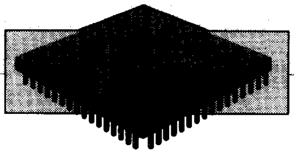


The Mips R4000 Processor

Computer architects estimate that the current generation of 32-bit machines will be obsolete by 1997. The R4000 employs a 64-bit architecture, using 64-bit registers and generating 64-bit virtual addresses. Superpipelining techniques allow it to process more instructions simultaneously than the previous generation of microprocessors. Specmark ratings indicate it performs higher than other single-chip microprocessors.

Sunil Mirapuri

Michael Woodacre

Nader Vasseghi

Mips Computer Systems

The R4000 is a highly integrated, 64-bit RISC microprocessor that provides a simple solution to the increasing demands on the size of address space, while maintaining full compatibility with previous Mips processors. Its primary features include

- on-chip CPU, FPU, MMU, primary caches, and system interface logic (See Figure 1),[1]
- superpipelining techniques,
- on-chip secondary cache control logic with a flexible interface,
- a programmable system interface for high-performance multiprocessor servers and low-cost desktop systems,
- flexible multiprocessor support, and
- 1.2 million transistors implemented in CMOS technology.

In addition, the R4000's single-chip implementation makes it easier to scale the clock as technology improves. According to SPEC benchmark tests, it achieves the highest performance of any microprocessor chip.

A 64-bit architecture

With programs growing by one-half to one bit of address space per year,[2] a greater than 32-bit address space should be useful by 1993 and required by 1997. In creating the 64-bit R4000, designers extended the R3000 architecture by increasing the data word size and virtual address space. This design entailed widening the machine registers and data paths, and sign-extending 32-bit data when loading into registers. Since certain operations work differently on 64-bit data than on sign-extended 32-bit data, we added additional instructions for 64-bit data, including integer loads, stores, adds, subtracts, shifts, multiplies, divides, and coprocessor moves.

The chip also supports a 64-bit virtual address space with wide virtual address data paths. It stores 32-bit addresses as 64-bit entities in sign-extended form and stores the results of address computation on these entities in sign-extended form. Thus it continues to support the previous 32-bit architecture's addressing.[3]

The hardware cost of extending the architecture to 64 bits was about 7 percent of the die

0272-1732/92/0400-0010$03.00 © 1992 IEEE

area. A longer, 64-bit ALU stage represents the cycle time speed penalty.

CPU pipeline

The R4000's eight pipeline stages allow it to process more instructions at once than can the R3000's five-stage pipeline.[4] Superpipelining has split the instruction and data memory references across two stages. Consequently, we could distribute the logic more evenly across pipeline stages. (See Figure 2.) The single-cycle ALU stage takes slightly more time than each of the cache access stages.

Although the superpipeline increases the cycles per instruction due to longer branch and load delays, it greatly improves the achievable cycle time. Future increases in cache size will not require a fundamental redesign of the superpipeline. We considered superscalar design as another way to increase instruction-level parallelism, but our studies showed that with current technology the chip could perform higher with a less complex superpipeline.

Figure 3 on the next page shows optimal pipeline movement, completing one instruction every internal clock cycle. The internal, or pipeline, clock rate of the R4000 is twice the external input, or master, clock frequency.

The processor accesses the instruction cache during the instruction first (IF) and instruction second (IS) stages, with a new cache access starting every cycle. The MMU translates the instruction virtual address into a physical address during these stages. The instruction bits available at the beginning of the register file (RF) stage are decoded and used to access the register file. Also at this time, the tags read from the instruction cache are compared with the physical address to determine whether the instruction cache access was a hit. If so, the instruction can advance to its execution (EX) stage. For nonmemory operations, the instruction's result is available by the end of the EX stage.

In the data first (DF) and data second (DS) stages, the R4000 accesses

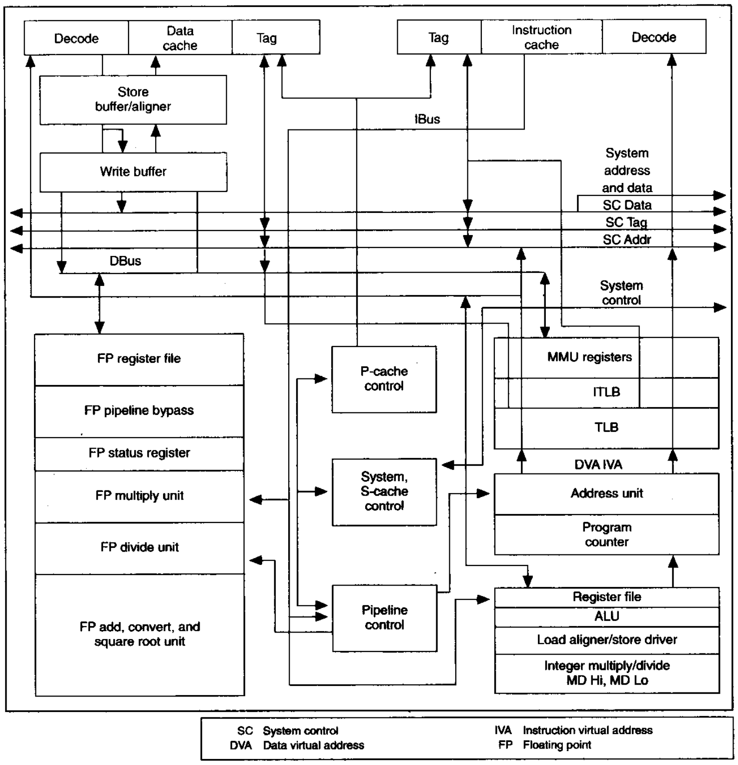


Figure 1. R4000 internal block diagram.

