MPR601TSU-02
(IBM Order Number)

MPC601/D
(Motorola Order Number)

11/93
REV 1

PowerPC™

Advance Information

PowerPC™ 601 RISC Microprocessor Technical Summary

This document provides an overview of the PowerPC 601 RISC microprocessor features, including a block diagram showing the major functional components. It also provides an overview of the PowerPC architecture, and information about how the 601 implementation differs from the architectural definitions.

This document is divided into three parts:

- Part 1, “PowerPC 601 Microprocessor Overview,” provides an overview of the 601 features, including a block diagram showing the major functional components.
- Part 2, “Levels of the PowerPC Architecture,” describes the three levels of the PowerPC architecture.
- Part 3, “PowerPC 601 Microprocessor: Implementation,” describes the PowerPC architecture in general, noting where the 601 differs.

In this document, the terms “PowerPC 601 RISC Microprocessor” and “601” are used to denote the first microprocessor from the PowerPC architecture family. The PowerPC 601 microprocessors are available from IBM as PPC601 and from Motorola as MPC601.

IBM Microelectronics

MOTOROLA

Part 1 PowerPC 601 Microprocessor Overview

Part 1 describes the features of the 601, provides a block diagram showing the major functional units, and gives an overview of how the 601 operates.

The 601 is the first implementation of the PowerPC family of reduced instruction set computer (RISC) microprocessors. The 601 implements the 32-bit portion of the PowerPC architecture, which provides 32-bit effective (logical) addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits. For 64-bit PowerPC implementations, the PowerPC architecture provides 64-bit integer data types, 64-bit addressing, and other features required to complete the 64-bit architecture.

The 601 is a superscalar processor capable of issuing and retiring three instructions per clock, one to each of three execution units. Instructions can complete out of order for increased performance; however, the 601 makes execution appear sequential.

The 601 integrates three execution units—an integer unit (IU), a branch processing unit (BPU), and a floating-point unit (FPU). The ability to execute three instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for 601-based systems. Most integer instructions execute in one clock cycle. The FPU is pipelined so a single-precision multiply-add instruction can be issued every clock cycle.

The 601 includes an on-chip, 32-Kbyte, eight-way set-associative, physically addressed, unified instruction and data cache and an on-chip memory management unit (MMU). The MMU contains a 256-entry, two-way set-associative, unified translation lookaside buffer (UTLB) and provides support for demand paged virtual memory address translation and variable-sized block translation. Both the UTLB and the cache use least recently used (LRU) replacement algorithms.

The 601 has a 64-bit data bus and a 32-bit address bus. The 601 interface protocol allows multiple masters to compete for system resources through a central external arbiter. Additionally, on-chip snooping logic maintains cache coherency in multiprocessor applications. The 601 supports single-beat and burst data transfers for memory accesses; it also supports both memory-mapped I/O and I/O controller interface addressing.

The 601 uses an advanced, 3.6-V CMOS process technology and maintains full interface compatibility with TTL devices.

1.1 PowerPC 601 Microprocessor Features

This section describes details of the 601's implementation of the PowerPC architecture. Major features of the 601 are as follows:

- High-performance, superscalar microprocessor
  - — As many as three instructions in execution per clock (one to each of the three execution units)
  - — Single clock cycle execution for most instructions
  - — Pipelined FPU for all single-precision and most double-precision operations
- Three independent execution units and two register files
  - — BPU featuring static branch prediction
  - — A 32-bit IU
  - — Fully IEEE 754-compliant FPU for both single- and double-precision operations
  - — Thirty-two GPRs for integer operands
  - — Thirty-two FPRs for single- or double-precision operands

- High instruction and data throughput
  - — Zero-cycle branch capability
  - — Programmable static branch prediction on unresolved conditional branches
  - — Instruction unit capable of fetching eight instructions per clock from the cache
  - — An eight-entry instruction queue that provides look-ahead capability
  - — Interlocked pipelines with feed-forwarding that control data dependencies in hardware
  - — Unified 32-Kbyte cache—eight-way set-associative, physically addressed; LRU replacement algorithm
  - — Cache write-back or write-through operation programmable on a per page or per block basis
  - — Memory unit with a two-element read queue and a three-element write queue
  - — Run-time reordering of loads and stores
  - — BPU that performs condition register (CR) look-ahead operations
  - — Address translation facilities for 4-Kbyte page size, variable block size, and 256-Mbyte segment size
  - — A 256-entry, two-way set-associative UTLB
  - — Four-entry BAT array providing 128-Kbyte to 8-Mbyte blocks
  - — Four-entry, first-level ITLB
  - — Hardware table search (caused by UTLB misses) through hashed page tables
  - — 52-bit virtual address; 32-bit physical address
- Facilities for enhanced system performance
  - — Bus speed defined as selectable division of operating frequency
  - — A 64-bit split-transaction external data bus with burst transfers
  - — Support for address pipelining and limited out-of-order bus transactions
  - — Snooped copyback queues for cache block (sector) copyback operations
  - — Bus extensions for I/O controller interface operations
  - — Multiprocessing support features that include the following:
    - – Hardware enforced, four-state cache coherency protocol (MESI)
    - – Separate port into cache tags for bus snooping
- In-system testability and debugging features through boundary-scan capability

1.2 Block Diagram

Figure 1 provides a block diagram of the 601 that illustrates how the execution units—IU, FPU, and BPU—operate independently and in parallel.

The 601's 32-Kbyte, unified cache tag directory has a port dedicated to snooping bus transactions, preventing interference with processor access to the cache. The 601 also provides address translation and protection facilities, including a UTLB and a BAT array, and a four-entry ITLB that contains the four most recently used instruction address translations for fast access by the instruction unit.

Instruction fetching and issuing is handled in the instruction unit. Translation of addresses for cache or external memory accesses are handled by the memory management unit. Both units are discussed in more detail in Sections 1.3, “Instruction Unit,” and 1.5, “Memory Management Unit (MMU).”

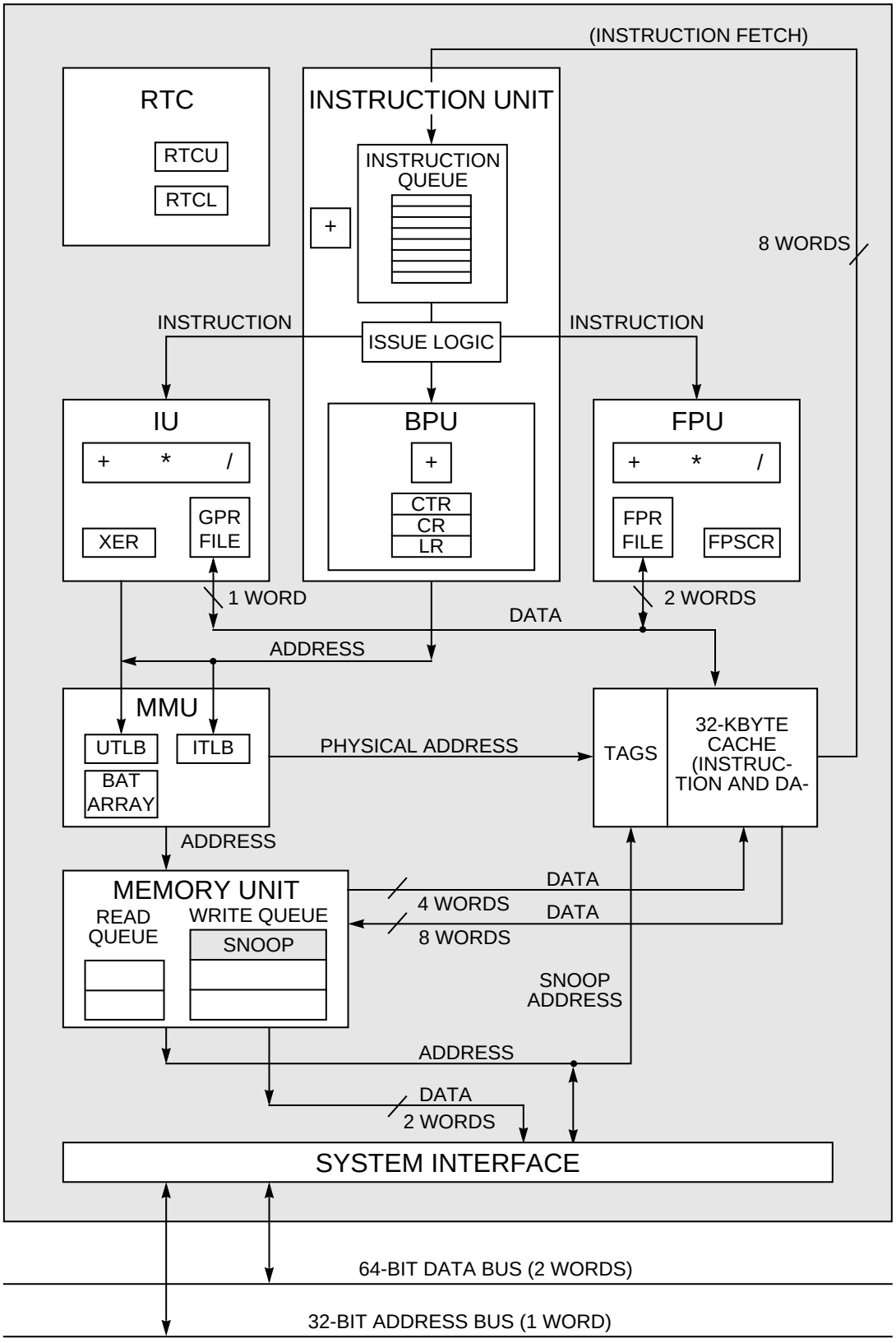


Figure 1. PowerPC 601 Microprocessor Block Diagram

1.3 Instruction Unit

As shown in Figure 1, the 601 instruction unit, which contains an instruction queue and the BPU, provides centralized control of instruction flow to the execution units. The instruction unit determines the address of the next instruction to be fetched based on information from a sequential fetcher and the BPU. The IU also enforces pipeline interlocks and controls feed-forwarding.

