AIX

PowerPC 601 Microprocessor

By Charles R. Moore

This article describes the first member of the PowerPC processor family. The PowerPC 601 microprocessor uses state-of-the-art CMOS technology to achieve an extremely compact single-chip design. The superscalar machine organization, large internal cache, and versatile bus interface provide competitive performance at a low cost. As a result, the 601 microprocessor achieves an outstanding price/performance ratio and is well-suited for a wide range of system design points.

The PowerPC 601 microprocessor is the first member of a family of devices developed jointly by IBM and Motorola in Austin, Texas. Both companies are committed to providing a wide range of processor designs that are compliant with the PowerPC Architecture (see the IBM RISC microprocessor roadmap on page 6 of the August 1993 issue of *AIXpert*).

The 601 microprocessor project had several key goals. First, it was important that the 601 chip establish the PowerPC Architecture in the market as early as possible. Second, the processor had to offer competitive performance at a low cost. Finally, the 601 microprocessor had to be suitable for a wide range of system design points, including those requiring support for symmetric multiprocessing.

To achieve these goals, the 601 design group accepted an aggressive 12-month development schedule. The design exploited technology from both IBM and Motorola. From IBM, the RISC Single-Chip (RSC) microprocessor became the base design for the 601 microprocessor. The 601 enhanced the superscalar machine organization to achieve greater performance, and applied additional custom circuit design to reduce the die size and to allow higher frequency operation. The Motorola 88110 microprocessor bus interface was used as the development basis of the 601 bus interface.

Block Diagram

As shown in Figure 1, the 601 is a superscalar design with three pipelined execution units. The processor can dispatch up to three 32-bit instructions each cycle—one each to the Fixed-Point Execution Unit (FXU), the Floating-Point Execution Unit (FPU), and the Branch Processing Unit (BPU). The 32 KB unified cache provides a 32-bit interface to the FXU, a 64-bit interface to the FPU, and a 256-bit interface to both the instruction queue and the memory queue. The chip I/Os include a 32-bit address bus and a 64-bit data bus. In addition, the chip supports a special asynchronous serial port, the Common On-chip Processor (COP) bus, to provide advanced debugging and testing features.

Pipeline Description

The designers optimized the 601 pipeline structure (shown in Figure 2) for high performance and concurrent instruction processing in each execution unit. In general, the FXU serves as the master pipeline, managing the synchronization control required to achieve precise exceptions. The fixed-point pipeline performs all integer Arithmetic Logic Unit (ALU) operations and all processor load and store instructions, including floating-point loads and stores. In some cases, the floating-point loads and stores also progress through the floating-point pipeline. Hardware handles all hazards and dependencies. Forwarding logic allows the integer ALU operations to operate in a fully pipelined manner. There is a one-cycle penalty for a dependent operation following a load instruction.

Charles R. Moore

1993 IEEE. Reprinted with permission; from *Proceedings of COMPCON 1993*; San Francisco; February 1993; pp.109–116.

The branch instruction pipeline has only two stages. The first stage can dispatch, decode, evaluate, and, if necessary, predict the direction of a branch instruction in one cycle. On the next cycle, the resulting fetch can access new instructions from the cache. This allows the processor to react quickly to branches detected in the instruction stream and to reduce the latency of subsequent instructions.

The floating-point instruction pipeline, containing six stages, has been optimized for fully pipelined execution of single-precision operations. Although hardware supports all double-precision operations (most of which are fully pipelined), those that involve multiplication are double-pumped through the execute1 and execute2 stages of the pipe.

The 601 microprocessor contains several functional units as shown in Figure 1. The following sections describe each of these functional units.