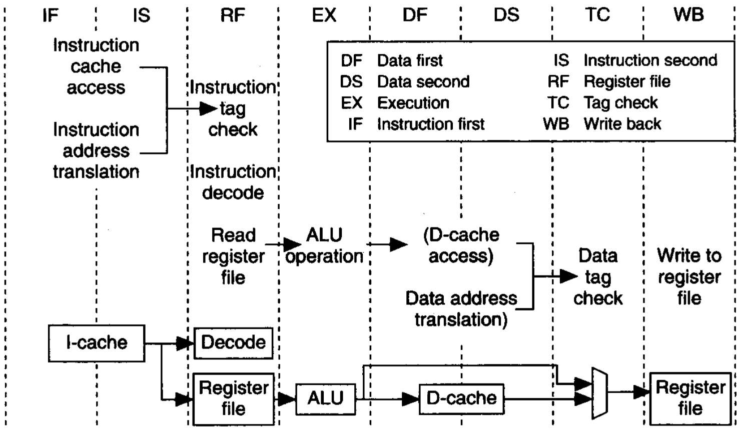
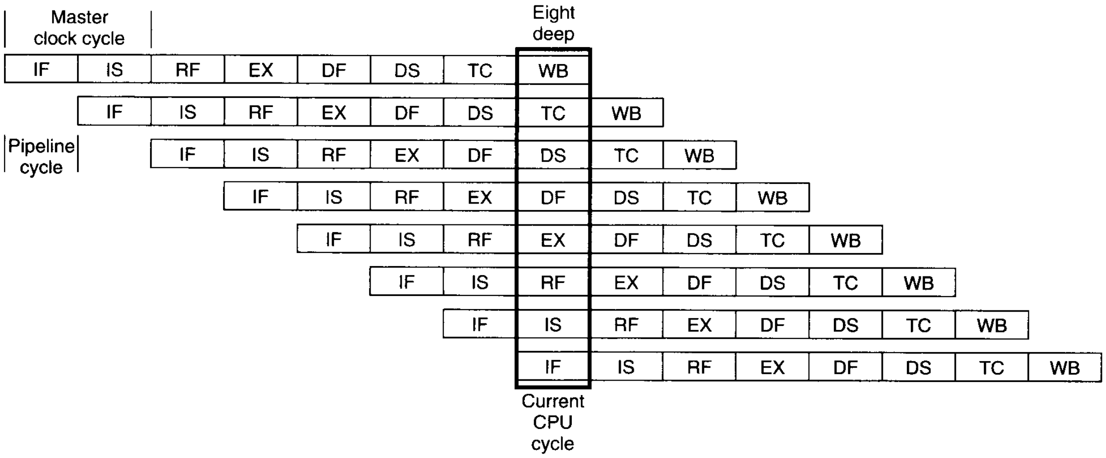


Figure 2. R4000 pipeline activities.

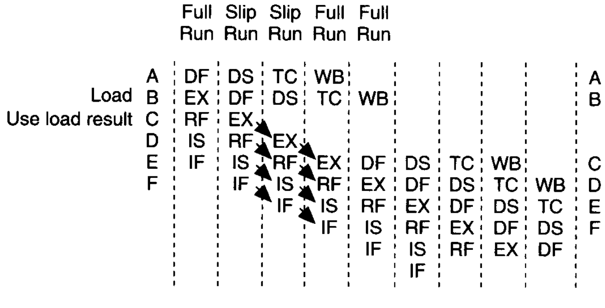


IF	IS	RF	EX	DF	DS	TC	WB							
	IF	IS	RF	EX	DF	DS	TC	WB						
		IF	IS	RF	EX	DF	DS	TC	WB					
			IF	IS	RF	EX	DF	DS	TC	WB				
				IF	IS	RF	EX	DF	DS	TC	WB			
					IF	IS	RF	EX	DF	DS	TC	WB		
						IF	IS	RF	EX	DF	DS	TC	WB	
							IF	IS	RF	EX	DF	DS	TC	WB

Figure 3. R4000 pipeline and instruction overlapping.

the data cache, with a new access starting every cycle. The MMU translates the data virtual address into a physical address during these stages. In the tag check (TC) stage, the R4000 compares the data tags from the cache tag array with the translated address to determine if the data cache access was a hit. For stores, if the tag check passes in TC, the data travel to the store buffer and enter the data cache the next time cache bandwidth is available. Instructions finally go to the write back (WB) stage where the data are written to the register file if necessary.

Load interlocks and branch instructions disrupt the normal flow of the pipeline. For loads, the data are not ready until the end of the cache access in the DS stage. If any of the two instructions after a load use the result of the load in their EX stages, the hardware interlocks and slips. As shown in Figure 4, during the slip the DF, DS, TC, WB stages of the pipeline advance while the IF, IS, RF, EX stages do not. For the load interlock, this permits the load instruction to advance and complete its cache access, while the instruction that depends on the load remains in the EX stage.



		Full Run	Slip Run	Slip Run	Full Run	Full Run					
	A	DF	DS	TC	WB						A
Load	B	EX	DF	DS	TC	WB					B
Use load result	C	RF	EX								
	D	IS	RF	EX							
	E	IF	IS	RF	EX	DF	DS	TC	WB		C
	F		IF	IS	RF	EX	DF	DS	TC	WB	D
				IF	IS	RF	EX	DF	DS	TC	E
					IF	IS	RF	EX	DF	DS	F
						IF	IS	RF	EX	DF	
							IF				

Figure 4. Load interlock/slip cycle.

The result of a branch condition check and a branch target address calculation are not known until the end of the EX stage. (See Figure 5.) By that time, up to three subsequent instructions have entered the pipeline. If the branch is not taken, the processor can continue to execute all instructions that have entered the pipeline with no penalty. If the branch is taken, the processor accesses instructions at the branch target address. For taken branches, the Mips architecture allows one instruction after the branch to complete before executing the branch target instruction. The other two instructions that have already entered the pipeline are nullified. We considered a branch target scheme that prefetches instructions from both paths of a branch, producing a smaller branch penalty. However, implementation constraints required the simpler approach without a prefetching scheme.

Results of instructions that have completed their execution, but have not yet written their results into the register file, may be bypassed as operands for subsequent instructions.

Integer data path

The R4000's 64-bit execution unit includes a 64-bit register file, load aligner, ALU, shifter, multiplier, and divider. The 64-bit data path supports extended addressing without the use of long pointers or segment registers.[5]

The ALU stage, EX, was a speed-critical path. To shorten the cycle time, the ALU comprises an adder and a logical unit. The 64-bit, carry-select adder manipulates all 32-bit operands as sign-extended, 64-bit operands. It also performs address calculations for loads, stores, and branches, and is used in integer multiply and divide.

R4000 provides hardware support for integer multiply and divide. It uses a 2-bit Booth algorithm for integer multiplication and breaks each iteration into four stages: Booth decoding, multiplicand selection, partial product generation, and product accumulation. The carry-save adder (CSA) adds intermediate partial products, and two separate 64-bit registers Hi and Lo store the final product.

The multiplier cycles at twice the pipeline clock frequency to produce two sums for each pipeline cycle. Since the R4000 uses a CSA, the multiply results are in a sum-and-carry form and must be combined through full carry propagation. The integer ALU performs this operation when the result moves to the general registers. Integer multiply latency is 10 pipeline cycles for 32-bit operations and 20 pipeline cycles for 64-bit operations.

Divides use a 1-bit-per-iteration, nonrestoring algorithm. This algorithm leaves the quotient in a signed-digit form that must be converted back to a binary representation and possibly corrected at the end of the divide. Divides use the main integer adder for the remainder add or subtract operations, thus preventing the instructions from entering the pipeline during a divide. The implementation takes two pipeline cycles per iteration; each iteration resolves 1 bit of dividend. The latencies are 69 pipeline cycles for a 32-bit divide and 133 pipeline cycles for a 64-bit divide operation. We found this performance sufficient, due to the infrequent occurrence of the integer divide operations.