The sequential fetcher contains a dedicated adder that computes the address of the next sequential instruction based on the address of the last fetch and the number of words accepted into the queue. The BPU searches the bottom half of the instruction queue for a branch instruction and uses static branch prediction on unresolved conditional branches to allow the instruction fetch unit to fetch instructions from a predicted target instruction stream while a conditional branch is evaluated. The BPU also folds out branch instructions for unconditional branches.

Instructions issued beyond a predicted branch do not complete execution until the branch is resolved, preserving the programming model of sequential execution. If any of these instructions are to be executed in the BPU, they are decoded but not issued. FPU and IU instructions are issued and allowed to complete up to the register write-back stage. Write-back is performed when a correctly predicted branch is resolved, and instruction execution continues without interruption along the predicted path.

If branch prediction is incorrect, the instruction fetcher flushes all predicted path instructions and instructions are issued from the correct path.

1.3.1 Instruction Queue

The instruction queue, shown in Figure 1, holds as many as eight instructions (a cache block) and can be filled from the cache during a single cycle. The instruction fetch can access only one cache sector at a time and will load as many instruction as space in the IQ allows.

The upper half of the instruction queue (Q4–Q7) provides buffering to reduce the frequency of cache accesses. Integer and branch instructions are dispatched to their respective execution units from Q0 through Q3. Q0 functions as the initial decode stage for the IU.

For a more detailed overview of instruction dispatch, see Section 3.7, “Instruction Timing.”

1.4 Independent Execution Units

The PowerPC architecture’s support for independent floating-point, integer, and branch processing execution units allows implementation of processors with out-of-order instruction issue. For example, because branch instructions do not depend on GPRs or FPRs, branches can often be resolved early, eliminating stalls caused by taken branches.

The following sections describe the 601’s three execution units—the BPU, IU, and FPU.

1.4.1 Branch Processing Unit (BPU)

The BPU performs condition register (CR) look-ahead operations on conditional branches. The BPU looks through the bottom half of the instruction queue for a conditional branch instruction and attempts to resolve it early, achieving the effect of a zero-cycle branch in many cases.

The BPU uses a bit in the instruction encoding to predict the direction of the conditional branch. Therefore, when an unresolved conditional branch instruction is encountered, the 601 fetches instructions from the predicted target stream until the conditional branch is resolved.

The BPU contains an adder to compute branch target addresses and three special-purpose, user-control registers—the link register (LR), the count register (CTR), and the CR. The BPU calculates the return pointer for subroutine calls and saves it into the LR for certain types of branch instructions. The LR also contains the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction. The CTR contains the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction. The contents of the LR and CTR can be copied to or from any GPR. Because the BPU uses dedicated registers rather than general-purpose or floating-point registers, execution of branch instructions is largely independent from execution of integer and floating-point instructions.

1.4.2 Integer Unit (IU)

The IU executes all integer instructions and executes floating-point memory accesses in concert with the FPU. The IU executes one integer instruction at a time, performing computations with its arithmetic logic unit (ALU), multiplier, divider, integer exception register (XER), and the general-purpose register file. Most integer instructions are single-cycle instructions.

The IU interfaces with the cache and MMU for all instructions that access memory. Addresses are formed by adding the source 1 register operand specified by the instruction (or zero) to either a source 2 register operand or to a 16-bit, immediate value embedded in the instruction.

Load and store instructions are issued and translated in program order; however, the accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering.

Load and store instructions are considered to have completed execution with respect to precise exceptions after the address is translated. If the address for a load or store instruction hits in the UTLB or BAT array and it is aligned, the instruction execution (that is, calculation of the address) takes one clock cycle, allowing back-to-back issue of load and store instructions. The time required to perform the actual load or store operation varies depending on whether the operation involves the cache, system memory, or an I/O device.

1.4.3 Floating-Point Unit (FPU)

The FPU contains a single-precision multiply-add array, the floating-point status and control register (FPSCR), and thirty-two 64-bit FPRs. The multiply-add array allows the 601 to efficiently implement floating-point operations such as multiply, add, divide, and multiply-add. The FPU is pipelined so that most single-precision instructions and many double-precision instructions can be issued back-to-back. The FPU contains two additional instruction queues. These queues allow floating-point instructions to be issued from the instruction queue even if the FPU is busy, making instructions available for issue to the other execution units.

Like the BPU, the FPU can access instructions from the bottom half of the instruction queue (Q3–Q0), which permits floating-point instructions that do not depend on unexecuted instructions to be issued early to the FPU.

The 601 supports all IEEE 754 floating-point data types (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software exception routines.

1.5 Memory Management Unit (MMU)

The 601's MMU supports up to 4 Petabytes (2^{52}) of virtual memory and 4 Gigabytes (2^{32}) of physical memory. The MMU also controls access privileges for these spaces on block and page granularities. Referenced and changed status are maintained by the processor for each page to assist implementation of a demand-paged virtual memory system.

The instruction unit generates all instruction addresses; these addresses are both for sequential instruction fetches and addresses that correspond to a change of program flow. The integer unit generates addresses for data accesses (both for memory and the I/O controller interface).

After an address is generated, the upper order bits of the logical (effective) address are translated by the MMU into physical address bits. Simultaneously, the lower order address bits (that are untranslated and therefore considered both logical and physical), are directed to the on-chip cache where they form the index into the eight-way set-associative tag array. After translating the address, the MMU passes the higher-order bits of the physical address to the cache, and the cache lookup completes. For cache-inhibited accesses or accesses that miss in the cache, the untranslated lower order address bits are concatenated with the translated higher-order address bits; the resulting 32-bit physical address is then used by the memory unit and the system interface, which accesses external memory.

The MMU also directs the address translation and enforces the protection hierarchy programmed by the operating system in relation to the supervisor/user privilege level of the access and in relation to whether the access is a load or store.

For instruction accesses, the MMU first performs a lookup in the four entries of the ITLB for both block- and page-based physical address translation. Instruction accesses that miss in the ITLB and all data accesses cause a lookup in the UTLB and BAT array for the physical address translation. In most cases, the physical address translation resides in one of the TLBs and the physical address bits are readily available to the on-chip cache. In the case where the physical address translation misses in the TLBs, the 601 automatically performs a search of the translation tables in memory using the information in the table search description register 1 (SDR1) and the corresponding segment register.

Memory management in the 601 is described in more detail in Section 3.6.2, “PowerPC 601 Microprocessor Memory Management.”

1.6 Cache Unit

The PowerPC 601 microprocessor contains a 32-Kbyte, eight-way set associative, unified (instruction and data) cache. The cache line size is 64 bytes, divided into two eight-word sectors, each of which can be snooped, loaded, cast-out, or invalidated independently. The cache is designed to adhere to a write-back policy, but the 601 allows control of cacheability, write policy, and memory coherency at the page and block level. The cache uses a least recently used (LRU) replacement policy.

As shown in Figure 1, the cache provides an eight-word interface to the instruction fetcher and load/store unit. The surrounding logic selects, organizes, and forwards the requested information to the requesting unit. Write operations to the cache can be performed on a byte basis, and a complete read-modify-write operation to the cache can occur in each cycle.

The instruction unit provides the cache with the address of the next instruction to be fetched. In the case of a cache hit, the cache returns the instruction and as many of the instructions following it as can be placed in the eight-word instruction queue up to the cache sector boundary. If the queue is empty, as many as eight words (an entire sector) can be loaded into the queue in parallel.

The cache tag directory has one address port dedicated to instruction fetch and load/store accesses and one dedicated to snooping transactions on the system interface. Therefore, snooping does not require additional clock cycles unless a snoop hit that requires a cache status update occurs.

1.7 Memory Unit

The 601's memory unit contains read and write queues that buffer operations between the external interface and the cache. These operations are comprised of operations resulting from load and store instructions that are cache misses and read and write operations required to maintain cache coherency, table search, and other operations. The memory unit also handles address-only operations and cache-inhibited loads and stores. As shown in Figure 2, the read queue contains two elements and the write queue contains three elements. Each element of the write queue can contain as many as eight words (one sector) of data. One element of the write queue, marked snoop in Figure 2, is dedicated to writing cache sectors to system memory after a modified sector is hit by a snoop from another processor or snooping device on the system bus. The use of the write queue guarantees a high priority operation that ensures a deterministic response behavior when snooping hits a modified sector.

