AIX

PowerPC: A High-Performance Architecture with a History

By Ed Silha

This article provides an overview of the PowerPC Architecture and compares it to the POWER Architecture, which forms the base of the RISC System/6000 product line.

With the POWER Architecture as a base, in early 1991 a group of technical leaders from Apple, Motorola, and IBM began developing a sleeker and cleaner architecture with expanded functionality. The resulting PowerPC Architecture allows implementation from low-cost controllers through high-performance processors. It allows processors to be implemented for desktop and notebook systems, yet it has features to efficiently implement processors in multiprocessor systems.

The POWER Architecture

The RISC System/6000 product line is based on the POWER Architecture, a second-generation RISC architecture. Fixed-length (four bytes) POWER Architecture instructions have consistent formats for simple instruction decoding. Load and store instructions provide all accesses to memory. The architecture has a set of General-Purpose Registers (GPRs) for fixed-point computation, including the computation of memory addresses. It provides a separate set of Floating-Point Registers (FPRs) for floating-point computation. All computations retrieve source operands from one register set and place results in the same register set. Most instructions perform one simple operation.

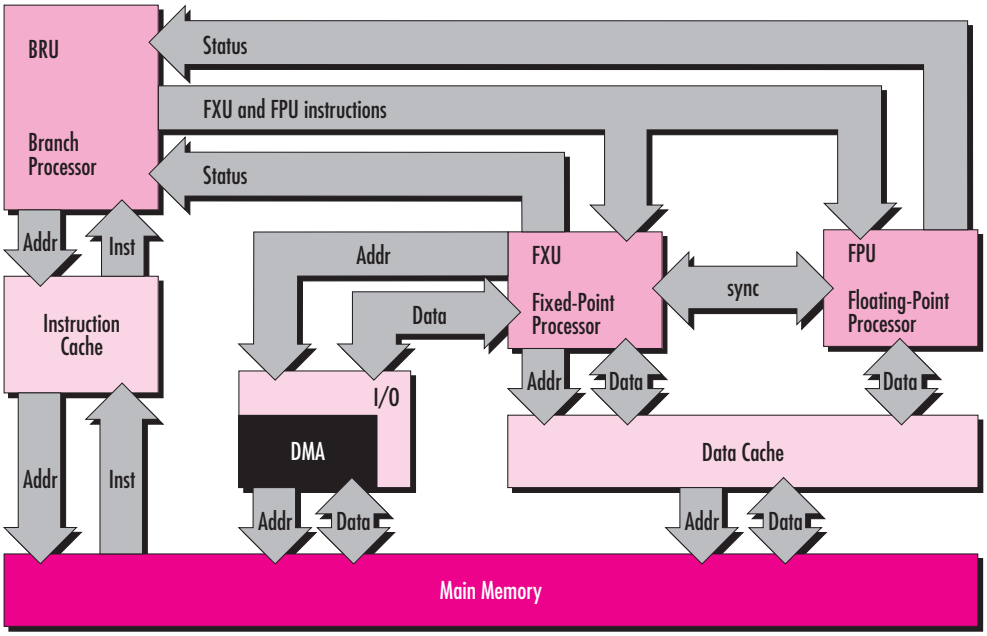
The POWER Architecture, introduced in 1990, was unique among RISC architectures because it was functionally partitioned. This partitioning separated the functions of program-flow control, fixed-point computation, and floating-point computation. It also enabled superscalar designs in which multiple functional units concurrently executed independent instructions. Figure 1 shows the organization of the functional units and how they interact while executing independent instructions.

The POWER Architecture was unique in another way. The primary objective of other RISC architectures at the time was to be simple enough for implementations to have a short cycle time. Processors could execute instructions at the fastest possible clock rate. Designers of the POWER Architecture chose to minimize the total time required to complete a task. The total time is the product of three components: path length, number of cycles needed to complete an instruction, and cycle time.

Loads and stores account for 20% to 30% of the instructions executed by most programs. In addition, many applications manipulate arrays, for which the pattern of memory accesses is often regular (such as every n element). Based on these observations, the designers included update forms of most load and store instructions. These update forms perform the memory access and place the updated address in the base address register. Using these instructions avoids computing a separate address after each access.