The integer shifter performs immediate or variable shifts from zero to 63 places. We designed the shifter to shift up to 32 bits in one cycle, making it half the size of a 64-bit shifter. To accomplish shifts greater than 32 bits, the pipeline slips for one cycle while forcing a 32-bit shift in the EX cycle. In the next cycle, the shifter performs the remainder of the shift. A trade-off between area and performance led to this decision.

The register file is a 32-entry by 64-bit array with two read ports and one write port. It can read and write in the same cycle. In the case of reading and writing the same location in the same cycle, the R4000 provides local bypassing of the write data into the read bus.

Floating-point unit

The FPU implements the IEEE Std 754-1985.[6] Its three functional units—multiplier, adder, and divider—operate on single- and double-precision operands. While the FPU executes a multicycle operation, the CPU pipeline can continue in parallel until the FPU detects a data or resource dependency. It can transfer data directly to or from the CPU or cache memory. The FPU executes up to three instructions concurrently, one per functional unit. It retires only one instruction per cycle.[7]

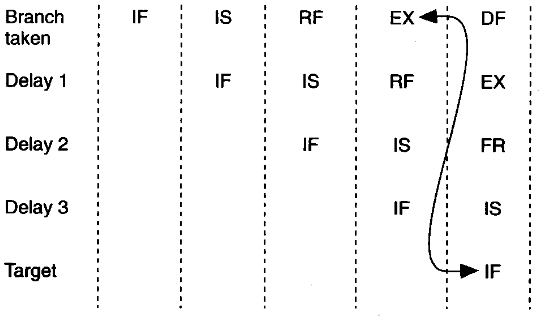


Figure 5. Branch delay.

The floating-point multiplier (see Figure 6 on next page) uses a modified Booth algorithm that scans four overlapping groups of 3 bits at once. Thus 8 bits of the multiplier operand can retire with each iteration. The mantissa portion of the multiply array uses four CSAs in a pipeline fashion. The multiplier pipeline includes four stages:

- Booth encoding and multiplicand selection,
- partial sum-and-carry generation of selected multiplicands,
- partial product summation of the previous stage result with the previous iteration result, and
- guard, round, and sticky-bit generation.

In the cycle following the last iteration of the multiply, the sum and carry from the multiplier array travel to the floating-point adder to produce the final rounded product.

The multiplier cycles at twice the pipeline clock frequency, so each iteration through the multiplier takes only half a pipeline cycle. R4000's high-speed operation demands that the multiplier array use a two-phase design approach. To reduce the clock skew in this region, the multiplier uses stronger clock drivers (with lower fanout). These drivers allow more aggressive latch designs with improved set-up times, and thus reduce overhead. All CSA and Booth multiplexers use dynamic logic design due to speed criticality.

The floating-point multiply latency is seven pipeline cycles for single-precision and eight for double-precision operations. The repeat rate is three pipeline cycles for single precision and four for double precision.

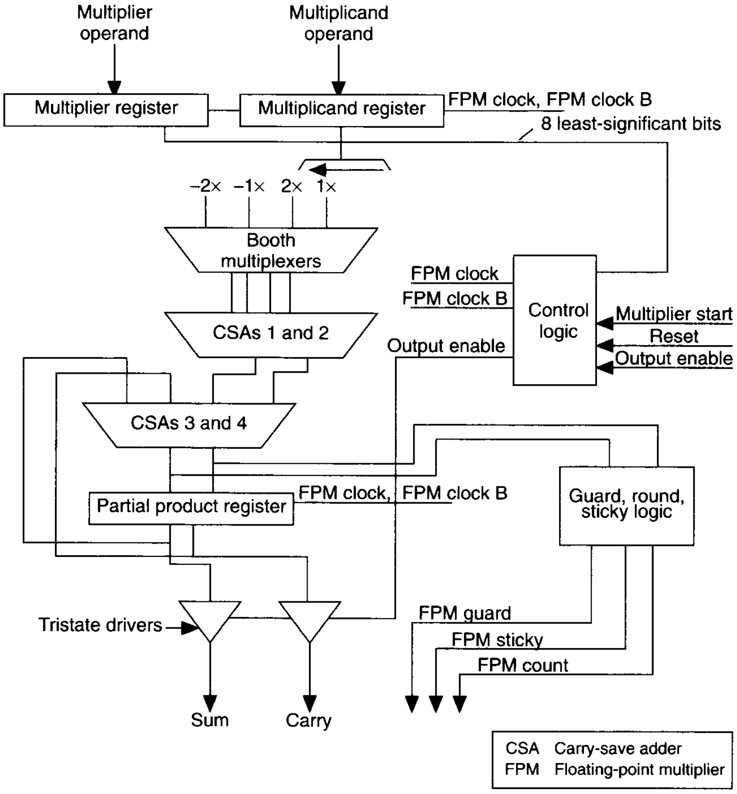


Figure 6. Block diagram of the floating-point multiplier.

The floating-point adder (Figure 7) processes one add or subtract in four pipeline cycles and starts a new operation every three pipeline cycles for both single- and double-precision operations. The adder also assists the multiplier and divider for cleanup operations, such as rounding, and final result computation.

To provide necessary bandwidth to support a two-staged, pipelined multiplier (as seen by the adder), we designed the adder to process a pair of double-precision, multiply-and-add instructions every four cycles.

The adder comprises four stages:

- unpack,
- mantissa add,
- result rounding, and
- mantissa shift (alignment/normalize).

The adder has two data entry paths. One accommodates the normal source operands that go through the unpack stage to form data inputs for all adder-supported operations. The multiplier/divider units send their intermediate results on the other path to the adder's input stage for final computation. No new instructions can enter the pipeline while the intermediate result travels from multiplier or divider to the adder for the cleanup cycles. The one data repacker in the FPU packs the final result produced by the adder to the correct data format.

We based the floating-point divide operation on the SRT divide algorithm,[8] which selects the quotient digit based on an estimation of the partial remainder. This technique has the advantage of not requiring a full-precision adder to add or subtract the partial remainder with a divisor multiple. Therefore it runs faster. The latency and repeat rates for floating-point divide operations are 23 and 22 cycles for single-precision operations and 36 and 35 cycles for double-precision operations. (See Table 1.)

The adder calculates square root by generating 1 root bit per cycle using the SRT algorithm. Since the adder also supports multiply and divide instructions, no new computational instruction may start while it calculates a square root. The square-root latency is 54 and 112 cycles for single- and double-precision operations.

Designers equipped the floating-point divider and the multiplier units with features that allow the circuit to power down at the end of every operation by recirculating zeros in the unit.