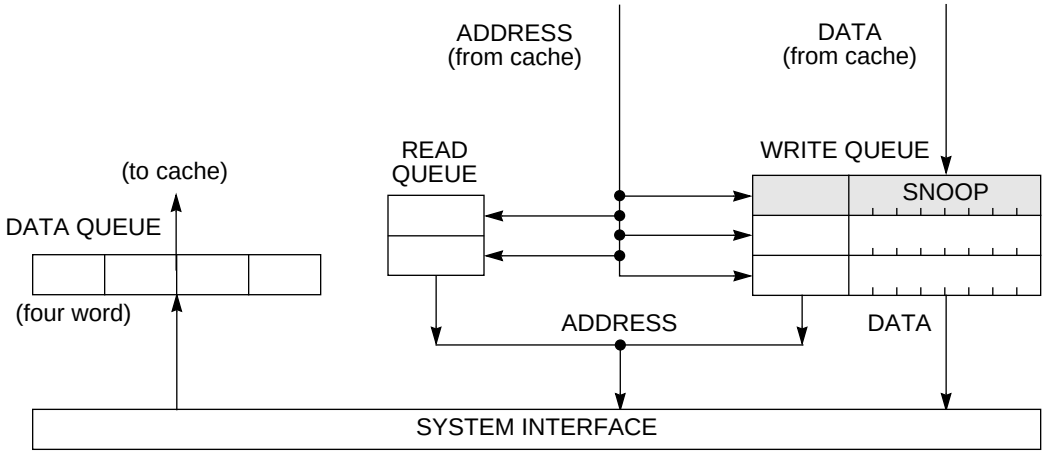


Figure 2. Memory Unit

The other two elements in the write queue are used for store operations and writing back modified sectors that have been deallocated by updating the queue; that is, when a cache location is full, the least-recently used cache sector is deallocated by first being copied into the write queue and from there to system memory. Note that snooping can occur after a sector has been pushed out into the write queue and before the data has been written to system memory. Therefore, to maintain a coherent memory, the write queue elements are compared to snooped addresses in the same way as the cache tags. If a snoop hits a write queue element, the data is first stored in system memory before it can be loaded into the cache of the snooping bus master. Coherency checking between the cache and the write queue prevents dependency conflicts. Single-beat writes in the write queue are not snooped; coherency is ensured through the use of special cache operations that accompany the single-beat write operation on the bus.

Execution of a load or store instruction is considered complete when the associated address translation completes, guaranteeing that the instruction has completed to the point where it is known that it will not generate an internal exception. However, after address translation is complete, a read or write operation can still generate an external exception.

Load and store instructions are always issued and translated in program order with respect to other load and store instructions. However, a load or store operation that hits in the cache can complete ahead of those that miss in the cache; additionally, loads and stores that miss the cache can be reordered as they arbitrate for the system bus.

If a load or store misses in the cache, the operation is managed by the memory unit which prioritizes accesses to the system bus. Read requests, such as loads, RWITMs, and instruction fetches have priority over single-beat write operations. The 601 ensures memory consistency by comparing target addresses and prohibiting instructions from completing out of order if an address matches. Load and store operations can be forced to execute in strict program order.

The 601 ensures memory consistency by comparing target addresses and prohibiting instructions from completing out of order if an address matches. Load and store operations can be forced to execute in strict program order.

1.8 System Interface

Because the cache on the 601 is an on-chip, write-back primary cache, the predominant type of transaction for most applications is burst-read memory operations, followed by burst-write memory operations, I/O controller interface operations, and single-beat (noncacheable or write-through) memory read and write operations. Additionally, there can be address-only operations, variants of the burst and single-beat operations (global memory operations that are snooped, and atomic memory operations, for example), and address retry activity (for example, when a snooped read access hits a modified line in the cache).

Memory accesses can occur in single-beat (1–8 bytes) and four-beat burst (32 bytes) data transfers. The address and data buses are independent for memory accesses to support pipelining and split transactions. The 601 can pipeline as many as two transactions and has limited support for out-of-order split-bus transactions.

Access to the system interface is granted through an external arbitration mechanism that allows devices to compete for bus mastership. This arbitration mechanism is flexible, allowing the 601 to be integrated into systems that implement various fairness and bus parking procedures to avoid arbitration overhead. Additional multiprocessor support is provided through coherency mechanisms that provide snooping, external control of the on-chip cache and TLB, and support for a secondary cache. Multiprocessor software support is provided through the use of atomic memory operations.

Typically, memory accesses are weakly ordered—sequences of operations, including load/store string and multiple instructions, do not necessarily complete in the order they begin—maximizing the efficiency of the bus without sacrificing coherency of the data. The 601 allows read operations to precede store operations (except when a dependency exists, of course). In addition, the 601 can be configured to reorder high priority write operations ahead of lower priority store operations. Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

Part 2 Levels of the PowerPC Architecture

The PowerPC architecture consists of the following layers, and adherence to the PowerPC architecture can be measured in terms of which of the following levels of the architecture is implemented:

- PowerPC user instruction set architecture—Defines the base user-level instruction set, user-level registers, data types, floating-point exception model, memory models for a uniprocessor environment, and programming model for uniprocessor environment.
- PowerPC virtual environment architecture—Describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the PowerPC virtual environment architecture also adhere to the PowerPC user instruction set architecture, but may not necessarily adhere to the PowerPC operating environment architecture.
- PowerPC operating environment architecture—Defines the memory management model, supervisor-level registers, synchronization requirements, and the exception model. Implementations that conform to the PowerPC operating environment architecture also adhere to the PowerPC user instruction set architecture and the PowerPC virtual environment architecture definition.

Note that while the 601 is said to adhere to the PowerPC architecture at all three levels, it diverges in aspects of its implementation to a greater extent than should be expected of subsequent PowerPC processors. Many of the differences result from the fact that the 601 design provides compatibility with an existing architecture standard (POWER), while providing a reliable platform for hardware and software development compatible with subsequent PowerPC processors.

Note that except for the POWER instructions and the RTC implementation, the differences between the 601 and the PowerPC architecture are primarily differences in the operating environment architecture.

The PowerPC architecture allows a wide range of designs for such features as cache and system interface implementations.

Part 3 PowerPC 601 Microprocessor: Implementation

The PowerPC architecture is derived from the IBM Performance Optimized with Enhanced RISC (POWER) architecture. The PowerPC architecture shares the benefits of the POWER architecture optimized for single-chip implementations. The architecture design facilitates parallel instruction execution and is scalable to take advantage of future technological gains. For compatibility, the 601 also implements instructions from the POWER user programming model that are not part of the PowerPC definition.

Part 3, “PowerPC 601 Microprocessor: Implementation,” describes the PowerPC architecture in general, noting where the 601 differs. The organization of Part 3 follows the sequence of the chapters in the *PowerPC 601 RISC Microprocessor User’s Manual* as follows:

- Features—Section 3.1, “Features,” describes general features that the 601 shares with the PowerPC family of microprocessors. It does not list PowerPC features not implemented in the 601.
- Registers and programming model—Section 3.2, “Registers and Programming Model,” describes the registers for the operating environment architecture common among PowerPC processors and describes the programming model. It also describes differences in how the registers are used in the 601 and describes the additional registers that are unique to the 601.
- Instruction set and addressing modes—Section 3.3, “Instruction Set and Addressing Modes,” describes the PowerPC instruction set and addressing modes for the PowerPC operating environment architecture. It defines the PowerPC instructions implemented in the 601 as well as additional instructions implemented in the 601 but not defined in the PowerPC architecture.

- Cache implementation—Section 3.4, “Cache Implementation,” describes the cache model that is defined generally for PowerPC processors by the virtual environment architecture. It also provides specific details about the 601 cache implementation.
- Exception model—Section 3.5, “Exception Model,” describes the exception model of the PowerPC operating environment architecture and the differences in the 601 exception model.
- Memory management—Section 3.6, “Memory Management,” describes generally the conventions for memory management among the PowerPC processors. This section also describes the general differences between the 601 and the 32-bit PowerPC memory management specification.
- Instruction timing—Section 3.7, “Instruction Timing,” provides a general description of the instruction timing provided by the superscalar, parallel execution supported by the PowerPC architecture.
- System interface—Section 3.8, “System Interface,” describes the signals implemented on the 601.

3.1 Features

The 601 is a high-performance, superscalar PowerPC implementation. The PowerPC architecture allows optimizing compilers to schedule instructions to maximize performance through efficient use of the PowerPC instruction set and register model. The multiple, independent execution units allow compilers to maximize parallelism and instruction throughput. Compilers that take advantage of the flexibility of the PowerPC architecture can additionally optimize system performance of the PowerPC processors.

The 601 implements the PowerPC architecture, with the extensions and variances listed in Appendix H, “Implementation Summary for Programmers,” in the *PowerPC 601 RISC Microprocessor User’s Manual*.

Specific features of the 601 are listed in Section 1.1, “PowerPC 601 Microprocessor Features.”

3.2 Registers and Programming Model

The following subsections describe the general features of the PowerPC registers and programming model and of the specific 601 implementation, respectively.

3.2.1 PowerPC Registers and Programming Model

The PowerPC architecture defines register-to-register operations for most computational instructions. Source operands for these instructions are accessed from the registers or are provided as immediate values embedded in the instruction opcode. The three-register instruction format allows specification of a target register distinct from the two source operands. Load and store instructions transfer data between registers and memory.

PowerPC processors have two levels of privilege—supervisor mode of operation (typically used by the operating environment) and one that corresponds to the user mode of operation (used by the application software). The programming models incorporate 32 GPRs, 32 FPRs, special-purpose registers (SPRs), and several miscellaneous registers. Note that there are several registers that are part of the PowerPC architecture that are not implemented in the 601; for example, the time base registers are not implemented in the 601. Likewise, each PowerPC implementation has its own unique set of hardware implementation (HID) registers, which are implementation-specific.

This division allows the operating system to control the application environment (providing virtual memory and protecting operating-system and critical machine resources). Instructions that control the state of the processor, the address translation mechanism, and supervisor registers can be executed only when the processor is operating in supervisor mode.

The following sections summarize the PowerPC registers that are implemented in the 601 processor. Chapter 2, “Register Models and Data Types,” in the *PowerPC 601 RISC Microprocessor User’s Manual* provides detailed information about the registers implemented in the 601.

3.2.1.1 General-Purpose Registers (GPRs)

The PowerPC architecture defines 32 user-level, general-purpose registers (GPRs). These registers are either 32 bits wide in 32-bit PowerPC implementations and 64 bits wide in 64-bit PowerPC implementations. The GPRs serve as the data source or destination for all integer instructions.

3.2.1.2 Floating-Point Registers (FPRs)

The PowerPC architecture also defines 32 user-level 64-bit floating-point registers (FPRs). The FPRs serve as the data source or destination for floating-point instructions. These registers can contain data objects of either single- or double-precision floating-point formats.

