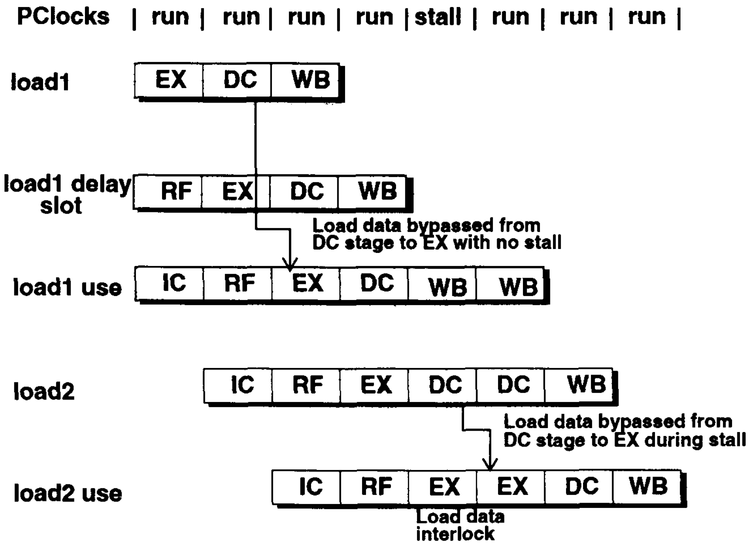


Figure 4: Scheduling of load delay slots

2.1: Instruction cache

The 16-Kilobyte instruction cache is direct-mapped, virtually-indexed, and physically-tagged for fast access: virtual-to-physical address translation occurs in parallel with cache lookup. Each cache line contains 8 instructions (32 bytes), each byte of which is parity-protected, 21 bits of physical tag, a valid bit and a parity bit covering the tag and valid bit. Cache refills are serviced without critical-word-first memory reads or early restart: the doublewords are fetched in order. As instruction misses tend to be concentrated in the sequentially-first doubleword, servicing misses out of order (critical-word-first) does not reduce average miss penalties much in instruction caches. Early restart also has little advantage in instruction caches due to the sequential nature of instruction accesses; the processor soon (potentially 2 PClocks) stalls again waiting for the second doubleword of the cache line.

Although the instruction cache can be accessed in a single processor clock cycle, the access time spans two processor pipeline stages - an instruction cache access starts in the second half of the IC pipeline stage and completes in the first half of the RF stage. This allows branches to be resolved with a single delay slot and no interlock. This is discussed in more detail in the section entitled “CPU pipeline”.

The instruction cache power consumption is reduced by two means: the introduction of an instruction buffer and the implementation of the data array in four banks. Each instruction cache access loads the instruction buffer with two instructions. Thus, during purely sequential execution, the instruction cache access rate is halved. Whenever a taken branch or jump instruction enters the EX stage, the instruction cache is accessed with the virtual address from the branch adder in the instruction address unit. The instruction cache data array is implemented in four banks, only one of which is selected by the most-significant cache index address bits. This significantly reduces the power consumption of the array.

The instruction cache achieves a hit rate of 96.4% while executing the SPECint92 benchmark suite.

2.2: Data cache

The 8-Kilobyte data cache is also direct-mapped, virtually-indexed and physically-tagged. A data cache line contains two doublewords (16 bytes) of data, each byte of which is parity-protected, 21 bits of physical tag, a valid bit, a dirty bit and a tag parity bit. The data cache implements a write-back write policy with a write-allocate scheme.

While a data cache read requires only one processor cycle, a store into the cache which hits requires two cycles. However, the pipeline is interlocked only if the store is immediately followed by another instruction requiring a data cache access, such as another store, a

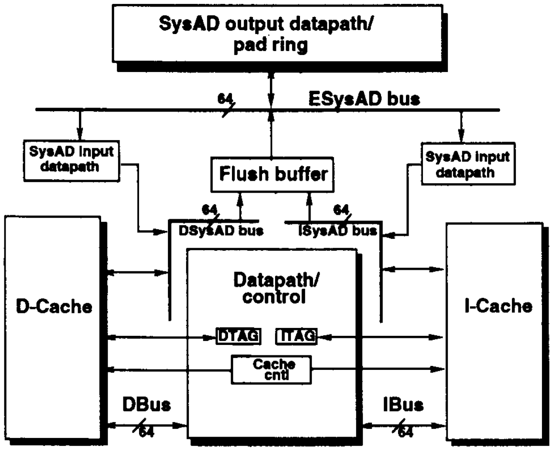


Figure 5: R4200 cache organization

load or a cache operation instruction. This reduces the effective data cache write access time to a single cycle in most instances.

Writing back data cache dirty victim lines is non-blocking - the cache line to be written back is stored into the flush buffer, and the pipeline continues while the cache line is sent out on the system interface. Implementing a write-back policy decreases switching of the I/O pads, resulting in a significant decrease in chip power.