The floating-point register file is a 32-entry by 64-bit array with two read ports and two write ports. We dedicated one of the write ports for FP computational result writebacks and the other for FP load, store, and move instructions. In the case of reading and writing the same location in the same cycle, the register file locally bypasses the write data onto the read buses.

Stalls, slips, and exceptions

Pipeline hazards interrupt smooth pipeline flow (Figure 2), causing stalls, slips, or exceptions. In stall cycles, the pipeline does not advance. When the R4000 processes the stall, it restarts the pipeline and reissues several instructions to generate correct results.

For slips, such as the load interlocks detailed earlier, only the DF, DS, TC, and WB stages advance while the IF, IS, RF, and EX stages do not. When the slip condition is resolved, the instructions in the pipeline resume from whatever stage they are in. For exceptions, the processor suspends the normal sequence of instruction execution and transfers control to an exception handler, detailed later.

Figure 8 on the next page shows how the entire pipeline stalls for a data cache miss on load instruction 1. Since the load miss processing takes several cycles, the pipeline stalls until the secondary cache and main memory access completes. Note that before we got into the stall, instruction 4 may have used erroneous data in its EX stage that was bypassed from the load instruction. During the restart sequence, the processor repeats the EX stage for instruction 4 to obtain the correct data from the LOAD operation. The different stall types include

- *Data cache miss*, detected by the data tag check
- *Data first stage stalls*, which can occur for three mutually exclusive groups of instructions. 1) The pipeline stalls to resolve whether the FP instruction will cause an exception before moving on to guarantee precise exceptions. 2) The pipeline stalls to let the instruction sign extend the result. 3)The pipeline stalls to let the store buffer entries retire to memory because control logic has detected a load to the same memory location.
- *Instruction cache miss*, detected by the instruction tag check
- *Instruction translation look-aside buffer stalls*, for instruction TLB misses (explained in detail later)
- *Multiprocessor*, generated by requests from other processors

Slips occur when the result of an instruction is not available until the DS stage of an instruction, as occurs with loads. Floating-point instructions interlocked for resources also cause slips, as do integer instructions waiting for an integer multiply or divide operation to complete. Variable shifts and shifts greater than 32 bits also use slips since these operations take two cycles to complete.

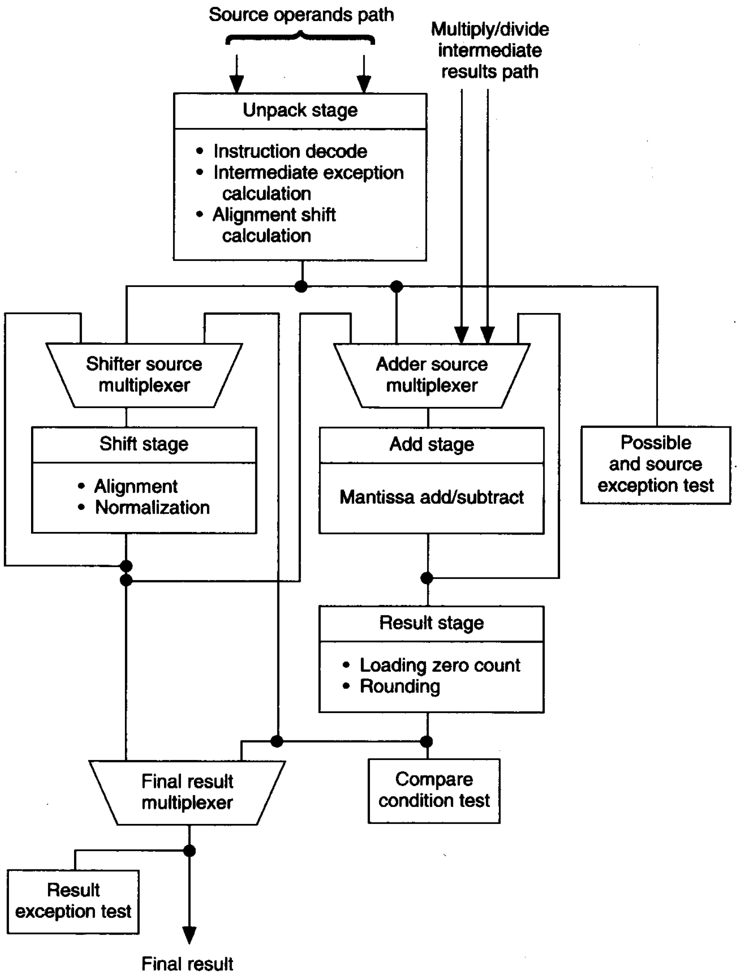


Figure 7. Adder logical block diagram.

Table 1. Integer and floating-point operation latencies and repeat rates in pipeline cycles.

	Integer		Floating point			
			Latency		Repeat	
	32 bits	64 bits	SP	DP	SP	DP
Add/subtract	1	1	4	4	3	3
Multiply	10	20	7	8	3	4
Divide	69	133	23	36	22	35

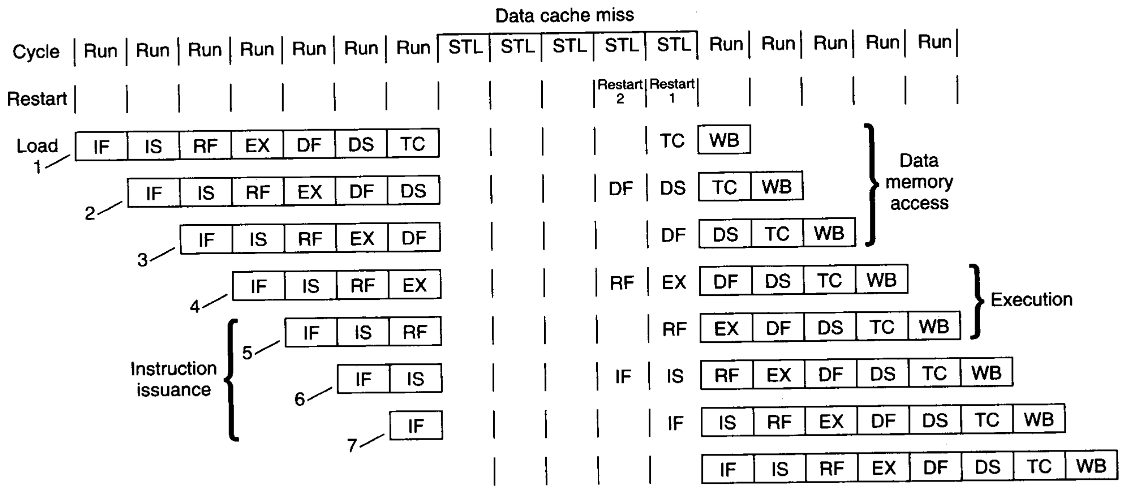


Figure 8. ADD data cache miss, use of load. STL indicates a stall.