3.2.1.3 Condition Register (CR)

The CR is a 32-bit user-level register that consists of eight four-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

3.2.1.4 Floating-Point Status and Control Register (FPSCR)

The floating-point status and control register (FPSCR) is a user-level register that contains all exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE 754 standard.

3.2.1.5 Machine State Register (MSR)

The machine state register (MSR) is a supervisor-level register that defines the state of the processor. The contents of this register is saved when an exception is taken and restored when the exception handling completes. The 601 implements the MSR as a 32-bit register; 64-bit PowerPC processors implement a 64-bit MSR.

3.2.1.6 Segment Registers (SRs)

For memory management, 32-bit PowerPC implementations implement sixteen 32-bit segment registers (SRs). The fields in the segment register are interpreted differently depending on the value of bit 0.

3.2.1.7 Special-Purpose Registers (SPRs)

The PowerPC operating environment architecture defines numerous special-purpose registers that serve a variety of functions, such as providing controls, indicating status, configuring the processor, and performing special operations. Some SPRs are accessed implicitly as part of executing certain instructions. All SPRs can be accessed by using the Move to/from Special Purpose Register instructions, **mtspr** and **mfspr**.

In the 601, all SPRs are 32 bits wide.

3.2.1.8 User-Level SPRs

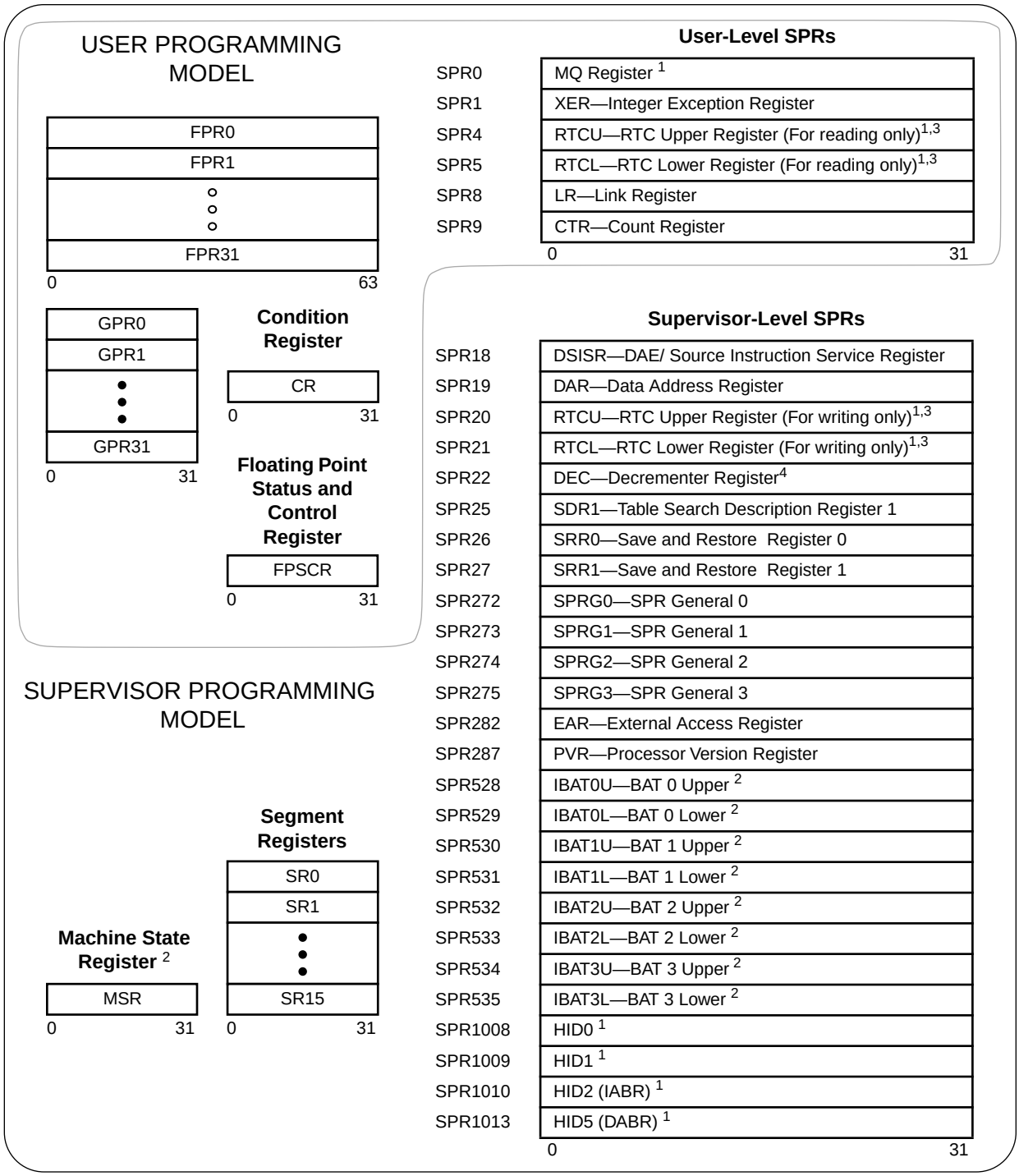
The following 601 SPRs are accessible by user-level software:

- Link register (LR)—The link register can be used to provide the branch target address and to hold the return address after branch and link instructions. The LR is 32 bits wide in 32-bit implementations.
- Count register (CTR)—The CTR is decremented and tested automatically as a result of branch-and-count instructions. The CTR is 32 bits wide in 32-bit implementations.
- Integer exception register (XER)—The 32-bit XER contains the integer carry and overflow bits and two fields for the Load String and Compare Byte Indexed (**lscbx**) instruction (a POWER instruction implemented in the 601 but not defined by the PowerPC architecture).

3.2.1.9 Supervisor-Level SPRs

The 601 also contains SPRs that can be accessed only by supervisor-level software. These registers consist of the following:

- The 32-bit data access exception (DAE)/source instruction service register (DSISR) defines the cause of data access and alignment exceptions.
- The data address register (DAR) is a 32-bit register that holds the address of an access after an alignment or data access exception.
- Decrementer register (DEC) is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay. PowerPC architecture defines that the DEC frequency be provided as a subdivision of the processor clock frequency; however, the 601 implements a separate clock input that serves both the DEC and the RTC facilities.
- The 32-bit table search description register 1(SDR1) specifies the page table format used in logical-to-physical address translation for pages.
- The machine status save/restore register 0 (SRR0) is a 32-bit register that is used by the 601 for saving the address of the instruction that caused the exception, and the address to return to when a Return from Interrupt (**rfi**) instruction is executed.
- The machine status save/restore register 1 (SRR1) is a 32-bit register used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed.
- General SPRs, SPRG0–SPRG3, are 32-bit registers provided for operating system use.
- The external access register (EAR) is a 32-bit register that controls access to the external control facility through the External Control Input Word Indexed (**eciwx**) and External Control Output Word Indexed (**ecowx**) instructions.
- The processor version register (PVR) is a 32-bit, read-only register that identifies the version (model) and revision level of the PowerPC processor.
- Block address translation (BAT) registers—The PowerPC architecture defines 16 BAT registers, divided into four pairs of data BATs (DBATs) and four pairs of instruction BATs (IBATs). The 601 includes four pairs of unified BATs (BAT0U–BAT3U and BAT0L–BAT3L). See Figure 3 for a list of the SPR numbers for the BAT registers. Note that the format for the 601's implementation of the BAT registers differs from the PowerPC architecture definition.



[1] 601-only registers. These registers are not necessarily supported by other PowerPC processors.
[2] These registers may be implemented differently on other PowerPC processors. The PowerPC architecture defines two sets of BAT registers—eight IBATs and eight DBATs. The 601 implements the IBATs and treats them as unified BATs.
[3] RTCU and RTCL registers can be written only in supervisor mode, in which case different SPR numbers are used.
[4] DEC register can be read by user programs by specifying SPR6 in the **mfspr** instruction (for POWER compatibility).

Figure 3. PowerPC 601 Microprocessor Programming Model—Registers

3.2.2 Additional Registers in the PowerPC 601 Microprocessor

During normal execution, a program can access the registers, shown in Figure 3, depending on the program's access privilege (supervisor or user, determined by the privilege-level (PR) bit in the machine state register (MSR)). Note that registers such as the general-purpose registers (GPRs) and floating-point registers (FPRs) are accessed through operands that are part of the instructions. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions) or implicit as the part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

The numbers to the left of the SPRs indicate the number that is used in the syntax of the instruction operands to access the register.

Figure 3 shows all the 601 registers and includes the following registers that are not part of the PowerPC architecture:

- Real-time clock (RTC) registers—RTCU and RTCL (RTC upper and RTC lower). The registers can be read from by user-level software, but can be written to only by supervisor-level software. As shown in Figure 3, the SPR numbers for the RTC registers depend on the type of access used.
- MQ register (MQ). The MQ register is a 601-specific, 32-bit register used as a register extension to accommodate the product for the multiply instructions and the dividend for the divide instructions. It is also used as an operand of long rotate and shift instructions. This register, and the instructions that require it, is provided for compatibility with POWER architecture, and is not part of the PowerPC architecture. The MQ register is typically accessed implicitly as part of executing a computational instruction.
- Block-address translation (BAT) registers. The 601 includes eight block-address translation registers (BATs), consisting of four pairs of BATs (IBAT0U–IBAT3U and IBAT0L–IBAT3L). See Figure 3 for a list of the SPR numbers for the BAT registers. Note that the PowerPC architecture has twice as many BAT registers as the 601.
- Hardware implementation registers (HID0–HID2, HID5, and HID15). These registers are provided primarily for debugging. HID15 holds the four-bit processor identification tag (PID) that is useful for differentiating processors in multiprocessor system designs. Note that while it is not guaranteed that the implementation of HID registers is consistent among PowerPC processors, other processors may be designed with similar or identical HID registers.