Data cache misses are serviced using a critical-word-first, early-restart policy. The doubleword containing the desired data is requested from the system interface, and after the data is received, the pipeline is restarted while the second doubleword is being received.

2.3: Flush buffer

The flush buffer improves performance for store-intensive programs such as graphics routines. The R4200's flush buffer holds two doublewords (16 bytes) of data - an amount equal to the data cache line size. Thus, neither a dirty data cache miss nor an uncached store will stall the processor pipeline while the data is written back to memory.

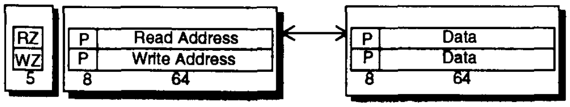
If the flush buffer is full and the processor attempts a load or store which requires external resources, the pipeline stalls until the flush buffer is empty. Figure 6: Flush buffer format shows the flush buffer format.

3: Memory-management and system-control coprocessor

The memory-management and exception-processing unit of the R4200, called the system-control coprocessor, or CP0, is similar to that of the R4000/4400. It uses a translation-lookaside buffer (TLB) to translate 64-bit virtual addresses to physical addresses. The major differences from the R4000/4400 are a decreased number of TLB entries, reduced-power mode, and a 33-bit physical address width.

The fully-associative joint instruction and data TLB (JTLB) contains 32 entries, each of which maps an even-odd pair of virtual pages. For compatibility, the R4200 supports the full range of page sizes as the R4000/4400. Although the number of JTLB entries is reduced, TLB index fields of the CP0 registers which address JTLB entries are still 6 bits wide. Only the least-significant 5 bits are used during TLB accesses, however - the most-significant bit is provided for compatibility and allows for easier expansion of the TLB in the future. The area of the JTLB and CP0 in general is of great significance since it is instrumental in determining one of the dimensions of the die. Implementing a 32-entry JTLB gave us a direct savings of 0.5 mm from the y-dimension of the die compared with implementing a 48-entry JTLB.

A 2-entry instruction micro-TLB (ITLB) is invisible to software, yet greatly reduces the demand on the JTLB, reducing power consumption and decreasing pro-



RZ Read size
WZ Write size
P Byte parity for both read/write address and data

Figure 6: Flush buffer format

cessor stall cycles. The ITLB acts as a hardware-managed cache on the JTLB. The JTLB is not accessed during a branch or jump unless the ITLB misses. This reduces power consumption at the expense of extra 3-cycle ITLB miss interlocks if the translation for the jump or branch target address is not found in the ITLB.

The smaller physical address width manifests itself in three places: the registers implementing the physical address trap feature, the registers containing the page frame numbers (PFNs) for TLB operations, and the kernel physical address space in 64-bit kernel mode. The registers containing the PFNs actually implement enough bits to hold 36-bit physical addresses, but the most-significant 3 bits of the PFN field are software-readable and -writable only and are ignored during TLB operations

A micro-TLB for data translations was considered in order to avoid driving the entire TLB on data accesses, but was rejected because the lack of locality in data references severely limits its effectiveness.

4: System interface

The system interface provides access to external system resources such as main memory, and allows access to resources on the R4200 by the system. The R4200 implements a 64-bit-wide system interface which is compatible with the system interface protocol of the R4000/4400. The system bus has been simplified by removing support for multiple processors since this feature was not determined to be critical for the target markets of the R4200. The system interface does not provide direct control of DRAM, since a DRAM interface requires more pins. While direct DRAM control decreases total system power requirements, it increases the power requirements of the R4200 due to the need to drive heavily-loaded RAS/CAS pins. This would elimi-

Table 1: Performance/implementation data

Processor	Size	Micron/Metal	Primary Cache		Price[1]	Power	MHz	SPEC92		SPECInt92	Dollars per
	(sq.mm)		Inst.	Data	Approx.	Watts	Ext/Int	Int	FP	per Watt	SPECInt92
R4000SC	184	0.8/2	8K	8K	$640	12	50/100	61.7	63.4	5.1	10.4
R4400SC	184	0.6/2	16K	16K	$1120	15	75/150	94	105	6.3	11.9
i486 DX/2 66[2]	82	0.8/3	8K (Unified)		$542	7	33/66	32.2	16.1	4.6	16.8
Pentium[2]	294	0.8/3	8K	8K	$965	16	66	64.5	56.9	4.0	15.0
PowerPC[3]	121	0.6/4	32K(Unified)		$468	9	66	60	80	6.7	7.8
MicroSPARC[4]	225	0.8/2	4K	2K	$224	4	25/50	22.8	18.4	5.7	9.8
Hobbit[5]	92	0.9/2	3K	0K	$35	0.4	20	11	-	27.5	3.2
R4200	76	0.6/3	16K	8K	$70	1.5	40/80	55	30	36.7	1.3

[1] Prices are approximate for volumes of 1000. Some prices are estimates from higher volume pricing.