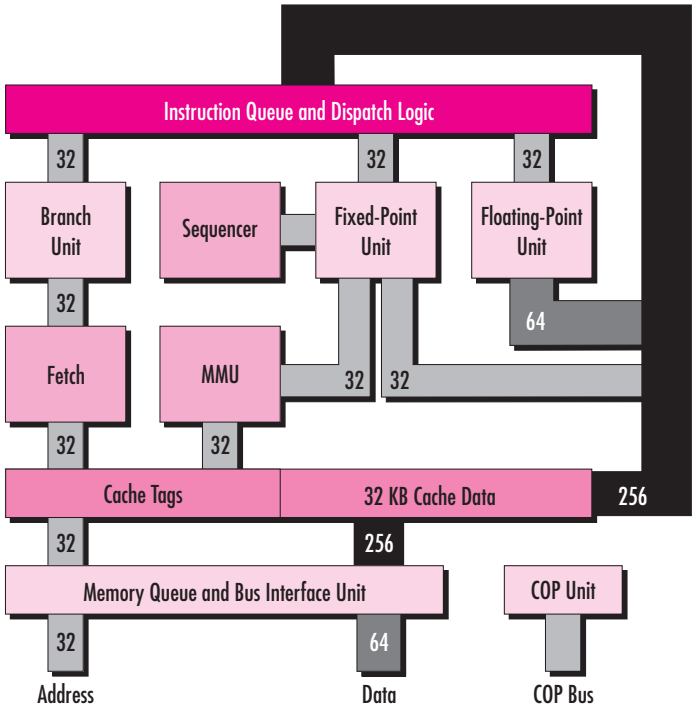

Figure 1. The PowerPC 601 microprocessor

Instruction Queue and Dispatch Unit

The 601 microprocessor has an eight-entry instruction queue for holding prefetched instructions. An eight-word bus from the cache feeds the queue. During each cycle, the dispatch logic considers the bottom four entries of the instruction queue and dispatches up to three instructions, as shown in Figure 3. There are no restrictions on instruction alignment, and the queue supports all possible shift amounts associated with the addition or removal of instructions.

The processor can dispatch branch instructions and most floating-point instructions from any of the bottom four entries of the instruction queue. The 601 microprocessor dispatches fixed-point instructions from only the bottom queue entry. The dispatch logic dispatches floating-point stores to both the FXU (for generating addresses) and the FPU (for sourcing data). The processor dispatches floating-point loads only to the FXU, but it makes the FPU aware of any dependencies. The cache forwards data directly to the FPU.

The 601 microprocessor can manage dispatches that are out of order. The processor can dispatch branch instructions and most floating-point instructions (and remove them from the instruction queue) even if an interlock exists for a preceding fixed-point instruction. These "folded" instructions are executed concurrently and do not occupy a position in the fixed-point pipeline. A unique tagging and counting mechanism preserves the program order completion of these instructions. Although out-of-order dispatch is more complex to implement, it also allows the 601 microprocessor to expose subsequent branches earlier than traditional approaches, reducing the potential dispatch stalls that could otherwise result.

Instruction Fetch Unit

The instruction fetch unit coordinates instruction fetching from the cache. Several different sources generate instruction fetch addresses. The branch processor provides the address that results from branch instructions. The sequencer unit provides addresses associated with interrupts and other synchronizing events. The instruction fetcher itself generates the next sequential address if no branch or interrupt has occurred. During each cycle, the processor selects, translates, and forwards the appropriate address to the cache arbitration logic for consideration to access the cache. The instruction queue and dispatch logic accept and process instructions fetched from the cache.

The instruction fetcher can also translate instruction fetch addresses. The Translation Shadow Array (TSA) automatically keeps track of the four most recently used instruction address translations and compares them with generating addresses of

