IBM Single Chip RISC Processor (RSC)

C. R. Moore, D. M. Balser, J.S. Muhich, and R.E. East

Advanced Workstation Division
International Business Machines Corporation
Austin, Texas

Abstract

A highly integrated single chip microprocessor is described that combines a powerful RISC architecture and superscalar machine organization with system design optimizations appropriate for low cost workstation applications. The RISC Single Chip (RSC) processor is capable of dispatching up to two instructions per cycle and concurrently executing up to three instructions per cycle. The design integrates a fixed point execution unit, a floating point execution unit, an in-page branch unit, an 8Kbyte unified cache, a memory management unit, a DMA controller, an interrupt controller, ECC on the memory interface, a real time clock and decrementer, built-in self test, and a versatile engineering support processor interface on a single die. The RSC processor is the computing engine used in the IBM RISC System / 6000, model 220 (1).

Key Design Decisions

The *RISC Single Chip (RSC)* microprocessor is optimized for use in low end RISC System / 6000 workstation computer systems. These systems present a unique set of challenges to the processor design since they require high performance computational capability at a reduced overall system cost. In addition, as a member of a family of workstation products, these systems must offer full compatibility with existing family members.

To achieve the performance and compatibility objectives, the RSC has implemented IBM's POWER Architecture (2) in a manner that allows for a high degree of concurrency among operations. During any particular cycle, the RSC may be simultaneously executing a fixed point instruction, a floating point instruction (including the compound multiply-add instruction), and a branch instruction while also sequencing a DMA transfer between the Memory Bus and the RSC I/O Bus.

1. RISC System / 6000 is a registered trademark of the IBM Corporation
2. POWER Architecture is a registered trademark of the IBM Corporation.

To achieve the system cost objectives, the RSC processor integrates the entire CPU function as well as several traditionally system related functions onto a single chip. The 72 bit memory data bus connects directly to industry standard memory SIMMs. The memory address bus has been designed to allow for a simple gate array to convert the 27 bit physical address into DRAM specific control signals. Similarly, the 32 bit RSC I/O bus is designed to easily connect to a variety of buffering and bus conversion chips.

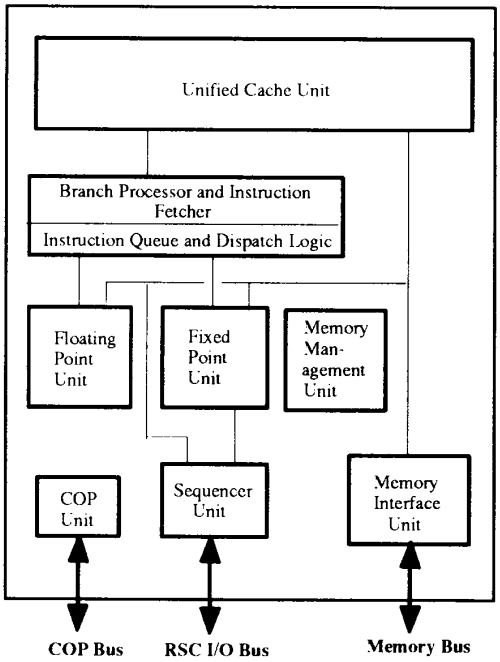

Figure 1: RSC Block Diagram

Processor Functional Units

The RSC processor may be broken down into several functional units as shown in the block diagram in Figure 1. Each of these functional units is described in the following sections.

Cache Structure

The RSC contains a two way set associative, 8 Kbyte, mixed instruction and data cache. It is managed with a store–thru policy with no reload on a store miss, and an LRU replacement algorithm. The line size is 64 bytes, and each line is sectored into four quadwords. Each quadword has a separate valid bit in the cache directory. The cache is automatically kept coherent with all I/O DMA traffic that is sequenced by the on–chip DMA processor. During each cycle, up to four words can be internally read from the cache, or up to two (consecutive) doublewords can be written to it.

One of the challenges in the design of the cache structure was balancing the accessibility to the cache from all of the possible requestors. During any particular cycle, the cache may have to arbitrate requests from the Fixed Point Unit, the Instruction Fetcher, the Sequencer Unit (on behalf of DMA operations and TLB tablewalks), and the Memory Interface Unit. A carefully tuned arbitration priority and queueing scheme allows the most important operations to proceed ahead of the others without causing significant stalls in any of the units. For example, the Fixed Point Unit is designed to tolerate a one cycle delay in accessing the cache without suffering any pipeline stall. Similarly, the Instruction Queue logic manages a total of seven instruction buffers and it can request access to the cache from several different priority levels.