[2] Performance and price taken from reference [6].

[3] Basic performance and price from reference [6] but performance is a vendor estimate and price is estimated by taking 1.25 times 20K volume price of $374.

[4] Basic performance and price from reference[7] but price is estimated by taking 1.25 times 10K volume price of $179.

[5] Dhrystone performance and price from reference[8]. SPECInt92 performance for Hobbit estimated by comparing R4200 performance of 137K Dhrystones with Hobbit's 27K Dhrystones. Price is for volumes of 10K.

nate the ability to fit the processor into a low-cost plastic package and would have prevented us from providing a R4000/R4400-pin-compatible package using the same design.

The R4200 system interface protocol was designed with the concept of a *system event* in mind. A system event refers to an event that occurs in the processor which requires access to an external system resource. Examples are instruction and data cache misses, uncached loads and stores, and actions resulting from cache operation instructions. When a system event occurs, the processor issues a request or a series of requests through the system interface to access some external resource to service the event.

Processor requests include read requests, which provide an address to an external agent, and write requests, which provide an address and a block of data to be written to an external agent.

External requests include read responses, which provide a block or a single transfer of data from an external agent in response to a processor read request., write requests, which provide an address and a doubleword of data to be written to a processor resource., and null requests, which release bus ownership to the processor.

When an external agent receives a read request, it accesses the specified resource and returns the requested data via a read response. The processor will not issue another request while a read is pending. A processor read request is complete after the last transfer of response data has been received from the external agent. A processor write request is complete after the last word of data has been transmitted. For load and store misses to dirty lines a read request is issued and, once the external agent has returned the complete cache line, the processor issues a write request for the victim line being replaced in the cache.

The processor is the default master of the system interface. An external agent becomes master of the system interface through arbitration or by default after a processor read request, and returns mastership to the processor after an external request completes.

The system interface byte order is set by the *BigEndian** pin. The byte order is big endian when this pin is low and little endian when high. The *Reverse Endian* (*RE*) bit in the system control coprocessor status register reverses the byte order in user mode.

The system interface accepts data from an external agent at a maximum data rate of one doubleword per system clock cycle (the system clock operates at half of the frequency of the processor clock). The rate at which the processor transmits data to an external agent is programmable via the *DataRate** pin. The R4200 supports two data rates: *Dxx*, for which the processor transmits one doubleword of data every three system clock cycles, and *DDx*, for which the processor transmits two doublewords of data every three system clock cycles. In order to minimize switching of the I/O pins, during *x* cycles, the processor holds the data from the previous *D* cycle.

To further minimize switching of the pins, during sub-doubleword processor writes the bytes corresponding to un-written bytes maintain the value from the preceding address cycle.

5: Performance

Table 1: Performance/implementation data lists key implementation details and performance measurements for various processors and compares them to the R4200. These numbers clearly show the excellent performance relative to cost and power dissipation of the R4200.

Acknowledgments

The authors would like to thank the entire R4200 team under the leadership of Andre Kowalczyk, Paul French, Mayank Gupta and Norman Yeung, for their efforts in bringing this project to fruition, as well as their invaluable help with this paper. We would also like to recognize the efforts of the CAD and Engineering Computer Services personnel, without whose efforts this project could not have been completed.

References

[1]: Hennessy, J., N. Jouppi, F. Baskett, and J. Gill, "MIPS: A VLSI Processor Architecture," Tech. Report 223, Computer Systems Lab., Stanford University, Stanford, Calif., 1983.

[2]: Heinrich, J., *MIPS R4000 User's Manual*, Prentice-Hall, Englewood Cliffs, N.J., 1993.

[3]: Hennessy, J., and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, San Mateo, Calif., 1990.

[4]: Mirapuri, S., M. Woodacre, and N. Vasseghi, "The MIPS R4000 Processor," *IEEE Micro*, Vol. 12, No. 2 (April), pp. 10-22, 1992.

[5]: Hill, M. D., "A case for direct-mapped caches," *Computer* Vol. 21, No. 12 (December), pp. 53-62, 1988.

[6]: Slater, M., "Motorola Begins Sampling PowerPC 601," *Microprocessor Report*, Vol. 7, No. 6 (May 10), pp. 10-11, 1993.

[7]: Case, B., "SPARC Hits Low End with TI's microSPARC," *Microprocessor Report*, Vol. 6, No.14 (October 28), pp. 11-14, 1992.

[8]: Gwennap, L., "Hobbit Enables Personal Communicators," *Microprocessor Report*, Vol. 6, No.14 (October 28), pp. 15-21, 1992.