In addition, a common operation in many floating-point computational algorithms is computing the sum of a series of products. In most architectures, this function is performed with n multiply instructions and n-1 addition instruc-

This article is reprinted with permission from Prentice-Hall (edited for *AIXpert*) from *PowerPC and POWER2: Technical Aspects of the IBM RISC System/6000*, edited by Steven W. White and John R. Reysa. Forthcoming in late Spring 1994 by PTR Prentice-Hall.



BRU — Fetch Instructions, execute branch and CR instructions, and dispatch FXU and FPU instructions.
FXU — Execute fixed-point instructions and compute addresses for floating-point loads and stores.
FPU — Execute floating-point instructions and manage data for floating-point loads and stores.
I/O — Definition is outside the scope of the processor architecture.

Figure 1. POWER and PowerPC architectures

tions. This observation prompted the inclusion of the floating-point multiply-add instruction, which performs the $2n$-1 floating-point operations using only n instructions.

The Evolving Architecture

The first POWER implementations provided exceptional performance. The processor complex (including a memory controller, instruction cache, and data cache) consisted of seven or nine chips, depending on the system model. The seven-chip complex provided a 32 KB data cache, and the nine-chip complex provided a 64 KB data cache. These systems contained an 8 KB instruction cache that was expanded to 32 KB in subsequent systems.

Silicon technology advances permitted a second POWER processor design that allowed designers to place more than a million transistors on a single chip. This RISC Single Chip (RSC) design integrated a simple branch unit, fixed-point unit, floating-point unit, unified cache, memory controller, and an I/O controller on a single silicon chip for use in low-cost desktop systems. Products using this processor, such as the POWERstation Model 220, began shipping in April 1992.

The third design, the POWER2, delivers more performance than either of the previous POWER designs by using nearly twice as many execution units. Products using this processor, such as the Model 590, began shipping in October 1993.

Broadening the Application Market

When purchasing computer systems, customers consider not only performance, but how these systems improve productivity and reduce the cost of running their business. A key to achieving these objectives is the number of different tasks the systems can perform. Most customers prefer systems with a broad range of applications. This allows them to choose applications based on performance, cost, user skills, application integration requirements, and other needs.

In 1990, five RISC architectures competed for market share. It was unlikely that all five would succeed in the workstation market. The appli-

cation marketplace would be less attractive to application developers if it were fragmented by five architectures, because each version of an application would attract fewer customers.

PowerPC Architecture

The article entitled "PowerPC Alliance" in this issue describes IBM's alliance with Apple and Motorola. Early in 1991, processor architects, compiler experts, operating system developers, processor designers, system architects, and system designers from the three companies began working together to develop an architecture that would meet the needs of the alliance. Since it was impossible to develop a completely new architecture in time to satisfy the needs of their customers, the companies decided to use the POWER Architecture as the starting point. They made changes to achieve several specific goals. The architecture had to do the following:

Most POWER applications will benefit from the improved performance of new PowerPC processors.

- Allow a broad range of implementations from low-cost controllers to high-performance processors
- Be sufficiently simple to allow a processor design that has a very short cycle time
- Minimize effects that hinder the design of aggressive superscalar implementations
- Include multiprocessor features
- Define a 64-bit architecture that is a superset of the 32-bit architecture, providing application-binary compatibility for 32-bit applications

After analyzing the needs of applications and operating systems, considering typical instruction mixes, and inspecting application and operating system traces, the architecture group agreed on the definition of the PowerPC Architecture. It achieves the goals listed above, yet permits POWER customers to run their existing applications on new systems and to run new applications on their existing systems.

The PowerPC Architecture includes most POWER instructions. The excluded POWER instructions are executed infrequently and can be replaced with several instructions that are used in both architectures. The excluded instructions will cause a program interrupt on PowerPC processors and will be emulated by the AIX operating system.

Most POWER applications will benefit from the improved performance of new PowerPC processors. Other applications that frequently perform the operations listed below will produce correct results on PowerPC systems but may run slowly unless they are recompiled:

- Extended-precision bit string computation
- Extended-precision multiplication
- Integer division
- Generation or modification of instructions about to be executed

The first PowerPC processor, the 601, implements all but two of the non-privileged POWER instructions. This "bridge" processor allows additional time for the application vendors to recompile their products for PowerPC systems. Most existing POWER applications will run well on 601-based systems. Applications that generate or modify code use the POWER cache flush instruction that the 601 does not implement. These programs will produce correct results because the operation is emulated by AIX, but they may run slowly without recompilation. As noted above, the 601 implements most excluded non-privileged POWER instructions. New applications should not use these excluded instructions because future PowerPC processors will not implement them.