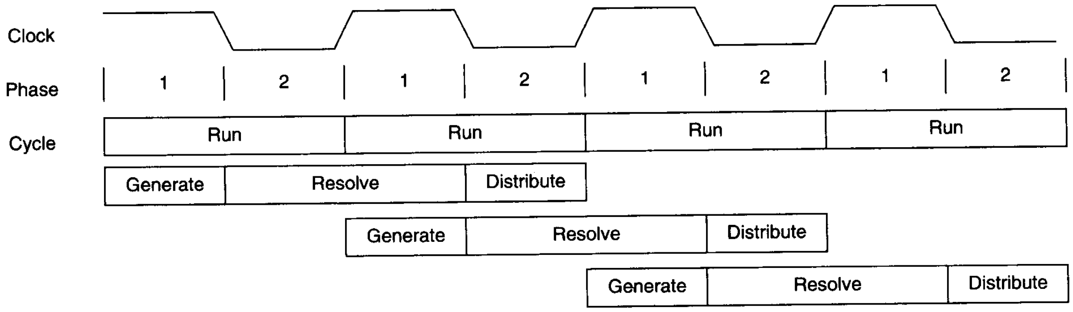


Figure 9. Circuit pipelining.

The R4000 processes many stalls and slips simultaneously. By slipping on instructions that need the same resources as a multicycle floating-point instruction, it can simultaneously accept other stall conditions from instructions that continue to advance further down the pipeline. Also, multiprocessor-initiated stalls, which can stall the pipeline to examine the cache, occur simultaneously with DCT, DFT, and ICT stalls described above.

Stall and slip implementation. The state machines that control pipeline flow (run, slip, and restart machines) operate in a pipelined fashion. When logic detects a stall or slip condition in a given cycle, the soonest the R4000 can process this condition is the end of the next cycle.

Figure 9 shows a sample timing diagram. In the first phase, the pipeline control unit evaluates logic that may generate a stall or slip condition. In phase 2 and the second phase 1, the state machines are resolved. Finally, the pipeline control signals are distributed throughout the chip during the second phase 2.

After processing a stall, the R4000 initiates a two-cycle restart sequence before the pipeline can run again. During this sequence, it reevaluates portions of the pipeline with corrected information before normal pipeline flow resumes. As shown in Figure 8, it repeats three activities: data memory access, execution, and instruction issuance.

Exception handling. The R4000 processes exceptions from sources in different pipeline stages. It prioritizes incoming exceptions and gives highest priority to the faulting instruction furthest along the pipeline. Table 2 lists different exceptions and the stages where they are signaled.

During normal processing, the R4000 nullifies pipeline stages for three reasons.

- When an exception occurs, it nullifies instructions after the faulting instruction.
- It nullifies certain instructions in branch delay slots when a branch is taken.
- When the pipeline slips, it creates a nullified instruction "bubble," as the back end of the pipeline advances and the front end does not.

After being nullified, the instruction does not commit to any state. For performance, the processor inhibits any stalls signalled by the instruction. For example, if an instruction will cause a data translation exception, which is detected at the end of the DS stage, the processor will not allow it to signal a cache miss in the TC stage.

Table 2. Exceptions.

Cycles	Exceptions
IF	–
IS	–
RF	Instruction translation
EX	Interrupt Bus error instruction Illegal instruction Breakpoint Syscall Coprocessor unusable ECC instruction Virtual coherency instruction
DF	–
DS	Overflow Floating point
TC	TLB modified Data translation
WB	Bus error data Virtual coherency data Watch NMI Reset

Memory management unit

The MMU translates virtual addresses into physical addresses using an on-chip translation look-aside buffer (TLB). It manages exceptions, controls the cache subsystem, and provides diagnostic and error recovery facilities. Compared to the R3000, the R4000 MMU provides enhanced operating system support including increased TLB entries, variable page sizes, 64-bit architecture support, supervisor privilege level, timer interrupts, and a physical address trap.

We wanted to increase the number of entries in the TLB over the 64 entries available in the R3000 since this boosts performance in a wide range of applications. Using 128 entries required too much area for the fully associative lookup circuit. Therefore, we implemented a 48-entry TLB with each entry mapping two consecutive pages and producing 96 effective entries. The TLB superpipelines in the R4000 (across the DF/DS pipeline stages) and runs in parallel with the cache access.

The instruction translation look-aside buffer (ITLB) is a two-entry, fully associative translation buffer that is a subset of the main TLB. This ITLB supports only a 4-Kbyte page size, to reduce complexity with minimum performance impact. When an instruction miss occurs in the instruction buffer, the pipeline stalls and the main TLB refills the ITLB. When a branch is taken into a different page, the branch target instruction address translation uses the TLB bandwidth available during the data first and data second stages of the branch instruction. Since the instruction first and instruction second stages of the branch target line up with the data first and data second stages of the branch instruction, the target address translation refills the ITLB without stalling the pipeline.

The R4000 implements variable page sizes on a per-page basis, varying from 4 Kbytes to 16 Mbytes. This helps to reduce thrashing of the TLB in some cases, such as in the use of a frame buffer which uses large data blocks. It implements variable page sizes by having a mask associated with each TLB entry. When addresses approach the TLB for translation, the corresponding mask bits in the TLB specify which virtual address bits participate in the comparison and translation.

The R4000 instruction set architecture supports 64-bit addressing. The current revision of the R4000 uses 40 bits of the 64-bit virtual address space. Increasing the effective virtual address size above 40 bits would have made the TLB wider than the data path and difficult to fit into the layout. Hardware explicitly checks the unused upper bits (bits 61:40) of the virtual address to make sure they are zero, ensuring a smooth transition for software as the size of the virtual address grows in future revisions. The R4000 supports a physical address of 36 bits.

The unit includes a supervisor privilege level of operation, in addition to the kernel and user levels present in previous company designs. This mode improves operating system support with more privilege levels.

A CACHE instruction provides a set of operations allowing the implementation of both a high-performance, symmetric, multiprocessing operating system and a high-performance workstation operating system. This instruction makes some tasks more efficient, including block copy, page zeroing, cache initialization, page flushing, and cache testing.

The CACHE instruction supports a number of operations including

- load and store of cache tags,
- selective invalidation of cache lines,
- create dirty exclusive data cache lines, and
- forced writeback of lines.

The R4000 provides a physical address trap feature for debugging software. This takes an exception on a reference to a selected physical address, which is specified in the Watch register.

The Count and Compare registers implement a timer interrupt service. The Count register acts as a timer, incrementing at half the pipeline clock rate. When the value in the Count register equals the value in the Compare register an interrupt occurs.