3.3 Instruction Set and Addressing Modes

The following subsections describe the PowerPC instruction set and addressing modes in general. Differences in the 601's instruction set are described in Section 3.3.2, "PowerPC 601 Microprocessor Instruction Set."

3.3.1 PowerPC Instruction Set and Addressing Modes

All PowerPC instructions are encoded as single-word (32-bit) opcodes. Instruction formats are consistent among all instruction types, permitting efficient decoding to occur in parallel with operand accesses. This fixed instruction length and consistent format greatly simplifies instruction pipelining.

3.3.1.1 PowerPC Instruction Set

The PowerPC instructions are divided into the following categories:

- Integer instructions—These include computational and logical instructions.
  - — Integer arithmetic instructions

  - — Integer compare instructions
  - — Integer logical instructions
  - — Integer rotate and shift instructions
- Floating-point instructions—These include floating-point computational instructions, as well as instructions that affect the floating-point status and control register (FPSCR).
  - — Floating-point arithmetic instructions
  - — Floating-point multiply/add instructions
  - — Floating-point rounding and conversion instructions
  - — Floating-point compare instructions
  - — Floating-point status and control instructions
- Load/store instructions—These include integer and floating-point load and store instructions.
  - — Integer load and store instructions
  - — Integer load and store multiple instructions
  - — Floating-point load and store
  - — Floating-point move instructions
  - — Primitives used to construct atomic memory operations (**lwarx** and **stwcx.** instructions)
- Flow control instructions—These include branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow.
  - — Branch and trap instructions
  - — Condition register logical instructions
- Processor control instructions—These instructions are used for synchronizing memory accesses and management of caches, UTLBs, and the segment registers.
  - — Move to/from special purpose register instructions
  - — Move to/from MSR
  - — Synchronize
  - — Instruction synchronize
  - — TLB invalidate
- Memory control instructions—These instructions provide control of caches, TLBs, and segment registers.
  - — Supervisor-level cache management instructions
  - — User-level cache instructions
  - — Segment register manipulation instructions
  - — Translation lookaside buffer management instructions

Note that this grouping of the instructions does not indicate which execution unit executes a particular instruction or group of instructions. This information, which is useful in taking full advantage of superscalar parallel instruction execution, is provided in Chapter 7, “Instruction Timing,” and Chapter 10, “Instruction Set,” in the *PowerPC 601 RISC Microprocessor User’s Manual*.”

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision (one word) and double-precision (one double word) floating-point operands. The PowerPC architecture uses instructions that are four bytes long and word-aligned. It provides for byte, half-word, and

word operand loads and stores between memory and a set of 32 general-purpose registers (GPRs). It also provides for word and double-word operand loads and stores between memory and a set of 32 floating-point registers (FPRs).

Computational instructions do not modify memory. To use a memory operand in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written back to the target location with distinct instructions.

PowerPC processors follow the program flow when they are in the normal execution state. However, the flow of instructions can be interrupted directly by the execution of an instruction or by an asynchronous event. Either kind of exception may cause one of several components of the system software to be invoked.

3.3.1.2 Calculating Effective Addresses

The effective address (EA) is the 32-bit address computed by the processor when executing a memory access or branch instruction or when fetching the next sequential instruction.

The PowerPC architecture supports two simple memory addressing modes:

- EA = (**r**A|0) + offset (including offset = 0) (register indirect with immediate index)
- EA = (**r**A|0) + **r**B (register indirect with index)

These simple addressing modes allow efficient address generation for memory accesses. Calculation of the effective address for aligned transfers occurs in a single clock cycle.

For a memory access instruction, if the sum of the effective address and the operand length exceeds the maximum effective address, the storage operand is considered to wrap around from the maximum effective address to effective address 0.

Effective address computations for both data and instruction accesses use 32-bit unsigned binary arithmetic. A carry from bit 0 is ignored in 32-bit implementations.

3.3.2 PowerPC 601 Microprocessor Instruction Set

The 601 instruction set is defined as follows:

- The 601 implements the 32-bit PowerPC architecture instructions except as indicated in Appendix C, “PowerPC Instructions Not Implemented,” in the *PowerPC 601 RISC Microprocessor User’s Manual*. Otherwise, all instructions not implemented in the 601 are defined as optional in the PowerPC architecture.
- The 601 supports a number of POWER instructions that are otherwise not implemented in the PowerPC architecture. These are listed in Appendix B, “POWER Architecture Cross Reference,” and individual instructions are described in Chapter 10, “Instruction Set,” in the *PowerPC 601 RISC Microprocessor User’s Manual*.
- The 601 implements the External Control Input Word Indexed (**eciwx**) and External Control Output Word Indexed (**ecowx**) instructions, which are optional in the PowerPC architecture definition.
- Several of the instructions implemented in the 601 function somewhat differently than they are defined in the PowerPC architecture. These differences typically stem from design differences; for instance, the PowerPC architecture defines several cache control instructions specific to separate instruction and data cache designs.
- When executed on the 601, such instructions may provide a subset of the functions of the instruction or they may be no-ops.

For a list of all PowerPC instructions and all 601-specific instructions, see Appendix A, “Instruction Set Listings,” in the *PowerPC 601 RISC Microprocessor User’s Manual*. Chapter 10, “Instruction Set,” in the *PowerPC 601 RISC Microprocessor User’s Manual* describes each instruction, indicating whether an instruction is 601-specific and describing any differences in the implementation on the 601.

3.4 Cache Implementation

The following subsections describe the PowerPC architecture’s treatment of cache in general, and the 601-specific implementation, respectively.

3.4.1 PowerPC Cache Characteristics

The PowerPC architecture does not define hardware aspects of cache implementations. For example, some PowerPC processors may have separate instruction and data caches (Harvard architecture), while others, such as the 601, implement a unified cache.

PowerPC implementations can control the following memory access modes on a page or block basis:

- Write-back/write-through mode
- Cache-inhibited mode
- Memory coherency

Note that in the 601 processor, a block is defined as an eight-word sector. The PowerPC virtual environment architecture defines cache management instructions that provide a means by which the application programmer can affect the cache contents.

3.4.2 PowerPC 601 Microprocessor Cache Implementation

The 601 has a 32-Kbyte, eight-way set-associative unified (instruction and data) cache. The cache is physically addressed and can operate in either write-back or write-through mode as specified by the PowerPC architecture.

The cache is configured as eight sets of 64 lines. Each line consists of two sectors, four state bits (two per sector), several replacement control bits, and an address tag. The two state bits implement the four-state MESI (modified-exclusive-shared-invalid) protocol. Each sector contains eight 32-bit words. Note that the PowerPC architecture defines the term block as the cacheable unit. For the 601 processor, the block is a sector. A block diagram of the cache organization is shown in Figure 4.

Each cache line contains 16 contiguous words from memory that are loaded from a 16-word boundary (that is, bits A26–A31 of the logical addresses are zero); thus, a cache line never crosses a page boundary. Misaligned accesses across a page boundary can incur a performance penalty.

Cache reload operations are always performed on a sector basis (that is, the cache is snooped and updated and coherency is maintained on a per-sector basis). However, if the other sector in the line is marked invalid, an optional, low-priority update of that sector is attempted after the sector that contained the critical word is filled. The ability to attempt the other sector update can be disabled by the system software.

External bus transactions that load instructions or data into the cache always transfer the missed quad word first, regardless of its location in a cache sector; then the rest of the cache sector is filled. As the missed quad word is loaded into the cache, it is simultaneously forwarded to the appropriate execution unit so instruction execution resumes as quickly as possible.

To ensure coherency among caches in a multiprocessor (or multiple caching-device) implementation, the 601 implements the MESI protocol. MESI stands for modified/exclusive/shared/invalid. These four states indicate the state of the cache block as follows:

- Modified—The cache block is modified with respect to system memory; that is, data for this address is valid only in the cache and not in system memory.
- Exclusive—This cache block holds valid data that is identical to the data at this address in system memory. No other cache has this data.
- Shared—This cache block holds valid data that is identical to this address in system memory and at least one other caching device.
- Invalid—This cache block does not hold valid data.

Cache coherency is enforced by on-chip hardware bus snooping logic. Since the cache tag directory has a separate port dedicated to snooping bus transactions, bus snooping traffic does not interfere with processor access to the cache unless a snoop hit occurs.