The PowerPC Architecture extends addressing and fixed-point computation to 64 bits, and supports dynamic switching between the 64-bit mode and the 32-bit mode. In 32-bit mode, a 64-bit PowerPC processor will execute application binaries compiled for the 32-bit subset architecture. Because a description of the entire architecture is too large to be addressed here, this article concentrates on describing changes that affect the user-mode 32-bit architecture.

The Slimming of POWER Architecture

In addition to excluding some POWER instructions, the PowerPC Architecture also removed or relaxed some non-critical requirements. PowerPC also defined new instructions and features to complete the architecture and to achieve the goals set by the architecture group.

The architecture group identified some features of the POWER Architecture that were too restrictive, too complex to implement efficiently, or not cost-effective. This resulted in excluding 32 problem-state instructions described below.

The POWER Architecture contained an MQ register with two characteristics that could complicate a superscalar design: the use of the register was implicit and it was a single resource. The MQ register would have required the instruction dispatching function in an aggressive design to identify all instructions that use the MQ and to

implement a special renaming mechanism for it. The functions provided by the MQ did not justify the added complexity. The exclusion of the MQ resulted in excluding 15 extended-precision shift instructions, two divide instructions, and one extended-precision multiply instruction.

Several instructions from the POWER Architecture would have made future processor designs significantly more complex. Three of the POWER fixed-point rotate-insert instructions required three source operands. An extra source operand could have required an additional register file port, which would have drastically increased the complexity of the renaming mechanism in superscalar design. Two instructions that computed absolute values and two others that performed a subtract with a lower bound of zero required selection logic after the arithmetic computation and before the writeback to the register file. The need for selection logic after the arithmetic computation could have resulted in a longer minimum cycle time for some implementations.

PowerPC also excluded five other seldom-used problem-state instructions. The most complex instruction in the POWER Architecture, Load String and Compare Byte, was one of these.

The specification for the PowerPC memory management mechanism excluded the POWER cache management instructions. PowerPC omitted the database locking and tracking mechanism, another feature of the POWER memory model. Without this exclusion, memory management would have been significantly more complex.

In some areas, PowerPC requirements are more flexible than POWER requirements. In the POWER Architecture, an update form load could specify the same register for both target (RT) and base address (RA). If this occurred, the processor performed the load but suppressed the update. The PowerPC Architecture relaxed this requirement, recognizing that there is no reason to use the update form instruction in this case because the processor will effectively discard the updated address. A similar problem exists with an update form for load and store instructions in which RA is GPR 0. If RA is GPR 0, the effective address computation uses the value 0 rather than the content of GPR 0. Saving the updated effective address is senseless when the next use also ignores the value in RA (GPR 0). These operations are not valid in the PowerPC Architecture and the result is undefined.

Another example of relaxed requirements deals with fields in instruction formats that are not necessary for a particular instruction. For example, many POWER instruction formats have an Rc bit that controls whether the condition code records the status information and the results of the operation. In the POWER Architecture, if the Rc bit is a 1 in an instruction for which this status information is irrelevant (such as loads and stores), the instruction executes correctly but the status information consists of undefined values. The PowerPC Architecture considers these instructions as *invalid forms*, and if executed, the result is undefined, except that memory protection and privilege are not violated.

The timing facility defined in the POWER Architecture would have been very difficult to implement in a power management environment where processor clock rates might be non-integral multiples of the system clock. Since the timing facility specified in the PowerPC Architecture allows more flexibility, it permits more cost-effective designs. A consequence of this approach is that PowerPC applications normally must use a system service to obtain the correct time. Direct use of the timing facility will not be possible without knowledge of environment variables, which are available only to the system.

PowerPC relaxed some requirements of the POWER Architecture, eliminating aspects that added significant complexity.

Because of the rapid progress of software and hardware technology, the cost-to-value relationship of some POWER Architecture features has changed. PowerPC relaxed some requirements of the POWER Architecture, eliminating those aspects of the POWER Architecture that added significant complexity, limited the implementation speed, and restricted the number of execution units that a superscalar design could implement.

