

IBM Second-Generation RISC Machine Organization

H.B. Bakoglu, G.F. Grohoski, L.E. Thatcher, J.A. Kahle, C.R. Moore, D.P. Tuttle, W.E. Maule,
W.R. Hardell, Jr., D.A. Hicks, M. Nguyenphu, R.K. Montoye, W.T. Glover, and S. Dhawan

Advanced Workstation Division
International Business Machines Corporation
Austin, Texas

Abstract

A highly concurrent second-generation RISC computer is described that combines a powerful RISC architecture with sophisticated hardware design techniques to achieve a short cycle time and a low cycles-per-instruction (CPI) ratio. Like earlier RISC processors, this design employs a register-oriented instruction set, the CPU is hardwired rather than microcoded, and it features a pipelined implementation. Unlike earlier RISC processors, however, several advanced architectural and implementation features are employed, including separate instruction and data caches, zero-cycle branches, multiple-instruction dispatch, and simultaneous execution of fixed- and floating-point instructions. The CPU has a four-word* data bus to main memory, a four-word instruction-fetch bus from the I-cache arrays, and a two-word data bus between the D-cache and floating-point unit. These provide high instruction and data bandwidths required for high performance. The CPU has a full 64-bit floating-point engine, and thirty-two 64-bit floating point registers in addition to thirty-two 32-bit fixed-point registers. In a single cycle, *four* instructions can be executed simultaneously (a branch, a condition-register instruction, a fixed-point instruction, and a floating-point instruction). In addition, the floating-point unit has an accumulate ($A \times B + C$) instruction that executes with the same delay as a multiply or add. Counted as two instructions, this increases the peak instruction execution rate to *five*.

Key Design Decisions

This second-generation RISC design is optimized to perform well in numeric-intensive scientific applications as well as in multiuser commercial environments. A number of design choices were made specifically to ensure a high-level of sustained performance in various environments.

In order to reduce the sustained CPI ratio close to one, the peak instruction execution rate must be better than one instruction per cycle. This requires independent functional units that can execute concurrently and a high instruction bandwidth to feed them. A high level of concurrency is achieved by implementing separate branch, fixed point, and floating point units and by establishing a four-word interface to the I-cache arrays in order to be able to dispatch a maximum of four instructions per cycle. This satisfies the peak instruction demand: a branch, a

* One word is 32 bits of data

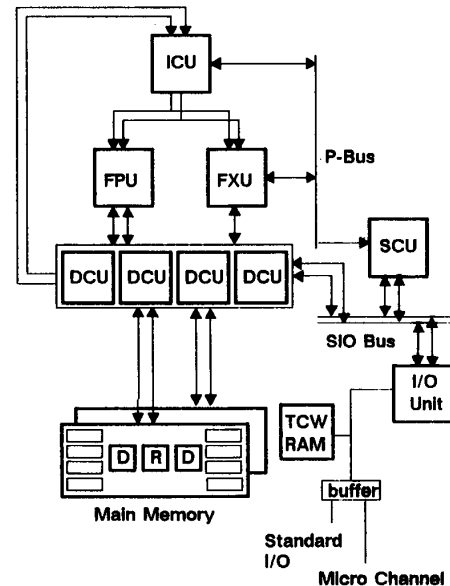


Figure 1. Central Electronics Complex

condition-register instruction, a fixed-point instruction, and a floating-point instruction. In a RISC processor, the pipeline delay penalty caused by branches can be a significant portion of the overall CPI. This is addressed by performing a sufficiently far lookahead to eliminate the branches from the instruction stream. The branch unit achieves this by taking advantage of the four-word interface to the I-cache arrays and by implementing a large number of instruction buffers. The floating point performance is advanced beyond the limits of floating-point coprocessors by designing a floating-point unit that can execute concurrently with the fixed-point unit and by dispatching two instructions per cycle to the fixed- and floating-point units. The floating-point performance is further enhanced by implementing an accumulate (multiply/add) instruction that can be executed with the same delay as a multiply or add, and by establishing a two-word interface between the floating-point unit and the D-cache.