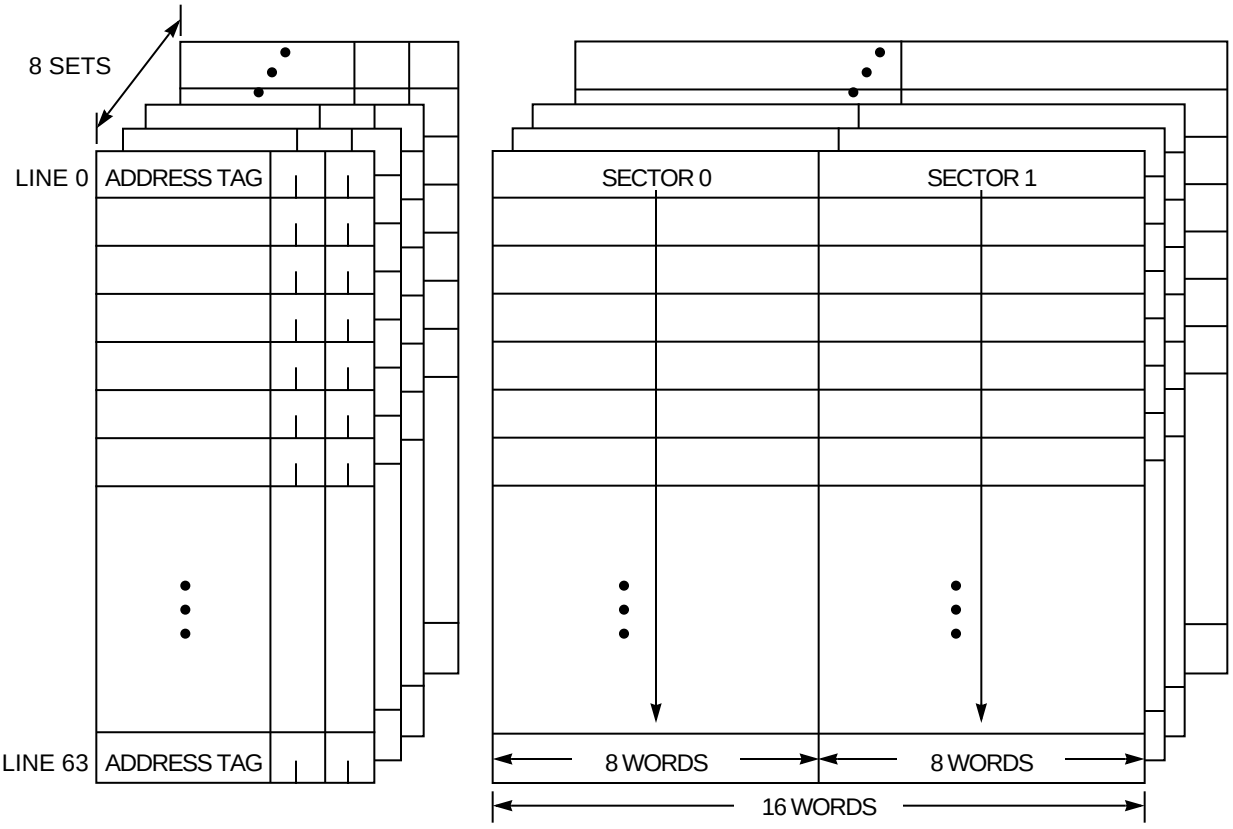


Figure 4. Cache Unit Organization

3.5 Exception Model

The following subsections describe the PowerPC exception model and the 601 implementation, respectively.

3.5.1 PowerPC Exception Model

The PowerPC exception mechanism allows the processor to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions. When exceptions occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (exception vector) predetermined for each exception. The exception handler at the specified vector is then processed with the processor in supervisor mode.

Although multiple exception conditions can map to a single exception vector, a more specific condition may be determined by examining a register associated with the exception—for example, the DAE/source instruction service register (DSISR) and the floating-point status and control register (FPSCR). Additionally, some exception conditions can be explicitly enabled or disabled by software.

The PowerPC architecture requires that exceptions be handled in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are presented strictly in order. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute state, are required to complete before the exception is taken. Any exceptions caused by those instructions are handled first. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until all instructions currently in the execute stage successfully complete execution and report their results.

Unless a catastrophic condition causes a system reset or machine check exception, only one exception is handled at a time. If, for example, a single instruction encounters multiple exception conditions, those conditions are encountered sequentially. After the exception handler handles an exception, the instruction execution continues until the next exception condition is encountered. However, in many cases there is no attempt to re-execute the instruction. This method of recognizing and handling exception conditions sequentially guarantees that exceptions are recoverable.

Exception handlers should save the information stored in SRR0 and SRR1 early to prevent the program state from being lost due to a system reset and machine check exception or to an instruction-caused exception in the exception handler, and before enabling external interrupts.

The PowerPC architecture supports four types of exceptions:

- Synchronous, precise—These are caused by instructions. All instruction-caused exceptions are handled precisely; that is, the machine state at the time the exception occurs is known and can be completely restored. This means that (excluding the trap and system call exceptions) the address of the faulting instruction is provided to the exception handler and that neither the faulting instruction nor subsequent instructions in the code stream will complete execution. The instructions that invoke trap and system call exceptions complete execution before the exception is taken. When exception processing completes, execution resumes at the address of the next instruction.
- Synchronous, imprecise—The PowerPC architecture defines two imprecise floating-point exception modes, recoverable and nonrecoverable. Even though the 601 provides a means to enable the imprecise modes, it implements these modes identically to the precise mode (that is, all enabled floating-point enabled exceptions are always precise on the 601).
- Asynchronous, precise—The external interrupt and decrementer exceptions are maskable asynchronous exceptions that are handled precisely. When these exceptions occur, their handling is postponed until all instructions, and any exceptions associated with those instructions, complete execution.
- Asynchronous, imprecise—There are two nonmaskable asynchronous exceptions that are imprecise: system reset and machine check exceptions. These exceptions may not be recoverable, or may provide a limited degree of recoverability for diagnostic purpose.

The PowerPC architecture defines several of the exceptions differently than the 601 implementation. For example, the PowerPC exception model provides a unique vector for the trace exception; the 601 vectors trace exceptions to the run-mode exception handler. Other differences are noted in Section 3.5.2, “PowerPC 601 Microprocessor Exception Model.”

3.5.2 PowerPC 601 Microprocessor Exception Model

As specified by the PowerPC architecture, all 601 exceptions can be described as either precise or imprecise and either synchronous or asynchronous. Asynchronous exceptions are caused by events external to the processor’s execution; synchronous exceptions, which are all handled precisely by the 601, are caused by instructions.

The 601 exception classes are shown in Table 1.

Table 1. PowerPC 601 Microprocessor Exception Classifications

Synchronous/Asynchronous	Precise/Imprecise	Exception Type
Asynchronous	Imprecise	Machine check System reset
Asynchronous	Precise	External interrupt Decrementer
Synchronous	Precise	Instruction-caused exceptions

Although exceptions have other characteristics as well, such as whether they are maskable or nonmaskable, the distinctions shown in Table 1 define categories of exceptions that the 601 handles uniquely. Note that Table 1 includes no synchronous imprecise instructions. While the PowerPC architecture supports imprecise handling of floating-point exceptions, the 601 implements these exception modes as precise exceptions.

The 601’s exceptions, and conditions that cause them, are listed in Table 2. Exceptions that are specific to the 601 are indicated.

Table 2. Exceptions and Conditions

Exception Type	Vector Offset (hex)	Causing Conditions
Reserved	00000	—
System reset	00100	A system reset is caused by the assertion of either $\overline{\text{SRESET}}$ or $\overline{\text{HRESET}}$.
Machine check	00200	A machine check is caused by the assertion of the $\overline{\text{TEA}}$ signal during a data bus transaction.

Table 2. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Data access	00300	The cause of a data access exception can be determined by the bit settings in the DSISR, listed as follows: 1 Set if the translation of an attempted access is not found in the primary hash table entry group (HTEG), or in the rehashed secondary HTEG, or in the range of a BAT register; otherwise cleared. 4 Set if a memory access is not permitted by the page or BAT protection mechanism described in Chapter 6, “Memory Management Unit,” in the *PowerPC 601 RISC Microprocessor User’s Manual*.; otherwise cleared. 5 Set if the access was to an I/O segment (SR[T] =1) by an **eciwx**, **ecowx, lwarx**, **stwcx.**, or **lscbx** instruction; otherwise cleared. Set by **an eciwx** or **ecowx** instruction if the access is to an address that is marked as write-through. 6 Set for a store operation and cleared for a load operation. 9 Set if an EA matches the address in the DABR while in one of the three compare modes. 11 Set if **eciwx** or **ecowx** is used and EAR[E] is cleared.
Instruction access	00400	An instruction access exception is caused when an instruction fetch cannot be performed for any of the following reasons: • The effective (logical) address cannot be translated. That is, there is a page fault for this portion of the translation, so an instruction access exception must be taken to retrieve the translation from a storage device such as a hard disk drive. • The fetch access is to an I/O segment. • The fetch access violates memory protection. If the key bits (Ks and Ku) in the segment register and the PP bits in the PTE or BAT are set to prohibit read access, instructions cannot be fetched from this location.
External interrupt	00500	An external interrupt occurs when the $\overline{\text{INT}}$ signal is asserted.
Alignment	00600	An alignment exception is caused when the 601 cannot perform a memory access for any of several reasons, such as when the operand of a floating-point load or store operation is in an I/O segment (SR[T] = 1) or when a scalar load/store operand crosses a page boundary. Specific exception sources are described in Chapter 5, “Exceptions,” in the *PowerPC 601 RISC Microprocessor User’s Manual*.”