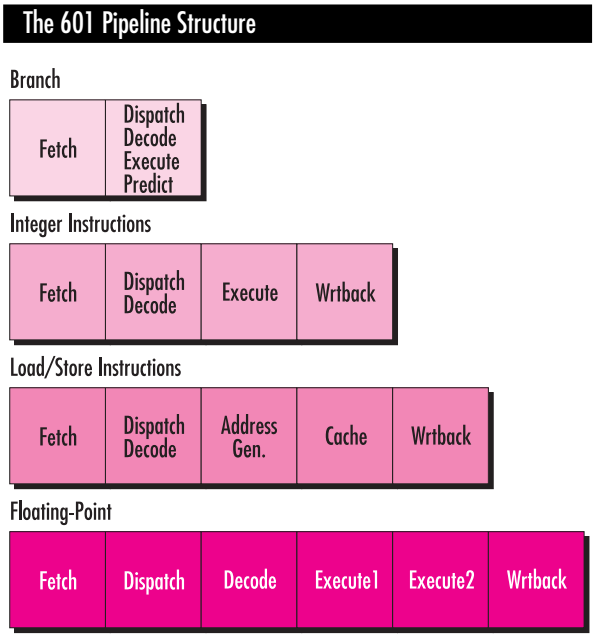

Figure 2. The 601 pipeline structure

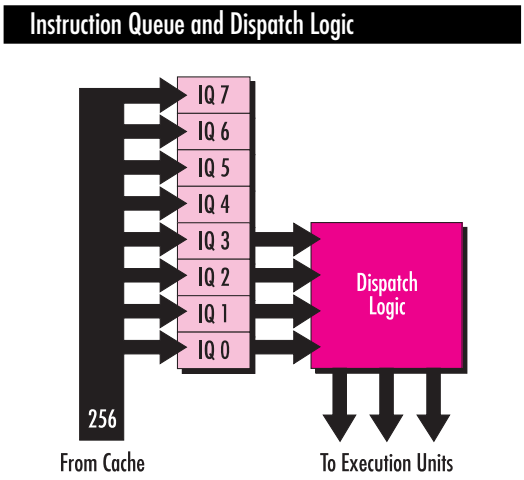

Figure 3. Instruction queue and dispatch logic

any instruction fetch. The TSA provides support for both page- and block-oriented address translations. If the TSA misses an address, the instruction fetcher arbitrates for access to the 601 microprocessor's primary memory management unit for translation. It then updates the TSA based on a Least Recently Used (LRU) replacement algorithm.

Branch Processing Unit

The BPU executes all branch instructions. In the PowerPC Architecture, branch instructions can depend on a bit in the Condition Register or on the state of the Count Register (useful in iterating loops). Alternatively, they can depend on both registers, or they can be simply unconditional. In addition, the branch target address can be absolute, relative to the program counter, or indirect from either the Link Register or the Count Register. Executing certain branch instructions requires that the Link Register automatically save the next sequential address (useful for subroutine linkages). The branch processor optimizations target the various interlocks and provide rapid response to all types of branch instructions.

The branch processor can execute branches out of order in relation to the fixed-point unit. As a result, it is possible for the BPU to alter the Link and/or Count Registers before all preceding fixed-point instructions are completed. To guarantee correct program operation when a preceding fixed-point instruction is dependent upon either the Link or Count Registers, the 601 microprocessor uses a technique called register renaming to keep the architected values of these registers synchronized with the fixed-point unit. This technique also allows the correct value of the Link and Count Registers to be restored if a preceding instruction takes an exception.

The BPU dispatches and completely executes unconditional branches (or branches that are conditional only on the Count Register) in a single cycle. As a result, these types of branch instructions generally provide zero-cycle branches; they present no delay in the instruction dispatch stream to the fixed-point or floating-point units.

Conditional branches that are dependent on the Condition Register can be either resolved or unresolved at the time of dispatch. Each of the eight fields of the Condition Register has a set of associated interlocks, which are activated by instructions that will update that field. If a conditional branch is dependent upon a non-interlocked Condition Register field, then the BPU can immediately resolve the branch. In this case, the BPU evaluates the condition based on the contents of the Condition Register, and the branch exhibits performance characteristics similar to the unconditional branches described above.

On the other hand, unresolved branches are conditional on an interlocked Condition Register field. In these cases, the 601 microprocessor employs a static branch prediction algorithm to predict the direction of the unresolved branch.

The 601 algorithm predicts that the branch will be "taken" if the displacement of the target address is negative; it predicts "not taken" if it is positive. As an aid for compiler-directed prediction, a bit in the opcode of the branch instruction allows this prediction scheme to be reversed. The processor can conditionally dispatch instructions fetched on behalf of a predicted conditional branch, but it will never execute them until it validates that the prediction was correct.

Eventually, the processor releases the Condition Register interlocks and checks the validity of the prediction. If the prediction was correct, then the branch is complete and the processor validates all prefetched instructions and makes them eligible for execution. If the prediction was incorrect, then the processor discards any prefetched instructions. Fetching then resumes down the correct path.

The 601 microprocessor employs several features to speed recovery. First, an alternate address restore mechanism saves the alternate path address and quickly redirects the instruction fetcher. Second, the fixed-point unit produces and forwards the results of compare instructions directly to the branch processor as well as the Condition Register. This permits earlier resolution of unresolved conditions.

Finally, the instruction queue implements a delayed purge mechanism. This retains any prefetched sequential instructions beyond the conditional branch until the prefetched predicted instructions are made available for loading into the instruction queue. If the 601 microprocessor resolves the condition before the predicted instructions are available, and the prefetch was incorrectly predicted down the "taken" path, then dispatch continues down the sequential stream using the remaining instructions in the instruction queue.

Fixed-Point Execution Unit

The FXU receives instructions from the instruction dispatch logic. The instructions flow through a three-stage pipeline (decode, execute, write-back), and hardware automatically handles all hazards. Hardware supports all PowerPC Architecture instructions. Most of the instructions are executed in a fully pipelined manner. Some of the arithmetic (multiply and divide) and multiple-word storage operations require several cycles in the execute stage of the pipeline.