Another interesting challenge in the design of the cache was the handling of store operations. After store instructions are processed by the fixed point unit, cache arbitration occurs and the cache is accessed. However, since there is not enough time to check for a cache hit and then write the cache, both the store data and the data read from the cache (in the event of a cache hit) are stored away in a four entry store queue. The memory interface unit reads entries from the store queue and writes them to memory and the cache as appropriate (that is, if the store originally hit in the cache). In addition, the modified data can be written back to the cache from the store queue as a low priority cache request in an effort to avoid the high priority request associated with the memory update. Coherency with the store queue for subsequent operations is automatically maintained in the hardware.

Branch Processing and Instruction Fetch Unit

The Branch Processing and Instruction Fetch Unit is responsible for handling branch instructions and prefetching new instructions into the Instruction Queue. One of the key challenges for this unit was to minimize the chip area required to perform the function but still provide enough instruction fetch bandwidth to achieve the overall chip performance goals. As a result, the design is optimized to effectively handle *in–page* branches and *in–page* sequential fetches (a page is 4K bytes).

The Branch Processor contains the logic necessary to coordinate the execution of all branch instructions. In–page branches that are not conditional upon the value in the Condition Register are completely handled by the Branch Processor. This includes the in–page branches that are conditional on the Count Register, which are commonly used by the POWER compiler to achieve iterative looping constructs. The direction of other conditional branches is predicted by the Branch Processor and forwarded to the Cache Unit for prefetching. The prediction is then later either validated or rejected by the Fixed Point Unit. To aid in the effectiveness of the branch prediction, the RSC employs a static prediction algorithm (predict taken if displacement is negative) that can be reversed by setting a bit in the instruction. All out of page branches are sent to the Fixed Point Unit for address generation and resolution.

Instruction fetch addresses are generated by several different sources in the RSC processor. The Branch Processor and the Fixed Point Unit cooperate to provide the address that results from branch instructions. The sequencer unit provides addresses associated with interrupts and other synchronizing events. The Instruction Fetch Unit generates the next sequential address in the event that no branch or interrupt has occurred. Each cycle, the appropriate address is selected, translated and forwarded to the cache arbitration logic for consideration to access the cache. Instructions fetched from the cache are accepted and processed by the Instruction Queue and Dispatch Logic.

For address translation, the first instruction fetch access to a particular page is forwarded through the Fixed Point Unit (FXU) to the Memory Management Unit (MMU). The result of the translation is then automatically stored in a single entry translation shadow buffer located in the Branch Processing and Instruction Fetch Unit. Subsequent instruction references within that same page can be translated very quickly through the translation shadow buffer. References outside of the page once again require assistance from the FXU and the MMU to resolve the translation and reload the translation shadow buffer.

Instruction Queue and Dispatch Logic

The Instruction Queue is internally divided into two sections: the primary instruction queue and the secondary instruction queue. The primary queue is three entries deep and is scanned by the Dispatch Logic each cycle for instructions that are eligible for dispatch. The secondary queue is four entries deep and provides additional buffering of instructions to avoid dispatch stalls when higher priority operations are arbitrated into the cache ahead of instruction fetch operations.

The Dispatch Logic processes instructions in the primary instruction queue and passes up to two instructions per cycle onto the Fixed Point Unit, the Floating Point Unit and/or the Branch Processing Unit. Floating point arithmetic operations and many branch instructions are folded directly out of the instruction queue and do not enter the fixed point pipeline. Floating point stores are dispatched to both the Fixed Point Unit (for address generation and translation) and the Floating Point Unit (for data sourcing). Floating point loads are dispatched only to the Fixed Point Unit, but the data is forwarded to the Floating Point Unit. Similarly, some branch instructions are dispatched to both the Fixed Point Unit and the Branch Processing Unit. The Dispatch Logic also assists in handling various structural hazards and synchronizing events.

Fixed Point Execution Unit

The Fixed Point Execution Unit (FXU) receives instructions from the Instruction Dispatch logic. The instructions are executed through a three stage pipeline (*decode, execute, writeback*) and all hazards are automatically interlocked by the hardware. All POWER Architecture instructions are handled in hardware, and most execute in fully pipelined manner. Some of the arithmetic and multiple word storage operations require several cycles in the execute stage of the pipeline. Full data forwarding is provided between pipeline stages (for register dependencies) and from the Memory Interface Unit (for load operations).

The Fixed Point Unit performs the address generation portion of all load and store instructions and it sequences the execution of the load/store multiple and the load/store string instructions. The hardware automatically handles both aligned and unaligned storage addresses, however in some unaligned cases, multiple cycles may be required. Requests for access to the cache can be made from either the execute stage or from the write-back stage without disrupting the flow of the pipeline.