Memory hierarchy

The R4000 fits a range of system configurations. A programmable system interface permits tuning to different system specifications and exploiting future improvements in DRAM and SRAM design. The R4000 supports a two-level cache hierarchy that configures to run with different line sizes. Multiple cache coherency protocols available on the R4000 support several multiprocessor systems.[9,10]

The limited available primary cache size necessitated support for a closely coupled off-chip secondary cache required by high-end systems. We estimated the cache control section required 10 percent extra logic to support systems both with and without secondary cache. The R4000 manages its primary and secondary caches using a write-back method, in which stores send data into the caches, but the data do not write back to memory until the cache line is replaced or flushed.

The processor maintains its primary caches as a subset of the secondary cache contents. This prevents the occurrence of virtual aliases, which could lead to incorrect operation. A virtual alias occurs when multiple virtual addresses in the primary cache map to the same physical address in the secondary cache.

The primary caches are virtually indexed, so the secondary cache stores 3 bits of the virtual address (bits 14 to 12) needed to locate the primary cache lines that may contain data from a particular secondary cache line. (This virtual address information will support primary caches up to 32 Kbytes each). Because only one copy of the secondary cache line can reside in the primary cache, no two virtual addresses in the primary cache can map to the same physical location. Without this capability, R4000 would have to flush the large secondary cache to prevent aliasing. This is time consuming, especially for aliases caused by reusing pages for I/O.

Primary cache. While the initial version of R4000 uses an on-chip primary cache size of 8 Kbytes of instruction and 8 Kbytes of data, we can easily increase these sizes. The current revision supports primary caches up to 32 Kbytes each of instruction and data.

The primary cache is a direct-mapped, virtually indexed, physically tagged cache. Direct mapping makes it easy to find the location of a particular line in the cache and to manage cache consistency between the primary and secondary caches.

As the primary cache is virtually indexed, the virtual address generated by R4000's address unit looks up the cache line, while the address translation occurs in parallel. The address translation produces the physical address of the access, and the comparator compares it with the physical address read from the tag of the cache lines. The processor uses data coming out of the cache before it checks the tag, reducing the delay before load data can be used by one cycle.

Direct-mapped caches access faster than associative caches, but their hit rate is not as high as for set-associative caches. This penalty decreases as we increase the size of the primary caches. The primary caches support two software-programmable line sizes (16 and 32 bytes) that users can change independently for the instruction and data caches.

R4000 needs two cycles to access data in the primary cache, but a new address may enter every cycle. This is possible because the processor accesses the cache array in one cycle, excluding the address buffering and the data drive time. The address does not acces the array until the beginning of phase 2 of the first cycle, when the data from the previous access have been latched.

The primary instruction and data caches have separate data and tag arrays. The data cache data array and tag array may be addressed separately every cycle. During the data first and data second stages of a store instruction, the processor accesses the tag array for the store, while it may access the data array for a previous store that has passed its tag check and has data waiting in the store buffer. The two-entry store buffer decouples the data to be stored from the rest of the pipeline.

Since the architecture supports byte stores, the data cache array is arranged in eight blocks. Each block has a byte of data, a parity bit, and a redundant bit. The primary caches access 64 bits of data at a time, with the ability to write selected bytes. Row and column redundancy terms improve the die yield. To replace a defective row or column in one of the cache arrays with a redundant row or column, the manufacturer must blow the laser fuses.

Secondary cache. The secondary cache is direct mapped, physically indexed, and physically tagged. Manufacturers can build it from industry-standard static RAMs of different speeds and densities. The 128-bit-wide secondary cache interface allows a single access to the secondary cache to fill a four-word primary cache line. This cache supports a line size of four, eight, 16, or 32 words.

A physically indexed secondary cache makes multiprocessor support easy as all addresses on the system bus can be physical, eliminating the need for extra address translation information.

With R4000 supporting a maximum secondary cache size of 4 Mbytes, and with several such caches present in a multiprocessor system, the probability of a soft error demands support for error checking and correction. This ECC support for the secondary cache corrects 1-bit errors and detects 2-bit errors. R4000 performs on-chip tag correction, but it needs external hardware support to correct data errors.

We chose parity support for the primary cache since the on-chip caches are small and less prone to soft-error failure. If the operating system finds a parity error in the primary cache on a clean line, it can arrange to refill the primary cache line. When it detects a cache error, the processor takes an exception and jumps to uncached space. There the operating system examines the cache error control register, which specifies the type and location of the cache error.

One complex operation carried out in the cache logic is the write-back of dirty lines to memory. During writebacks, a state machine, the *zipper*, merges dirty (corrupted) lines in the primary cache with the data from the secondary cache as the line transfers to the system interface. The zipper checks tags in the primary instruction and data caches. It invalidates both instruction and data lines while merging any dirty data from the primary data cache. This operation completes in four pipeline cycles to match the maximum speed supported by the secondary cache.

System interface. The system interface lets the processor access external resources required to satisfy cache misses. It also allows an external agent access to some of the processor's internal resources. For multiprocessor systems, the system interface provides the processor mechanisms necessary to maintain cache coherence of shared data.

R4000 uses a 64-bit-wide system interface to increase main memory bandwidth compared with previous 32-bit system interfaces. The system interface can receive a double word every two pipeline cycles. If R4000 is operating without a secondary cache, the system interface can operate at the maximum system interface data rate, since the primary cache has a 64-bit data path that supports this rate. With a secondary cache, the maximum data rate the processor can support directly relates to the secondary cache access time. If the access takes too long, the processor cannot transmit or accept data at the maximum rate. The secondary cache only accepts· reads and writes occurring in at least four cycles. With fast static RAMs that support a four-cycle access, the secondary cache interface can keep up with data coming in from the system interface at the maximum rate. Designers can program the system interface to transmit data in a range of rates, to suit different system and secondary cache speeds.

The system interface can be programmed to be clocked by a divided-down version (divided by two, three, or four) of the internal clock frequency. The internal clock runs at twice the processor's input, or master, clock. This allows systems designed for slower versions of the R4000 to run faster versions. For example, a system designed for a 50 MHz R4000 (with the system interface programmed to halve the internal 100 MHz pipeline clock) could implement a 75 MHz R4000 with the system interface clock divisor changed to divide by three. A 75 MHz external clock generates a 150 MHz internal pipeline clock, which the divisor divides by three to produce a 50 MHz system clock.

The R4000 supports an overlapped mode of operation on the system interface when configured with a secondary cache. When a miss occurs in the secondary cache that requires a line to be written back to main memory, the system interface sends out a read request for the miss and then immediately sends out a write with the writeback data. This saves the R4000 from having to buffer up secondary cache lines before they are written back, which would use significant chip area to support the largest secondary cache line of 32 words.

Multiprocessor support. The R4000 provides mechanisms to implement a variety of cache coherency protocols that may be snoopy or directory based (see Figure 10). Designers closely coupled the multiprocessor logic with the pipeline activity to allow access to the primary caches.