Table 2. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Program	00700	A program exception is caused by one of the following exception conditions, which correspond to bit settings in SRR1 and arise during execution of an instruction: • Floating-point enabled exception—A floating-point enabled exception condition is generated when the following condition is met: (MSR[FE0] | MSR[FE1]) & FPSCR[FEX] is 1. FPSCR[FEX] is set by the execution of a floating-point instruction that causes an enabled exception or by the execution of a “move to FPSCR” instruction that results in both an exception condition bit and its corresponding enable bit being set in the FPSCR. • Illegal instruction—An illegal instruction program exception is generated when execution of an instruction is attempted with an illegal opcode or illegal combination of opcode and extended opcode fields (including PowerPC instructions not implemented in the 601), or when execution of an optional instruction not provided in the 601 is attempted (these do not include those optional instructions that are treated as no-ops). • Privileged instruction—A privileged instruction type program exception is generated when the execution of a privileged instruction is attempted and the MSR register user privilege bit, MSR[PR], is set. In the 601, this exception is generated for **mtspr** or **mfspr** with an invalid SPR field if SPR[0] = 1 and MSR[PR] = 1. This may not be true for all PowerPC processors. • Trap—A trap type program exception is generated when any of the conditions specified in a trap instruction is met.
Floating-point unavailable	00800	A floating-point unavailable exception is caused by an attempt to execute a floating-point instruction (including floating-point load, store, and move instructions) when the floating-point available bit is disabled, MSR[FP] = 0.
Decrementer	00900	The decrementer exception occurs when the most significant bit of the decrementer (DEC) register transitions from 0 to 1. Must also be enabled with the MSR[EE] bit.
I/O error	00A00	An I/O controller interface error exception is taken only when an operation to an I/O controller interface segment fails (such a failure is indicated to the 601 by a particular bus reply packet). If an I/O controller interface exception is taken on a memory access directed to an I/O segment, the SRR0 contains the address of the instruction following the offending instruction. Note that this exception is not implemented in other PowerPC processors.
Reserved	00B00	—
System call	00C00	A system call exception occurs when a System Call (**sc**) instruction is executed.
Reserved	00D00	Other PowerPC processors may use this vector for trace exceptions.
Reserved	00E00	The 601 does not generate an interrupt to this vector. Other PowerPC processors may use this vector for floating-point assist exceptions.
Reserved	00E10–00FFF	—

Table 2. Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Reserved	01000–01FFF	Reserved, implementation-specific
Run mode exception	02000	The run mode exception is taken depending on the settings of the HID1 register and the MSR[SE] bit. The following modes correspond with bit settings in the HID1 register: • Normal run mode—No address breakpoints are specified, and the 601 executes from zero to three instructions per cycle • Single instruction step mode—One instruction is processed at a time. The appropriate break action is taken after an instruction is executed and the processor quiesces. • Limited instruction address compare—The 601 runs at full speed (in parallel) until the EA of the instruction being decoded matches the EA contained in HID2. Addresses for branch instructions and floating-point instructions may never be detected. • Full instruction address compare mode—Processing proceeds out of IQ0. When the EA in HID2 matches the EA of the instruction in IQ0, the appropriate break action is performed. Unlike the limited instruction address compare mode, all instructions pass through the IQ0 in this mode. That is, instructions cannot be folded out of the instruction stream. The following mode is taken when the MSR[SE] bit is set. • MSR[SE] trace mode—Note that in other PowerPC implementations, the trace exception is a separate exception with its own vector x'00D00'.

3.6 Memory Management

The following subsections describe the PowerPC memory management architecture, and the specific 601 implementation, respectively.

3.6.1 PowerPC Memory Management

The primary functions of the MMU are to translate logical (effective) addresses to physical addresses for memory accesses, I/O accesses (most I/O accesses are assumed to be memory-mapped), and I/O controller interface accesses, and to provide access protection on blocks and pages of memory.

There are three types of accesses generated by the 601 that require address translation: instruction accesses, data accesses to memory generated by load and store instructions, and I/O controller interface accesses generated by load and store instructions.

The PowerPC MMU and exception model support demand-paged virtual memory. Virtual memory management permits execution of programs larger than the size of physical memory; demand-paged implies that individual pages are loaded into physical memory from system memory only when they are first accessed by an executing program.

The hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of 2, and its starting address is a multiple of its size.

The page table contains a number of page table entry groups (PTEGs). A PTEG contains eight page table entries (PTEs) of eight bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

Address translations are enabled by setting bits in the MSR—MSR[IT] enables instruction translations and MSR[DT] enables data translations.

3.6.2 PowerPC 601 Microprocessor Memory Management

The 601 MMU provides 4 Gbytes of logical address space accessible to supervisor and user programs with a 4-Kbyte page size and 256-Mbyte segment size. Block sizes range from 128 Kbyte to 8 Mbyte and are software selectable. In addition, the 601 uses an interim 52-bit virtual address and hashed page tables in the generation of 32-bit physical addresses.

A UTLB provides address translation in parallel with the on-chip cache access, incurring no additional time penalty in the event of a UTLB hit. The UTLB is a cache of the most recently used page table entries. Software is responsible for maintaining the consistency of the UTLB with memory. The 601's UTLB is a 256-entry, two-way set-associative cache that contains instruction and data address translations. The 601 provides hardware table search capability through the hashed page table on UTLB misses. Supervisor software can invalidate UTLB entries selectively. In addition, UTLB control instructions can optionally be broadcast on the external interface for remote invalidations.

The 601 also provides a four-entry BAT array that maintains address translations for blocks of memory. These entries define blocks that can vary from 128 Kbytes to 8 Mbytes. The BAT array is maintained by system software.

To accelerate the instruction unit operation, the 601 uses a four-entry ITLB. The ITLB contains up to four copies of the most recently used instruction address translations (page or block) providing the instruction unit access to the most recently used translations without requiring the UTLB or BAT array. The processor ensures that the ITLB is consistent with the UTLB, and uses an LRU replacement algorithm when a miss is encountered.

The 601 MMU relies on the exception processing mechanism for the implementation of the paged virtual memory environment and for enforcing protection of designated memory areas. Exception processing is described in Chapter 5, "Exceptions," in the *PowerPC 601 RISC Microprocessor User's Manual*. In addition, the MSR of the 601 controls some of the critical functionality of the MMU.

As specified by the PowerPC architecture, the hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of 2, and its starting address is a multiple of its size.

Also as specified by the PowerPC architecture, the page table contains a number of PTEGs. A PTEG contains eight page table entries (PTEs) of eight bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

3.7 Instruction Timing

The 601 is a pipelined superscalar processor. A pipelined processor is one in which the processing of an instruction is broken down into discrete stages, such as decode, execute, and writeback. Because the tasks required to process an instruction are broken into a series of tasks, an instruction does not require the entire resources of an execution unit. For example, after an instruction completes the decode stage, it can pass on to the next stage, while the subsequent instruction can advance into the decode stage. This improves the throughput of the instruction flow. For example, it may take three cycles for an integer instruction to complete, but if there are no stalls in the integer pipeline, a series of integer instructions can have a throughput of one instruction per cycle.

A superscalar processor is one in which multiple pipelines are provided to allow instructions to execute in parallel. The 601 has three execution units, one each for integer instructions, floating-point instructions, and branch instructions. The IU and the FPU each have dedicated register files for maintaining operands (GPRs and FPRs, respectively), allowing integer calculations and floating-point calculations to occur simultaneously without interference.

The 601 pipeline description can be broken into two parts, the processor core, where instruction execution takes place, and the memory subsystem, the interface between the processor core and system memory. The system memory includes a unified 32-Kbyte cache and the bus interface unit.

Figure 5 shows the 601's instruction queue and the IU, FPU, and BPU pipelines.

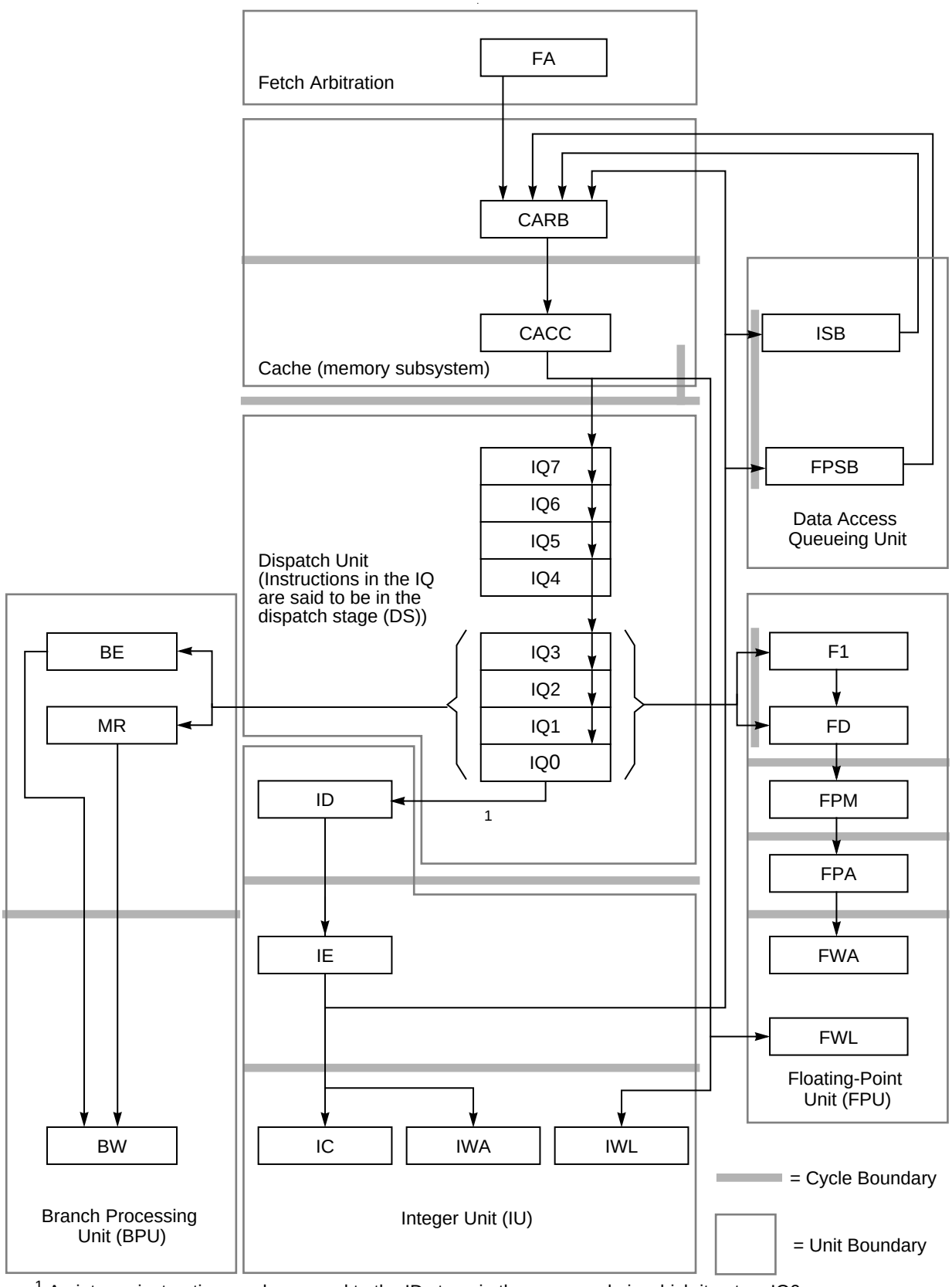
Each of the stages shown in Figure 5 is described in Chapter 7, "Instruction Timing," in the *PowerPC 601 RISC Microprocessor User's Manual*.