The FXU forwards data between pipeline stages (for register dependencies) and from the cache unit (for load operations). In addition, although the differences between IBM's POWER Architecture and the PowerPC Architecture at the user level are small, the 601 microprocessor also provides hardware support for all POWER Architecture user-mode instructions.

The FXU handles the address generation portion of all load and store instructions. It also sequences the execution of the load/store multiple and the load/store string instructions. Hardware handles most forms of unaligned addresses automatically; however, some cases require multiple accesses from the cache.

The FXU also features a rapid execution of compare instructions and forwards results to the branch processing unit. This allows subsequent branches that are conditional on the result of the compare instruction to be handled efficiently.

The 601 microprocessor FPU fully pipelines most single-precision operations and most double-precision operations.

Floating-Point Execution Unit

The FPU is compliant with the IEEE® 754 standard for both single-precision and double-precision floating-point arithmetic operations. The FPU also supports the compound multiply-add operations that are defined in the PowerPC Architecture. Furthermore, hardware automatically handles all floating-point data types including denormalized numbers, NaNs, QNaNs, and infinities.

The FPU pipeline consists of four stages. The decode stage contains 32 double-word registers, the instruction decoding logic, and the main pipeline control for the FPU. The execute1 or multiply stage contains the booth encoder and Carry Save Adder (CSA) tree as well as an alignment shifter for the second operand. The execute2 or add stage accepts the sum and carry values from the previous stage and produces a single intermediate result. Finally, the writeback stage rounds the result, normalizes, and updates the register.

The 601 microprocessor FPU fully pipelines most single-precision operations (except division) and most double-precision operations (except multiplication or division). The FPU achieves a four-cycle latency and a one-cycle throughput for these operations with normal data-type operands. Double-precision operations involving multiplication require double pumping of the execute1 and execute2 stages of the pipeline and achieve a five-cycle latency and two-cycle throughput. Division uses a 2-bit non-restoring algorithm that takes 17 cycles for single precision and 31 cycles for double precision.

The FPU receives instructions from the instruction dispatch logic. In many floating-point inten-

sive applications, FPU may lag behind the full dispatch rate of the dispatch unit. To alleviate the adverse effects of this lag, the FPU maintains an additional two-entry instruction queue. The queue allows the dispatch logic to remove floating-point instructions from the primary instruction queue and expose subsequent fixed-point and branch instructions earlier.

In general, the FPU operates independently of the FXU and can execute floating-point instructions concurrently. A carefully tuned synchronization scheme allows these execution units to progress independently and still achieve the effect of precise interrupts. Although these units cooperate in executing floating-point storage access instructions, the two units can process their respective portions of these operations independently. For example, the FXU can process the addressing portion of a floating-point store operation and pass it to the cache unit without waiting for the FPU to produce the store data. The cache unit then performs the store operation after the FPU provides the data.

Memory Management Unit

The Memory Management Unit (MMU) performs the virtual-to-real address translation for load and store instructions. It also acts as a backup for the instruction-fetcher address translations that are not resolved in the instruction fetcher's Translation Shadow Array (TSA). The MMU and FXU are tightly coupled; address translation occurs during the execute phase of the FXU pipeline. The cache access occurs during the following cycle using the translated address for tag comparison. Arbitration logic determines whether the FXU or the instruction fetcher can use the MMU on a particular cycle. It is important to note that the instruction fetcher's use of the MMU should be low since the hit rate of the TSA should be high for most applications.

The MMU provides support for segment-, page-, and block-oriented translations. Briefly, address translation starts with indexing into one of 16 Segment Registers. Control bits in the Segment Register allow the process to map immediately to a real address or I/O space, or to extend the translation process onto the paging and block translation mechanism.

The PowerPC Architecture specifies a hashed-page table structure and a fixed-page size of 4 KB. The translation process includes hashing portions of the virtual address into the page table entries and subsequently searching the page table entries for matching virtual page numbers. A matching entry produces a real page number for accessing real memory. If no match is found within 16 attempts, a page-fault exception occurs. To speed up the page-based address translation mechanism, the 601 microprocessor implements a 256-entry, 2-way set-associative Translation Look-aside Buffer (TLB) cache that is used for both instructions and data. Hardware automatically handles updates to the TLB and maintenance of the reference and change bits.