As discussed earlier, the Fixed Point Unit also cooperates with the Branch Processor in the execution of some branch instructions. All branches and instruction fetches that reference outside of the current fetch page are executed by the FXU. In addition, all predicted branches flow through the FXU pipeline and resolve the condition during the execute stage. If the prediction was correct, the BPU is notified and execution of the prefetched instructions is permitted. If the prediction was incorrect, the FXU redirects the BPU and the prefetched instructions are purged from the queue and the pipeline. Instruction fetching resumes from the correct address. Note that the RSC never speculatively executes prefetched instructions.

Floating Point Execution Unit

The Floating Point Execution Unit (FPU) is compliant with the IEEE standard for double precision floating point operations, and all data types are automatically handled by the hardware. The pipeline is divided into 4 stages. The *decode* stage contains 32 doubleword registers, the instruction decode logic and the main pipeline control for the FPU. The *multiply* stage contains the Booth Encoder and CSA tree as well as an alignment shifter for the second operand. The *add* stage accepts the sum and carry values from the previous stage and produces a single intermediate result. Finally, the *write-back* stage performs result rounding, normalization, and the register update. All floating point operations that do not involve a multiply or a divide can operate in a fully pipelined manner.

The key challenge for the Floating Point Unit was to provide full compatibility with the POWER Floating Point Architecture with only a limited silicon area budget. This was accomplished by implementing a single precision multiply-add array, and double pumping these pipeline stages for the double precision multiply-add type operations. In addition, although floating point register renaming can offer a performance benefit in certain types of floating point applications, the technique also requires additional silicon area. As a result, the RSC implementation does not provide support for register renaming.

The Floating Point Unit (FPU) receives instructions from the instruction dispatch logic and maintains an additional two word instruction queue. This 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 concurrently execute floating point instructions. A carefully tuned synchronization scheme allows these execution units to progress relatively independent from one another, and still achieve the effect of precise interrupts. Although these units cooperate in the execution of floating point storage access instructions, they are not required to operate on these instructions in lock step. For example, the FXU can process the addressing portion of

a floating point store operation and pass it onto the Cache and Memory Interface Units without waiting for the FPU to produce the store data. The Cache Unit and the Memory Interface Unit will then automatically complete the store operation once the data has been provided by the FPU.

Memory Management Unit

The Memory Management Unit (MMU) performs the virtual to real address translation. Instruction fetch addresses that are outside of the current instruction fetch page are passed to the MMU through the FXU. Load and store addresses are generated by the FXU and require the use of the MMU. As a result, the MMU is tightly coupled to the FXU writeback stage so that address translations can occur in parallel with the cache access. In addition, as the Sequencer Unit processes DMA I/O operations, it steals access to the MMU for translation of the I/O addresses.

Address translation starts with accessing one of sixteen Segment Registers to form the 52–bit virtual address. This addressed is then hashed and indexed into the inverted page frame table in memory to form a 32–bit real address. In an effort to speed up the address translation, a 64 entry, 2 way set associative TLB cache is maintained by the RSC. This TLB is used for holding address translation information for instruction, data and I/O page addresses. Translation requests that miss in the TLB are forwarded to the Sequencer Unit for resolution. The Sequencer performs the appropriate tablewalk and automatically updates the TLB cache.

Memory Interface Unit

The Memory Interface Unit (MIU) cooperates with the Cache Unit to effectively handle operations that require access to or from memory. A memory operation queue is maintained by the MIU which is capable of holding up to two outstanding load (or fetch) operations and up to four store operations. The MIU arbitration logic selects memory operations from these queues and will allow load operations to proceed ahead of store operations. Coherency between the load and store queues is automatically maintained by the hardware. The MIU is capable of pipelining memory requests in an effort to overlap the use of the address bus and the data bus.

On reload operations, data is requested from memory in quadword blocks (corresponding to the quadword sector size) using the critical doubleword address first. As data is returned, the ECC logic checks for errors (and corrects any single bit errors) and then forwards the data onto the requesting functional unit and into a four word reload buffer. When the reload buffer is full, the MIU arbitrates for access to the cache, and the data is written into the cache. In addition, the MIU conditionally performs a *dynamic reload* of the other sectors within the cache line depending upon what other operations are qucued in either the load or the store queue

Since the RSC implements a *store–through* cache, all store operations must update the cache and be reflected back to main memory. As the store instructions are executed, the store address and store data are put into the store queue (in some cases, not during the same cycle, and the hardware automatically handles these situations). For stores that hit in the cache, the update to the cache and the update to memory are not required to occur at the same time. This allows the cache arbitration logic to optimize cache bandwidth.

For store operations that are less than a doubleword in size, the MIU automatically performs the *read–modify–write* sequence with the memory. In addition, the MIU detects the situation where multiple store operations can be combined to form a single doubleword store making more effective use of the memory bandwidth.