The starting point for R4000's coherency model was the MESI (modified, exclusive, shared, invalid) protocol. MESI implements a four-state cache coherence protocol (the states are invalid, clean exclusive, dirty exclusive, and shared). R4000 implements a fifth, the dirty shared state (Figure 11 on the next page), which allows for efficient implementations of a semaphore given the support for update protocol. When a processor successfully acquires a semaphore by gaining a dirty shared copy of the semaphore, all the other processors using that semaphore will be updated with its new value. They don't need to generate additional transactions on the bus. With the MESI protocol, a request from another processor (that is, an intervention) can cause writebacks to the sys-

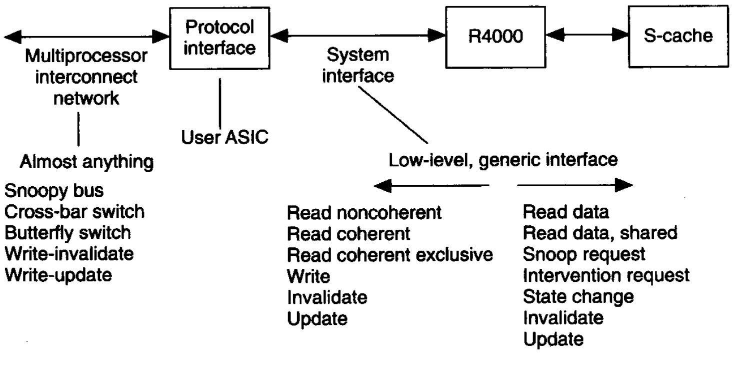


Figure 10. Multiprocessor protocols.

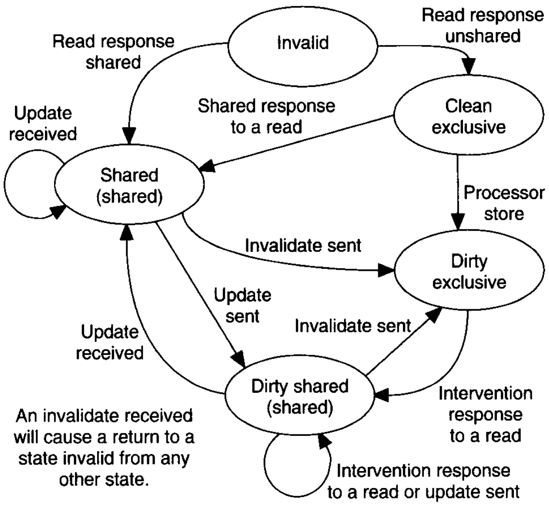


Figure 11. Cache coherency diagram.

tem memory. These writebacks place an additional burden on the system design. (The R4000 cannot process three-party transactions on interventions.) The processor stores the state of a cache line along with the tag and data for each line in the caches.

When R4000 receives an external snoop, intervention, invalidate, or update, it checks the secondary cache tag and state bits while allowing the processor to operate within the primary cache space in parallel. Misses in the secondary cache require no further action because the primary is a subset of the secondary. If an external event hits in the secondary cache, access to the primary may be required to complete the transaction. To gain access to the primary cache, the processor stalls the CPU pipeline.

The processor supports write-invalidate and write-update protocols, controlled on a per-page basis. The TLB may mark pages as uncached, noncoherent, coherent exclusive, coherent-write exclusive, and coherent-write update. Table 3 shows examples of the actions caused by these attributes.

Table 3. Examples of actions caused by coherency attributes.

Algorithm	Load-miss	Store-miss
Uncached	Word read	Word write
Noncoherent	Block read noncoherent	Block read noncoherent
Coherent exclusive	Block read exclusive	Block read exclusive
Coherent write exclusive	Block read	Block read exclusive
Coherent write updat	Block read	Block read/update

The R4000 provides a load linked and store conditional pair of instructions to provide synchronization between processors on the system bus based only on cache coherency. An example of this is the fetch-and-add operation.

```
Loop: ll     T0,0 (T1)     ;load counter, set load link bit
      addu   T0, T0, 1     ;increment
      sc     T0, 0 (T1)    ;store back if load link bit still set
      beq    T0, 0, Loop   ;retry if store failed
```

The store conditional instruction fails if the location has been invalidated or updated since the preceding load linked instruction. This mechanism can implement semaphores, bitlocks, fetch-and-add, and other synchronization mechanisms. It also guarantees that at least one processor on the bus will get the semaphore on the first attempt so deadlocks or long stalls will not occur.

Design methodology

We chose full-custom data path layout for maximum speed and the highest packing density. Designers implemented most of the control sections using a logic synthesis and optimization tool and laid them out using standard cell place-and-route methodology. However, to achieve our target cycle times, we had to custom design and lay out by hand some of the control sections in the critical paths.

We used a two-phase, zero-overlap clock strategy and distributed it throughout the chip with a balanced clock tree, to control skew. A phase-locked loop generates four times the frequency of the external input (master) clock and distributes it through the chip. Divide-by modules at the end of the clock tree generate 2x- and 1x clocks. The processor pipeline and most logic use the 2x clock, which cycles twice as fast as the master clock frequency. The integer multiplier and floating-point multiplier use the high-speed 4x clock, four times the master clock frequency.

The chip uses two types of register/latches: stacked and pass-gate dynamic. Stacked registers, used extensively, are immune to clock skew as long as there are zero or an even number of inversions between the two stacked latches. However, when a short setup time and fast clock-to-output delay were necessary, we used pass-gate dynamic latches. In these cases, a design rule enforced a delay equivalent to the time needed to pass through at least three inverters of a fan-out of three between the latches to prevent data slip-through.

We equipped the output buffers with a digitally controlled slew rate to reduce noise injected into the system buses. One buffer determines the digital control signal values for the rest of the buffers. This output buffer sends the pad a signal, which in turn feeds back into an input pad. The processor samples the round-trip delay and references it with the clock cycle. Users can program the desired amount of skew in terms of a fraction of a clock cycle. Depending on the control signals generated, the strength of the output buffer's pullup and pull downs are adjusted. (See Figure 12.)

We laid out the chip using a generic Mips design rule, so all our semiconductor partners can work from a single database. This database is based on a 1.0-μm-drawn, two-layer metal, CMOS technology. Manufacturers are producing the R4000 in 0.8μ technology.

Verification

Mips carried out a functional simulation of a register transfer level model during the development of the R4000. The RTL model executes at about 1,000 processor cycles per minute on a 20-MIPS, R3000-based Magnum workstation. Designers divided the chip into major functional blocks (CPU, FPU, MMU, caches, and system interface) and wrote directed diagnostic tests to exercise these functional units. Trace comparisons of diagnostic tests run on an instruction-level simulator and on the R4000 RTL verified compliance with our architecture. To trace all the required signals and data in the R4000 superpipeline, we added more verification logic to the R4000 RTL model so it could capture traces for comparison with the instruction-level simulator traces.