Block address translation allows up to four block mappings that can range from 128 KB to 8 MB. This is useful for handling large sections of memory that are not subject to demand paging but still require address translation and protection. Using the block translation registers properly can reduce TLB thrashing and increase overall processor performance.

Cache Structure

The 601 microprocessor provides a 32 KB, eight-way set-associative, unified cache. The line size is 64 bytes; each line is split into two 32-byte sectors. The cache is indexed with a real (or physical) address and the tags are associated with the real address as well. In general, the cache implements a store-in (or copy-back) policy; however, the cache allows for page- and block-level control of cacheability, write-through, and coherency (via the MMU). The cache replacement policy is LRU. The 601 microprocessor maintains coherency on a sector basis using the four-state Modified, Exclusive, Shared, Invalid (MESI) cache coherency protocol.

In any unified cache implementation, it is important to make the most effective use of the available cache bandwidth. The 601 uses several techniques to achieve optimum efficiency. First, the cache itself presents an eight-word read interface, which at 66 MHz allows an effective 2.1 GB per second data rate. Although the processor may not use the entire eight words during every cycle, this bandwidth is especially important for sector cast-outs, snoop pushes, and instruction fetching.

Second, the cache can perform a complete read-modify-write during every cycle, which allows the 601 to complete store operations using a single cache cycle. Third, the designers developed a balanced arbitration scheme to prioritize the various cache access requests that can occur each cycle. To minimize the effects that cache arbitration stalls have on the execution units, queues for fixed-point and floating-point storage operations are located above the cache. Finally, the cache design is non-blocking, and a five-entry

memory queue sits below the cache to hold pending storage operations.

The 601 cache supports a full range of cache control operations including flush, clean, kill, touch, and allocate with zeros. These are useful for applications that must account for the performance aspects of the memory hierarchy and for systems that perform software-assisted coherency schemes.

Memory Queue

The memory queue cooperates with the cache and the bus interface unit to effectively handle operations that require access to or from the 601 bus interface. It allows the processor to decouple the cache from the bus interface, which improves the overall effectiveness of the cache.

The memory queue has a two-entry read queue and a three-entry write queue. The write queue holds up to three 32-byte sectors of dirty data. The read and write queues, and in some cases the cache itself, arbitrate for access to the 601 bus. Strict program order is not a consideration in the arbitration logic, which allows dependent read operations to precede ahead of pending write operations. This can substantially reduce the latency of storage operations that miss in the first level (L1) cache, increasing the overall processor performance. Hardware automatically maintains coherency between the memory queue and the processor, cache, and memory.

Bus Interface Unit

The Bus Interface Unit (BIU) converts operations in the memory queue into transactions on the 601 bus. The 601 bus protocols originate from the interface defined by the Motorola 88110 microprocessor chip. There are several reasons for choosing this interface. First, it provides a general-purpose framework for system-level design alternatives. Second, the framework is easily extendable, so that several 601-specific features can be added without disrupting the overall footprint of the protocols. Finally, the base technology is readily available, meets the objectives for the 601 microprocessor, and has been proven in the 88110 chip. Meeting these objectives was critically important for achieving the aggressive development cycle. It also allowed an easy migration path for some of the early users of the part.

The interface defines a 32-bit address bus and a 64-bit data bus. These buses are logically and physically decoupled from one another so that the protocol can support system bus organizations that use pipelined, non-pipelined, or even split-bus transactions. To reduce latency and increase performance, the 601 microprocessor itself can pipeline up to two outstanding operations onto the bus.

The bus interface can operate at integer multiples of the processor cycle. As a result, one can construct simple bus structures using minimal external control logic. With advanced external control, more sophisticated and higher performance structures can be created. At 66 MHz, a full-speed, pipelined system bus can sustain a data rate of 422 MB per second.

Sequencer Unit

The sequencer unit is an embedded support processor that assists the core CPU in handling many algorithmic and area-intensive functions of the PowerPC Architecture. The sequencer contains a 1 KB entry micro-code ROM, eight single-word general-purpose registers, a 32-word private RAM, and the control logic required to execute the robust 18-bit instruction set.

For example, the unit sequences the power-on reset functions that include array self-test for the cache and array initialization for all other storage cells on the chip. It maintains an on-chip Real-Time Clock (RTC) and the system decrementer function. In addition, it handles the recording and sequencing of interrupts, context synchronizing events, and errors.