These exclusions created deficiencies in the resulting architecture, such as the lack of a divide instruction. The POWER Architecture lacked the features necessary to build multiprocessor systems. The architecture did not define single-precision floating-point computation instructions, so it was not possible to build a low-cost processor with good single-precision performance.

Completing the PowerPC Architecture

The PowerPC Architecture added features correcting both sets of deficiencies. Where possible, architects defined additional features to avoid adding hardware complexity to compensate for software errors.

Architects added a full set of single-precision floating-point arithmetic instructions, plus two instructions to convert floating-point values to integers. Because most of these instructions provide the single-precision version of existing

double-precision instructions, complexity was not much greater.

The architects also added fixed-point divide and multiply instructions to provide both normal and extended-precision computation without the complexities associated with the MQ register. Two other new instructions (subtract-without-carry and extend-sign-byte) completed the fixed-point computation set.

Apart from the changes to the instruction set, the most significant changes were in the memory model and the memory management definition. In the POWER Architecture, the processor did not maintain data memory that was consistent with either I/O accesses or instruction fetches. Software had to manage memory consistency for both these areas.

After changes to the instruction set, the most significant changes were in the memory model and the memory management definition.

Before copying an area of memory to disk, software ensured that any modified copies of the memory area that were in the data cache had been written to main memory. Before reading from disk, software had to verify that the data cache did not contain a copy of any part of the memory area. If a copy was present, software had to invalidate the copy before restarting the program that requested the operation. POWER processors always access main memory through the caches. The only memory accesses that did not use the caches were those to a separate address space referred to as "T=1 space."

The PowerPC memory model provides greater flexibility than the POWER model. Attributes associated with each page of virtual memory allow software to control how the system accesses memory. The model allows speculative access to any page unless it has an attribute indicating that it contains I/O or it exhibits other volatile characteristics. Other attributes control whether a page may be stored in a cache or written through the cache to main memory. The design makes it possible to map I/O into the memory space. Since the architecture supports processor-enforced data memory consistency, software is not responsible for the consistency of memory with respect to I/O operations.

As in the POWER memory model, the PowerPC memory model requires software to maintain instruction memory consistent with data memory. Programs that modify or generate instructions must ensure that the new instruction is consistent with the existing instruction before attempting to execute it. Because the PowerPC Architecture excludes the POWER cache management instructions, five new cache management instructions that provide similar functions were defined.

The architects defined the PowerPC memory model to be weakly consistent, allowing out-of-order memory access. This model requires software to include explicit synchronization instructions when data shared by multiple processors is altered. The Synchronize (`sync`) instruction provides this function for all memory. The Enforce In-order Execution of I/O (`eieio`) instruction efficiently controls the order of access to I/O devices.

Because the POWER Architecture lacks an atomic memory operation, two new instructions provide a means of performing a memory operation that appears to read and then modify a memory location as one atomic operation. Load Word and Reserve (`LWARX`) copies the content of the target memory location into a register and reserves that location. The program can use the loaded value to compute the new value to store. When the processor executes a Store Word Conditional (`STWCX`) instruction, it attempts to store the new value. If the location has not been modified since the `LWARX` was completed, the processor stores the value. The processor records the status to indicate the result. If the `STWCX` fails to store the value, the program can repeat the sequence.

Because these new instructions do not affect the binary compatibility of POWER applications, the architecture defines them to access only word-aligned memory locations. If these instructions attempt to access a location that is not word-aligned, the architecture does not define the result. This definition permits simpler implementations, avoiding the need for hardware checking for a programming error that is easily avoided.

The POWER memory model is big-endian. The PowerPC memory model supports both big-endian and little-endian. A mode, controlled by software, specifies whether the current memory semantics are big-endian or little-endian. This allows PowerPC processors to be used in systems designed to run big-endian applications, little-endian applications, or both.

***Ed Silba**, IBM Corporation, RISC System/6000 Division, 11400 Burnet Road, Austin, TX 78758. Mr. Silba joined the processor architecture department on the RISC System/6000 project and participated in migrating the POWER Architecture to the PowerPC Architecture. He received BS and MS degrees in Electrical Engineering from the University of Illinois at Urbana.*