In RISC computers the performance is extremely sensitive to the structure of the cache and memory subsystem. Accordingly, a highly optimized cache structure is designed. Separate I- and D-caches provide conflict-free access to instructions and data and allow independent

optimization of both caches. For example, a four-word interface to I-cache arrays integrated with the branch processor is essential for zero-cycle branches and multiple instruction dispatch. In addition, both caches are structured as set associative to minimize the possibility of cache thrashing. In a system with fine-tuned caches, translation becomes critical. The CPU has separate instruction and data translation-lookaside buffers (TLBs), which are also set associative. The fixed-point unit supports hardware TLB reload and page table update functions. This has significant performance advantage over managing the TLB reloads and page tables in software, which is common practice in many RISC processors. The memory interface is also carefully optimized. The memory bus is four words wide to ensure minimal interference between I-cache, D-cache, and direct memory accesses (DMA) to memory. A two-word system I/O (SIO) bus provides an interface for the I/O unit. The separate SIO bus allows independent optimization of memory and I/O buses and provides electrical isolation between the two.

Another feature that differentiates this architecture from most RISC processors is that it supports *precise interrupts*; when an instruction causes an interrupt, the pipeline is stopped before the subsequent instruction can affect the machine state. Consequently, return from an interrupt can be resumed at the interrupting instruction. The instruction set has several powerful instructions, including string operations and load/store with update. Typically these are not available in other RISC architectures. These instructions yield shorter path lengths in operating system and application code and improve performance. For ease of programmability and extensibility, 4 Petabyte (2^{52}) of virtual address and 4 Gigabyte (2^{32}) of real address space is provided.

Because of these careful design choices, this second-generation RISC computer not only attains excellent performance in industry standard benchmarks that usually deal with limited data volumes, but it also has plenty of reserve power for complex workloads with large applications or many users.

Reliability, availability, and serviceability (RAS) were also major considerations. All the chip-to-chip data buses have parity, and the memory bus has error detection and correction (ECC) and bit steering. Most of the chip data paths, including registers, register files, cache, TLB, and directory arrays also have parity. For main memory, double-bit detect single-bit correct ECC, bit steering, and memory scrubbing are implemented. ECC detects and corrects bit errors, bit steering substitutes a good spare bit for a failing bit, and memory scrubbing periodically passes the memory contents through ECC and bit steering logic to correct the single-bit soft errors before they accumulate into double-bit errors. In memory cards, bit scattering is used to prevent more than one bit in an ECC word coming from the same by-four DRAM chip. This makes it possible to correct the errors due to a single bad DRAM chip on a memory card using bit steering and still have additional error protection due to ECC. During power up, all the CPU chips go through a built-in self-test (BIST) sequence where all the logic and memory is extensively tested, and any errors are recorded.

Central Electronics Complex

The central electronics complex (Fig. 1) contains several semicustom chips: an instruction-cache unit (ICU), a fixed-point unit (FXU), a floating-point unit (FPU), four data-cache units (DCU), a storage-control unit (SCU), an input/output interface unit, and a clock chip. Every memory card contains two data multiplexing chips (D) and one control chip (R) for interleaving.