Compared to the processor core, the sequencer performs the table-walk through the hashed-page table for address translations that miss in the TLB. It also performs the update of the storage access recording bits as needed. To minimize the chip area overhead, the 601 microprocessor physically implements several of the seldom-used system control registers in the RAM structure within the sequencer. As a result, the FXU passes the instructions that operate with these registers to the sequencer for assistance in execution.

Common On-chip Processor Unit

The COP is the master control logic for the Built-In Self-Test (BIST), debug, and test features of the 601 chip. A simple serial command interface allows external devices to communicate with the COP and initiate various actions. This command interface can be configured to be compatible with the IEEE standard for JTAG access to boundary scan.

Multiprocessor Features

One of the key goals of the PowerPC 601 microprocessor design was integrating features to facili-

tate symmetric multiprocessor-based systems. In general, these systems require an ability to maintain memory coherency efficiently, perform atomic memory references, and control the order in which storage operations are presented onto the interface. The following sections describe these features.

Memory Coherency

The 601 chip helps to maintain memory coherency in several ways. First, the processor performs bus snooping and adheres to a four-state MESI cache coherency protocol. To increase performance, the 601 provides a second port into the cache directory that is dedicated for snooping purposes. Dedication serves to isolate the normal processor operation during bus transactions that are not related to the contents of the processor's cache.

One of the key goals of the PowerPC 601 microprocessor design was integrating features to facilitate symmetric multiprocessor-based systems.

Second, the processor provides support for a full range of cache control operations. These instructions are useful for some operating systems (and even some applications) that require the ability to perform software-oriented memory coherency. The 601 chip also broadcasts control operations onto the bus interface, which allows the operations to be exposed to subsequent levels of the cache hierarchy as well as to other processors on the bus. On the snooping side, the 601 will detect these bus transactions and perform the required functions for the internal cache on that processor.

Finally, the 601 microprocessor allows for page- and/or block-level control of cacheability, write-through, and coherency. Software can use these controls to manage special sections of the memory space. For example, the cacheability control is useful for memory-mapped I/O regions; the write-through control is useful for simple graphics frame buffers. Selective coherency control can increase overall system performance by enforcing coherency only on the sections of memory that require it.

Atomic Storage Access and Synchronization

For atomic storage access (that is, an undisturbed read, modify, or write operation), the PowerPC Architecture defines a storage reservation mechanism that is independent of the caching mechanism. The load-and-reserve instruction performs a normal load operation and reserves the address specified by the load address. The reservation remains valid until the processor detects that the address has been written to. The store conditional instruction performs a store operation that depends on the validity of the reservation. If the reservation is not valid (another device has already written to the reserved address), the processor does not perform the store and resets a bit in the Condition Register. If the reservation is valid, the processor performs the store and sets the bit in the Condition Register. A programmer can combine these operations with a conditional branch instruction to create sequences of code that perform traditional synchronization primitives, such as compare and swap, exchange memory, and fetch and add.

TLB Coherency

The 601 microprocessor's MMU automatically maintains a TLB cache for recently used, page table-based address translations. In any system (uniprocessor or multiprocessor), the system should be able to maintain coherency between the TLB and the page tables in memory. In the 601 microprocessor, the TLB Invalidate (`tlbi`) instruction allows software to invalidate entries selectively from the TLB. As software makes changes to the page table, it can use the `tlbi` instruction to enforce coherency.

Multiprocessor systems intensify the TLB coherency problem, since several processors' TLBs could contain information that another processor is modifying in storage. To assist in handling these situations efficiently, the 601 microprocessor broadcasts the essence of the `tlbi` instruction onto the bus interface. On the snooping side, the 601 microprocessor will detect the TLBI bus operation and will perform the required function with respect to its own internal TLB. Detection can reduce the number of interprocessor interrupts in a symmetric multiprocessor system and increase the overall performance.

***Charles R. Moore**, IBM Corporation, IBM RISC System/6000 Division, 11400 Burnet Road, Austin, TX 78758. Mr. Moore is an advisory engineer currently on assignment in the Somerset Design Center working in the high-end processor development group. He participated in the development of the PowerPC microprocessor roadmap and acted as the co-lead designer of the PowerPC 601 microprocessor. Previously, he was one of the key designers of the branch unit of the original RISC System/6000 chip set and was also involved in the development of the RISC Single Chip POWER microprocessor. Mr. Moore has a BS in Electrical Engineering from Rensselaer Polytechnic Institute and an MS in Electrical Engineering from the University of Texas at Austin.*