Sequencer Unit

The Sequencer Unit is essentially an embedded support processor that assists the core CPU in handling many of the algorithmic and area intensive functions of the chip. The Sequencer executes a robust 18 bit instruction set and contains a 3K entry microcode ROS, 16 single–word general purpose registers, a 96 word Private RAM, a 32 bit adder, and a rotate/mask function. Three levels of processing priority are defined so that the Sequencer can multitask among three different operations in progress at the same time. This allows operations that are sensitive to long latencies to be assigned at higher priority levels and to interrupt other tasks that are operating at lower priority levels.

The Sequencer effectively functions as a DMA controller for First Party and Third Party DMA operations by sequencing operations between the memory bus and the RSC I/O bus, and by performing the required tablewalks for I/O address translation. It functions as a synchronous PIO controller for programmed access to I/O space. It provides the system interrupt controller function which requires masking and prioritization of 16 interrupt requests. In addition, it sequences the power–on reset functions, assists in fetching instructions from the Boot ROM at power–on, maintains the Real Time Clock, provides the system Decrementor function, and handles the recording and sequencing of interrupts and errors.

Relative to the processor core, the sequencer performs the tablewalks through the inverted page frame table for address translations that miss in the TLB, and it performs the update of the storage access recording bits (reference and change) as appropriate. In addition, to

minimize the chip area overhead, several of the less frequently used system control registers are physically implemented in the RAM structure within the Sequencer Unit. As a result, some instructions that operate with these registers are passed from the FXU to the microcoded Sequencer for execution. In addition, synchronizing instructions, I/O access instructions, and supervisor call instructions are also handled by the Sequencer.

COP Unit

The Common On–chip Processor (COP) is the master control logic for the BIST, debug and test features of the RSC chip. A simple serial command interface allows external devices to communicate with the COP and initiate various actions.

The COP contains a *linear feedback shift register* (LFSR), a *multiple input signature register* (MISR) and the control logic required to sequence such BIST operations as DC logic self test, AC logic self test, array self test, array initiation and chip to chip wiring tests. In addition, the COP provides the capability to stop and start the internal clocks and to dump the state of all registers, RAMs and register files (via the scan strings) on the chip.

Chip and Packaging Technology

Physical characteristics of the RSC processor are summarized in Table 1. The processor is implemented in an IBM CMOS technology with .8 micron minimum feature size and three levels of metal wiring. The design is fully static and LSSD compliant. The die size of the chip is 14.9mm by 15.2mm and contains approximately one million transistors. The RSC is packaged in a 36mm ceramic pin grid array module.

Technology	.8 um CMOS
Die Size	14.9mm x 15.2mm
Transistors	1 million
Voltage	3.6 volts
Power	4 watts @ 33MHz
Package	36mm PGA
Operating Frequency	33MHz
Signal I/O	201

Table 1: Physical Characteristics of the RSC Processor

Conclusion

The RSC microprocessor successfully implements IBM's POWER Architecture onto a single chip. The machine organization of the processor allows concurrent execution of fixed point, floating point and branch instructions. In addition, by incorporating several traditionally system related functions onto the processor chip, the RSC achieves an outstanding price/performance ratio and is well suited for low–end engineering workstation applications.

Acknowledgements

The authors would like to acknowledge all of the logical design, physical design, methodology support and verification engineers that worked on the RSC processor. This project would not have been possible without the hard work and personal sacrifices made by these people. In particular, Bob Jackson and Bob Mansfield provided exceptional management focus and vision. Faradon Karim, Marty Schmookler, Tan Chu and Chris Olson provided key expertise in the implementation of the Floating Point Unit. David Barrera, Brian Vicknair and Ed Seewan designed the Cache and Memory Interface Units. Jack Wilson and Mike Vaden were the designers of the Fixed Point Unit and the Memory Management Unit. Mehdi Shahbazi designed the Branch Processor and Instruction Fetcher. John Zimmerman, Paul Greenwell and Le Ly were responsible for the Sequencer Unit and the microcode. Tim Elliott participated in the design of the FPU, the MMU and the Sequencer. Richard Billings and Robert Reese provided expertise in the design of the COP and in the test features of the chip. Jerry Lewis and Paul Harvey assisted with the simulation and timing analysis. Susan Tiner, Mike Schiffli, and Charlie Roth were key contributors to the design verification effort. Finally, Floyd Watson and PT Patel led the chip integration efforts.

References

[1] Oehler, R. R., and Groves, R. D., "IBM RISC System / 6000 Processor Architecture", *IBM Journal of Research and Development*, Vol. 34, No. 1, pp 23–36, January 1990.

[2] Bakoglu, H. B., et al, "IBM Second Generation RISC Machine Organization", *Proceedings of the 1989 International Conference on Computer Design*, 1989

[3] Grohoski, G.F., "Machine Organization of the IBM RISC System / 6000 Processor" *IBM Journal of Research and Development*, Vol. 34, No. 1, pp 37–58, January 1990.