I-Cache and Branch Processing Unit

The ICU contains a two-way set associative 8 Kbyte I-cache with a line size of 64 bytes. It also has the cache directories and a 32-entry two-way set-associative TLB. The ICU processes branches, condition register instructions, and supervisor calls, and dispatches the rest of the instructions to the fixed- and floating-point units. The ICU contains the Machine State Registers and has the central control for interrupts. The instruction bandwidth requirements are satisfied by integrating the cache arrays on the same chip with the branch processing unit. Four instructions per cycle can be fetched from the I-cache arrays to the instruction buffers and dispatch unit, which can dispatch up to four instructions per cycle. Two of these are internal dispatches to ICU (branches and condition register instructions) and two are external dispatches to FXU and FPU. The instruction buffers have 12 entries. In most cases, fixed- and floating-point units receive an uninterrupted instruction stream and do not see the effect of the branches. This is referred to as *zero-cycle branches*. Usually, unconditional branches cause no delay in the pipeline. Conditional branches that are not taken (fall-through) also have no penalty because ICU dispatches the branch-not-taken path to FXU and FPU before the outcome of the branch is determined. Of course, the branch-not-taken path instructions are cancelled if the conditional branch is taken. The branch-taken path is fetched from the I-cache arrays and placed in the I-buffers but dispatch is held off until the outcome of the branch is known. Conditional branches that are taken may delay the pipeline by 0 to 3 cycles depending on how much earlier the condition register was set. The compiler tries to move condition code setting instructions far ahead of the conditional branches to minimize the conditional-branch penalty. The Condition Register (CR) of this architecture is unique; it has multiple independent condition code fields[3]. This allows several CR altering instructions to be outstanding and increases the parallelism between the functional units because each functional unit can send its condition code to different CR fields. The branch processor knows which outstanding instructions can affect the dependent portion of the CR and resolves the branch as soon as these instructions are completed.

Fixed Point Unit

The FXU decodes and executes all fixed-point instructions and floating-point load/store instructions. Both fixed- and floating-point instructions go to the I-buffers of FXU and FPU and are executed concurrently in FXU and FPU. The FXU contains the general purpose registers and the arithmetic logic unit. Register tagging is implemented to allow data cache accesses (loads and stores) to overlap with the execution of subsequent independent register-register instructions. In addition, FXU contains the segment

registers and a 128-entry two-way set-associative TLB for address translation and page protection/data locking. Page table lookups for TLB reloads and page table updates are performed by the hardware for optimal performance. In addition, hardware data locking and hardware lock granting is supported for improved performance in database and transaction processing applications[3-4]. D-cache directories and controls are in the FXU. Consequently, address generation and D-cache controls for both fixed- and floating-point load/store instructions, as well as cache operations, are performed by the FXU. Data and address of one fixed-point store instruction can be held in store buffers in FXU waiting for a convenient time to be put into the D-cache. Consequently, loads can get ahead of the stores, and the execution unit obtains the data it needs sooner. The FXU also features a fixed-point multiply/divide unit.

Floating Point Unit

Unlike typical floating-point coprocessor chips, the FPU is tightly coupled with the rest of the CPU. FPU and FXU are equal-priority and independent functional units. They receive the instructions from the ICU simultaneously and execute them concurrently. The FPU has a full 64-bit double-precision data path and executes floating point arithmetic operations (multiply, add, divide, subtract). FPU conforms to the IEEE 754 binary floating point standard. A key feature of the FPU is the *multiply/add or accumulate instruction* ($A \times B + C$). The multiply/add operation is executed with a single round and with the *same* delay as a multiply or add instruction. This reduces instruction path lengths by combining two instructions into one and enhances the floating-point performance significantly. This is true especially in numeric intensive scientific or graphics applications where matrix operations can take advantage of the multiply/add instruction. The multiply/add also has additional accuracy compared with more traditional implementations because the result of the multiply is not rounded before the add, and consequently no accuracy is lost between the multiply and add steps. Due to the full 64-bit data flow, FPU can execute a double-precision multiply, add, or multiply/add every cycle in a pipelined fashion. A two-word interface to DCU provides the required data bandwidth and loads/stores are fully overlapped with the execution of arithmetic instructions for maximum performance.

The FPU uses *register renaming* to increase the overlap of the execution of floating- and fixed-point functional units. This allows floating-point loads to be executed independently from the floating-point arithmetic operations and makes it possible to carry on loads to a target register of a floating-point arithmetic instruction while the arithmetic instruction is still going on. This is done by remapping the target register to one of the remap registers. As a result, FXU can perform floating-point loads without having to wait for previous floating-point arithmetic operations to be completed. Similarly, a four-entry *pending store queue* in FPU enables the FXU to execute floating-point stores before the FPU produces the data. This allows the FXU to generate the address, initiate TLB or cache reload sequences, and check for data protection for a floating-point store instruction, and then continue executing the subsequent instructions without being held back by the

FPU. It also allows the FPU to read data from the DCU before writing the data of a previous store instruction in DCU. This allows loads to get ahead of the stores and reduces the possibility of the floating-point execution unit starving for data. The FPU has thirty-two 64-bit floating-point registers, six rename registers, and two divide registers. It features a leading zero anticipator to avoid the full delay of a leading zero detector and to ease the overlap of multiplication and addition.