We performed extensive automatically generated random diagnostic tests, again using our instruction-level simulator for trace comparison. We wrote additional verification diagnostics to ensure that all the arcs of the state machines within the R4000 were exercised. Our designers executed R4000 diagnostics within an RTL model of a system configurable at runtime to include a secondary cache and change any of the programmable parameters that control the system interface. They booted the Unix operating system on the R4000 RTL model about six months before Mips gave the design to its manufacturing partners. It took a 50-MIPS Mips 6280 seven days of processing to reach the Unix prompt.

We verified the multiprocessor capabilities of the R4000 using a number of different simulation models. A uniprocessor RTL simulation of the R4000 checked that the R4000 could generate and process all the multiprocessor requests defined by the R4000 interface specification. We also developed a simulation environment that could support multiple R4000 processors at the RTL level. Under this environment we ran directed diagnostic tests and self-checking random tests.

Finally, we verified that the physical implementation of the R4000 matched the RTL description by generating a gate-level model from schematics. Obviously, this model ran much slower than the RTL model, and so we needed a large compute resource to run the diagnostic test suite at the gate level. In the final stages of verification we used ten 6280 machines and around thirty 20-MIPS Magnum workstations.

Testability and packaging

The R4000 implements JTAG (IEEE Std. 1149.1) boundary scan specifications, intended to provide a test capability for the interconnection between the R4000 processor, the printed circuit board, and other components on the board.

The chip comes in two package configurations. The R4000MC and R4000SC, which have the 128-bit data interface to the secondary cache, are packaged in a 447-pin lead or plastic grid array. The R4000MC supports multiprocessor systems while the R4000SC supports high-performance uniprocessor systems. The R4000PC, for desktop, low-end servers, and embedded control systems, comes in a 179-pin PGA with no secondary cache interface.

TABLE 4 LISTS SPECMARKS FOR SIMULATED RESULTS of a realistic memory system. (See next page). We simulated the CPU time and most of the important aspects of memory and heuristically added the I/O times. Correlation of simulations with R4000 systems in the lab show the simulations to be pessimistic.

Acknowledgments

The R4000 became a reality due to an enormous team effort managed by our leader, Tom Riordan. We also had

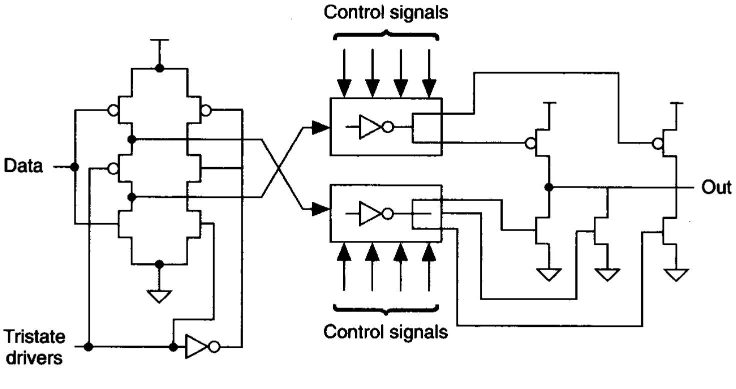


Figure 12. Output buffer.

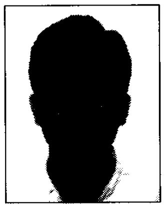
Table 4. Simulated Specmarks for a 50-MHz external-clock R4000.

	S-cache size		P-cache
Benchmark	4 Mbytes	512 Kbytes	only
Gcc	46	43	27
Espresso	54	54	38
Spice2g6	42	38	27
Doduc	49	46	33
Nasa7	56	46	43
Li	66	65	47
Eqntott	54	52	50
Matrix300	278	273	177
Fpppp	55	54	29
Tomcatv	58	59	37
Simulated SPEC	63	59	42
Simulated SPEC int	55	53	39
Simulated SPEC fp	69	64	44
CPI (simulated SPEC)	1.5	1.6	2.3

great support from other groups within Mips that had to put up with our constant demands for support. Unfortunately, we cannot list each member of the R4000 team, but we thank them all.

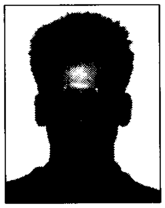
References

1. J. Hennessy et al., "Mips: A VLSI Processor Architecture," Tech. Report 223, Computer Systems Laboratory, Stanford University, Stanford, Calif.,1983.
2. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, San Mateo, Calif., 1990.
3. J.R. Mashey, "64-Bit Computing," *Byte*, Sept. 1991, pp. 135-142.
4. *MIPS R4000 Microprocessor User's Manual*, Mips Computer Systems Inc., Sunnyvale, Calif., 1991.
5. S. Waser and M. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, Holt, Rinehart, and Winston, New York, 1982.
6. *ANSI/IEEE Std. 745-1985, Standard for Binary Floating-point Arithmetic*, IEEE, New York, 1985.
7. C. Rowen, M. Johnson, and P. Ries, "The Mips R3010 Floating-Point Coprocessor," *IEEE Micro*, Vol. 8, No. 3, June 1988, pp. 53-62.
8. D.E. Atkins, "High-Radix Division Using Estimates of the Divisor and Partial Reminders," *IEEE Trans. Computers*, Vol. C-17, No. 10, Oct. 1968, pp. 925-934.
9. S.A. Przybylski, *Cache and Memory Hierarchy Design*, Morgan Kaufmann, 1990.
10. A.J. Smith, "Cache Memories," *Computing Surveys*, Vol 14, No. 3, Sept. 1982.



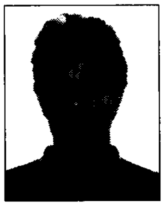
Sunil S. Mirapuri defines and designs advanced microprocessor products at Mips Computer Systems. Previously, he worked for Rolm Milspec Computers and Intel Japan. His research interests include computer architecture, digital systems, and computer programming.

Mirapuri received his BS and MS degrees in electrical engineering from Stanford University. He is a member of the IEEE Computer Society.



Michael S. Woodacre is a member of Mips' VLSI design verification team. Prior to joining Mips to work on the R4000, he was a VLSI design engineer for Inmos' transputer microprocessor team.

Woodacre received a BS in computer systems engineering from the University of Kent in Canterbury, England.



Nader Vasseghi is a member of Mips' VLSI design group. He worked on the design and development of the R4000 and now works on the next-generation RISC processor. Prior to joining Mips, he designed network controller products and high-speed programmable array logic devices at Advanced Micro Devices.

Vasseghi received his BSEE from the University of California at Santa Barbara and his MSEE from Southampton University in England. He is a member of the IEEE Computer Society.

Address questions regarding this article to Sunil Mirapuri at Mips Computer Systems, 928 Arques Ave., Sunnyvale, CA 94086; or via e-mail at sunil@mips.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 153	Medium 154	High 155