Data Cache Unit

The D-cache is a four-way set associative 64 Kbyte cache divided into four identical DCU chips of 16 Kbytes each. The cache-line size is 128 bytes, and the cache is implemented as a store-back cache to minimize the memory bus traffic. (When data is stored in the D-cache, it is not sent to memory. The data is written into main memory only when a dirty line in the cache is replaced.) The four DCUs form a four-word interface to memory, a two-word interface to FPU, a single-word interface to FXU, a two-word interface to ICU, and a two-word interface to I/O unit. DCUs support bit steering and ECC for load/store, I-cache reload, DMA, and memory scrub operations. DCUs also contain temporary storage buffers for I-cache reloads and DMA to account for the width difference between the memory bus and I-cache reload and SIO buses. D-cache directories, status arrays, and D-TLBs are in the FXU.

The performance advantages of a custom-designed D-cache come mainly from the *Cache Reload Buffers* (CRB) and *Store Back Buffers* (SBB). A 128-byte CRB divided across four DCUs receives data from memory, FXU, or FPU. Unlike simpler cache implementations that do not have a CRB, the CPU does not have to wait for the entire cache line to be brought from main memory before it can access the cache arrays. In this design, the line is brought from the memory and stored in the CRB such that the first packet contains the data that caused the cache miss. A fast load-through path that bypasses the cache arrays and CRB is provided from the memory bus to FXU/FPU to minimize the cache-miss penalty for loads. This bypass is done in parallel with error detection, and any errors are signaled to FXU/FPU at the beginning of the next cycle. Consequently, ECC comes at minimal performance penalty. The cache line is loaded into the CRB in eight cycles but the cache arrays are not tied up during the reload sequence. Any subsequent load/store can read/write data from the cache arrays or CRB before the data in the CRB is loaded into the arrays. If the data were being written into the arrays as they arrive from memory, subsequent loads and stores would not be able to access the cache arrays and would have to wait for the completion of the reload. In this design, the entire line is loaded into the cache arrays later when the cache is not being used.

Store-back buffers are also 128 bytes. They can accept data from D-cache arrays or directly from CRB and pass it to main memory. SBB improves the performance because, with an SBB, the dirty line does not have to be written back to the memory before the new line is brought into the cache. In addition, the data cache arrays are not kept busy during the whole store-back sequence. The entire line is parallel loaded into the SBB in two cycles, and the store-back data is sent to memory after the new line is brought back from memory. The DCUs can service the

CPU during cache reload and store back sequences because the cache arrays are freed up by the CRB and SBB. In addition, the store-back data can be left in the SBB and stored back later if a higher priority memory access is pending. Without CRB and SBB, D-cache line size would be limited to a shorter length and would require larger directories. Longer cache lines minimize the cache directory area and make it possible to fit them in the FXU. Longer cache lines also provide a prefetch-like effect and improve the hit ratios. In addition, a longer cache line makes it possible to overlap the memory access latency (leading edge) with the previous data transfer (trailing edge) and enables a peak memory bus utilization of 100%.

For increased reliability and availability, the DCU supports memory scrubbing, double-bit detect single-bit correct ECC, and bit steering. Memory scrubbing is performed by periodically reading the memory locations, passing them through the ECC and bit steering logic, and writing them back to memory when a correctable error is detected. In this way soft errors can be corrected before too many of them can accumulate and become uncorrectable. When a correctable error is detected during scrubbing, the sequence is repeated to determine whether the error was soft or hard. Bit steering logic in the DCU can be programmed to bypass a hard failure in memory by substituting a good spare bit for a failing bit. Bit steering is reprogrammed during every system power up. The combination of memory scrubbing, ECC, bit steering, and bit scattering minimizes the possibility of encountering an uncorrectable memory error.

Storage Control Unit

SCU is the central system controller. All the communication between CPU (ICU, FXU, DCU), main memory, and I/O is arbitrated by the SCU. CPU sends I-cache reload, D-cache reload, and D-cache store-back requests to SCU over the P-bus, and SCU generates the appropriate memory control signals. SCU is the bus master for the memory and SIO buses. It controls the interface between D-cache and system memory and oversees DMA operations between main memory and the I/O unit. The SCU provides a data path for I/O loads and stores between the CPU and I/O unit via the P-bus and SIO bus and forms an interface to the IPL ROS. The memory scrubbing is controlled by SCU, and memory errors detected by DCU are recorded by SCU. The SCU contains the Bank Configuration Registers, which indicate the size and starting point of each bank of storage in system memory.

Memory Cards

The memory cards implement a four-way interleaved design in order to provide two words of data every machine cycle from each memory card. A minimum of two memory cards is required to support the four-word memory bus to CPU (Fig. 1). The interleaving is performed by two data multiplexing chips (D) and one control chip (R) on the memory cards. The memory cards accept generic read/write instructions and generate the required read/write, refresh, and page mode read/write signals for a variety of DRAMs. The memory cards can buffer up to four instructions and 16 words of write data. Using standard modular SIMM packages for the DRAMs provides flexibility in memory capacity and cost. The memory cards can support both 1 Mbit and 4 Mbit DRAMs. Bit scattering is implemented to

guarantee that no more than one bit of a 40-bit ECC word is stored in an individual DRAM. Accordingly, the bit steering circuitry in DCU can detect and correct errors caused by a single bad by-four DRAM chip and still have additional error protection due to ECC.

I/O Unit

The I/O unit contains an I/O channel controller (IOCC) which generates an enhanced Micro Channel interface. The data interface between CPU/system memory and the I/O unit is via the two-word SIO bus. Micro Channel has a one-word address bus and a one-word data bus. The IOCC supports an I/O architecture geared for performance, robustness, and error recoverability. The main function of the IOCC is to transfer data between system memory and adapters on the Micro Channel. The CPU can transfer data to/from adapters using I/O load and store operations, and adapters can transfer data to/from system memory using DMA. The IOCC supports both DMA bus masters and DMA slaves. All data transfers support address protection mechanisms to provide data security. Up to 15 DMA channels and 16 levels of interrupts are supported by the IOCC. The Micro Channel is enhanced by adding *streaming data*, address and data parity, and synchronous exception reporting (I/O loads/stores cause precise interrupts like regular loads/stores). With the new streaming data mode, multiple data cycles can be transferred within one bus envelope. This amortizes device selection overhead across the entire packet and nearly doubles the performance for large data bursts. Precise I/O load/store interrupts improve error recoverability. The enhanced Micro Channel is fully compatible with the Micro Channel—any standard Micro Channel adapter can be attached to the enhanced Micro Channel and vice versa.

Cost-Reduced CPU

One of the goals of this design was to use a common chip set to produce a family of processors with varying cost and performance. This was accomplished by designing FXU, DCU, and SCU such that they can operate with two DCUs as well as with four. This system configuration is illustrated in Fig. 2. Because the chips are common between the two versions, they can be sorted such that faster chips are used in the high-end and slower ones in the low-end configurations. In this way, sorting for high speed is achieved without sacrificing the overall yield. This configuration has a lower cost for two reasons. First, it has fewer chips (only two data cache chips rather than four). Second, it requires a minimum of one memory card rather than two. (This is because two DCUs in the cost-reduced CPU have a two-word memory interface compared with four DCUs in the full-size CPU which have a four-word interface.) To accommodate for the smaller cache size and narrower memory bus width, the D-cache line size is reduced to 64 bytes. In the cost-reduced CPU, fixed- and floating-point data buses are dotted together. In addition, DCU sends the data to reload the I-cache over the SIO bus rather than having a dedicated I-cache reload bus to ICU.

Chip and Packaging Technologies

Physical attributes of the chip set are summarized in Table 1. The chips are implemented in a CMOS technology,

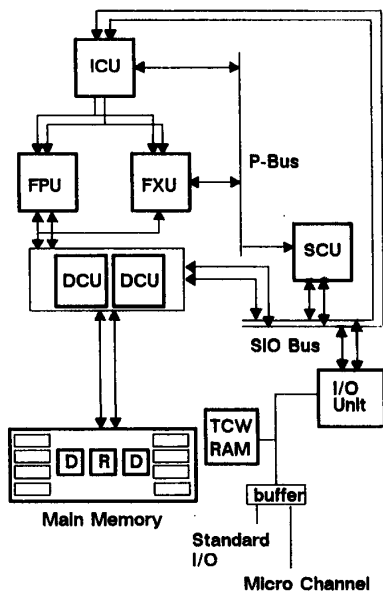


Figure 2. Cost-Reduced CPU

which has 1 micron minimum feature sizes, 0.9 micron effective channel lengths, one level of polysilicon, and three levels of metal wiring. Maximum die size is 12.7 mm. Chips are placed on individual ceramic pin grid array modules, which can support a maximum of 300 signal pins or 256 signal pins, and 4 Watt of power dissipation. The pins on the regular grid are 100 mil spaced. Additional pins are placed at the interstitial points to support the required pin count. A large number of power/ground pins are provided to minimize the simultaneous switching noise. The chip carriers are mounted on an advanced mixed-grid card by through-hole technology. The card supports holes drilled on demand on a 50 mil spaced grid. There are four signal and four power/ground planes (four power/ground planes in the middle, and two signal planes on both sides).

Conclusion

One of the main goals of this second-generation RISC project was to design a high-performance and truly balanced machine that avoided bottlenecks in the CPU, caches, memory interface, and I/O subsystem. This was achieved by implementing

- multiple functional units that can execute concurrently
- a highly fine-tuned cache and memory subsystem
- a high bandwidth I/O subsystem

In addition, throughout the system, all the components are designed to meet a high reliability, availability, and serviceability criteria in the tradition of IBM.

Table 1. Physical Characteristics of the Chip Set

Chip	Transistor Count		Die Size (mmxmm)	Signal Pin Count
	Logic	Memory		
ICU	200,000	550,000	12.7x12.7	252
FXU	250,000	250,000	12.7x12.7	256
FPU	360,000	60,000	12.7x12.7	224
DCUx4	700,000	3,800,000	11.3x11.3	184
SCU	230,000	—	11.3x11.3	255
I/O Unit	300,000	200,000	12.7x12.7	293
Total	2,040,000	4,860,000	1284 mm ²	

Acknowledgments

The key concepts and vision behind this second-generation RISC hardware architecture and machine organization were conceived at the IBM Thomas J. Watson Research Center under the leadership of John Cocke and Andy Heller. In addition this machine would not have been possible without the dedication of the Austin team, including logic design, physical design, design tools, simulation, verification, and test groups. The hardware and manufacturing technology was provided by various IBM locations, including Burlington, Endicott, and Austin.

References

- [1] G. Radin, "The 801 Minicomputer," *IBM Journal of Research and Development*, vol. 27, no. 3, pp. 237-246, May 1983.
- [2] P.D. Hester and R.O. Simpson, "The IBM RT PC ROMP Processor and Memory Management Unit Architecture," *IBM Systems Journal*, vol. 26, no. 4, pp. 246-360, 1987.
- [3] R. Groves and R. Oehler, "An IBM Post-RISC Processor Architecture," *This Proceedings*
- [4] A. Chang and M.F. Mergen, "801 Storage: Architecture and Programming," *ACM Transactions on Computer Systems*, vol. 6, no. 1, pp. 28-50, Feb. 1988.