As shown in Figure 5, integer instructions are dispatched only from IQ0 (where they are also usually decoded); whereas branch and floating-point instructions can be dispatched from any of the bottom four elements in the instruction queue (IQ0–IQ3). The dispatch of integer instructions is restricted in this manner to provide an ordered flow of instructions through the integer pipeline, which in turn provides a mechanism that ensures that all instructions appear to complete in order. As branch and floating-point instructions are dispatched their position in the instruction stream is recorded by means of tags that accompany the previous integer instruction through the integer pipeline. Note that when a floating-point or branch instruction cannot be tagged to an integer instruction, it is tagged to a no-op, or bubble, in the integer pipeline.

Logic associated with the integer completion (IC) stage reconstructs the program order, checks for data dependencies, and schedules the write-back stages of the three pipelines. Note that it is not necessary that the write-back stages need only be serialized if there are data dependencies. For example, instructions that update the condition register (CR) must perform write-back in strict order.

The tagging mechanism is described in Chapter 7, "Instruction timing," in the *PowerPC 601 RISC Microprocessor User's Manual*."

To minimize latencies due to data dependencies, the IU provides feed-forwarding. For example, if an integer instruction requires data that is the result of the execution of the previous instruction, that data is made available to the IU at the same time that the previous instruction's write-back stage updates the GPR. This eliminates an additional clock cycle that would have been necessary if the IU had to access the GPR. Feed-forwarding is available between IU execute and decode stage and IU write-back and decode stage.



[1] An integer instruction can be passed to the ID stage in the same cycle in which it enters IQ0.

Figure 5. Pipeline Diagram of the Processor Core

Most integer instructions require one clock cycle per stage. Because results for most integer instructions are available at the end of the execute stage, a series of single-cycle integer instructions allow a throughput of one instruction per clock cycle. Other instructions, such as the integer multiply, require more than one clock cycle to complete execution. These instructions reduce the throughput accordingly.

The floating-point pipeline has more stages than the IU pipeline, as shown in Figure 5. The 601 supports both single- and double-precision floating-point operations, but double-precision instructions generally take longer to execute, typically by requiring two cycles in the FD, FPM, and FPA stages. However, many of these instructions, such as the double-precision floating-point multiply (**fmul**) and double-precision floating-point accumulate instructions (**fmadd**, **fmsub**, **fnmadd**, and **fnmsub**), allow stages to overlap. For example, when the second cycle of the FD stage begins, the first stage of FPM begins. Similarly the FPM stage overlaps with the FPA stage, allowing these instructions to complete these stages in four clock cycles instead of six.

Because the PowerPC architecture can be applied to such a wide variety of implementations, instruction timing among various PowerPC processors varies accordingly.

3.8 System Interface

The system interface is specific for each PowerPC processor implementation.

The 601 provides a versatile system interface that allows for a wide range of implementations. The interface includes a 32-bit address bus, a 64-bit data bus, and 52 control and information signals (see Figure 6). The system interface allows for address-only transactions as well as address and data transactions. The 601 control and information signals include the address arbitration, address start, address transfer, transfer attribute, address termination, data arbitration, data transfer, data termination, and processor state signals. Test and control signals provide diagnostics for selected internal circuitry.

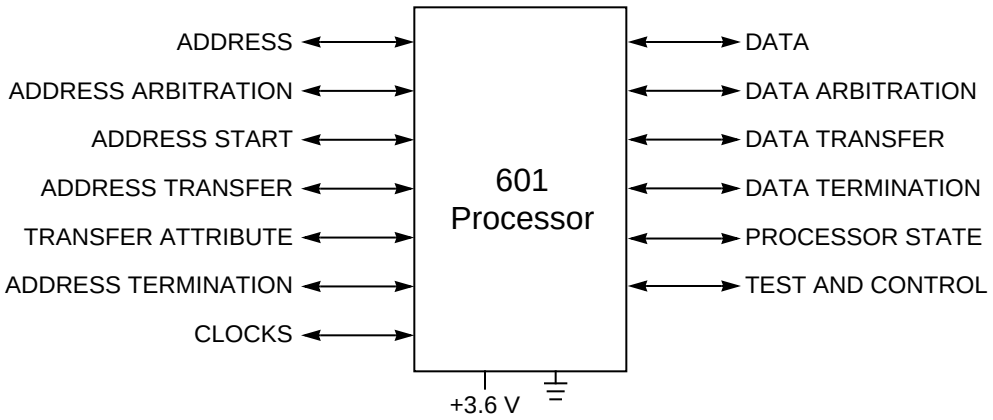


Figure 6. System Interface

The system interface supports bus pipelining, which allows the address tenure of one transaction to overlap the data tenure of another. The extent of the pipelining depends on external arbitration and control circuitry. Similarly, the 601 supports split-bus transactions for systems with multiple potential bus masters—one device can have mastership of the address bus while another has mastership of the data bus. Allowing multiple bus transactions to occur simultaneously increases the available bus bandwidth for other activity and as a result, improves performance.

The 601 supports multiple masters through a bus arbitration scheme that allows various devices to compete for the shared bus resource. The arbitration logic can implement priority protocols, such as fairness, and can park masters to avoid arbitration overhead. The MESI protocol ensures coherency among multiple devices and system memory. Also, the 601's on-chip cache and UTLB and optional second-level caches can be controlled externally.

The 601 clocking structure allows the bus to operate at integer multiples of the processor cycle time.

The following sections describe the 601 bus support for memory and I/O controller interface operations. Note that some signals perform different functions depending upon the addressing protocol used.

3.8.1 Memory Accesses

Memory accesses allow transfer sizes of 8, 16, 24, 32, 40, 48, 56, or 64 bits in one bus clock cycle. Data transfers occur in either single-beat transactions or four-beat burst transactions. A single-beat transaction transfers as much as 64 bits. Single-beat transactions are caused by noncached accesses that access memory directly (that is, reads and writes when caching is disabled, cache-inhibited accesses, and stores in write-through mode). Burst transactions, which always transfer an entire cache sector (32 bytes), are initiated when a sector in the cache is read from or written to memory. Additionally, the 601 supports address-only transactions used to invalidate entries in other processors' TLBs and caches.

3.8.2 I/O Controller Interface Operations

Both memory and I/O accesses can use the same bus transfer protocols. The 601 also has the ability to define memory areas as I/O controller interface areas. Accesses to the I/O controller interface redefine the function of some of the address transfer and transfer attribute signals and add control to facilitate transfers between the 601 and specific I/O devices that respond to this protocol. I/O controller interface transactions provide multiple transaction operations for variably-sized data transfers (1 to 128 bytes) and support a split request/response protocol. The distinction between the two types of transfers is made with separate signals—$\overline{\text{TS}}$ for memory-mapped accesses and $\overline{\text{XATS}}$ for I/O controller interface accesses. Refer to Chapter 9, "System Interface Operation," in the *PowerPC 601 RISC Microprocessor User's Manual* for more information.

3.8.3 PowerPC 601 Microprocessor Signals

The 601 signals are grouped as follows:

- Address arbitration signals—The 601 uses these signals to arbitrate for address bus mastership.
- Address transfer start signals—These signals indicate that a bus master has begun a transaction on the address bus.
- Address transfer signals—These signals, which consist of the address bus, address parity, and address parity error signals, are used to transfer the address and to ensure the integrity of the transfer.
- Transfer attribute signals—These signals provide information about the type of transfer, such as the transfer size and whether the transaction is bursted, write-through, or cache-inhibited.
- Address transfer termination signals—These signals are used to acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.
- Data arbitration signals—The 601 uses these signals to arbitrate for data bus mastership.
- Data transfer signals—These signals, which consist of the data bus, data parity, and data parity error signals, are used to transfer the data and to ensure the integrity of the transfer.

- Data transfer termination signals—Data termination signals are required after each data beat in a data transfer. In a single-beat transaction, the data termination signals also indicate the end of the tenure, while in burst accesses, the data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat. They also indicate whether a condition exists that requires the data phase to be repeated.
- System status signals—These signals include the interrupt signal, checkstop signals, and both soft- and hard-reset signals. These signals are used to interrupt and, under various conditions, to reset the processor.
- Processor state signals—These two signals are used to set the reservation coherency bit and set the size of the 601's output buffers.
- Miscellaneous signals—These signals provide information about the state of the reservation coherency bit.
- COP interface signals—The common on-chip processor (COP) unit is the master clock control unit and it provides a serial interface to the system for performing built-in self test (BIST).
- Test interface signals—These signals are used for internal testing.
- Clock signals—These signals determine the system clock frequency. These signals can also be used to synchronize multiprocessor systems.

NOTE

A bar over a signal name indicates that the signal is active low—for example, $\overline{\text{ARTRY}}$ (address retry) and $\overline{\text{TS}}$ (transfer start). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active-low, such as AP0–AP3 (address bus parity signals) and TT0–TT4 (transfer type signals) are referred to as asserted when they are high and negated when they are low.

3.8.4 Signal Configuration

Figure 7 illustrates the 601 microprocessor's logical pin configuration, showing how the signals are grouped.

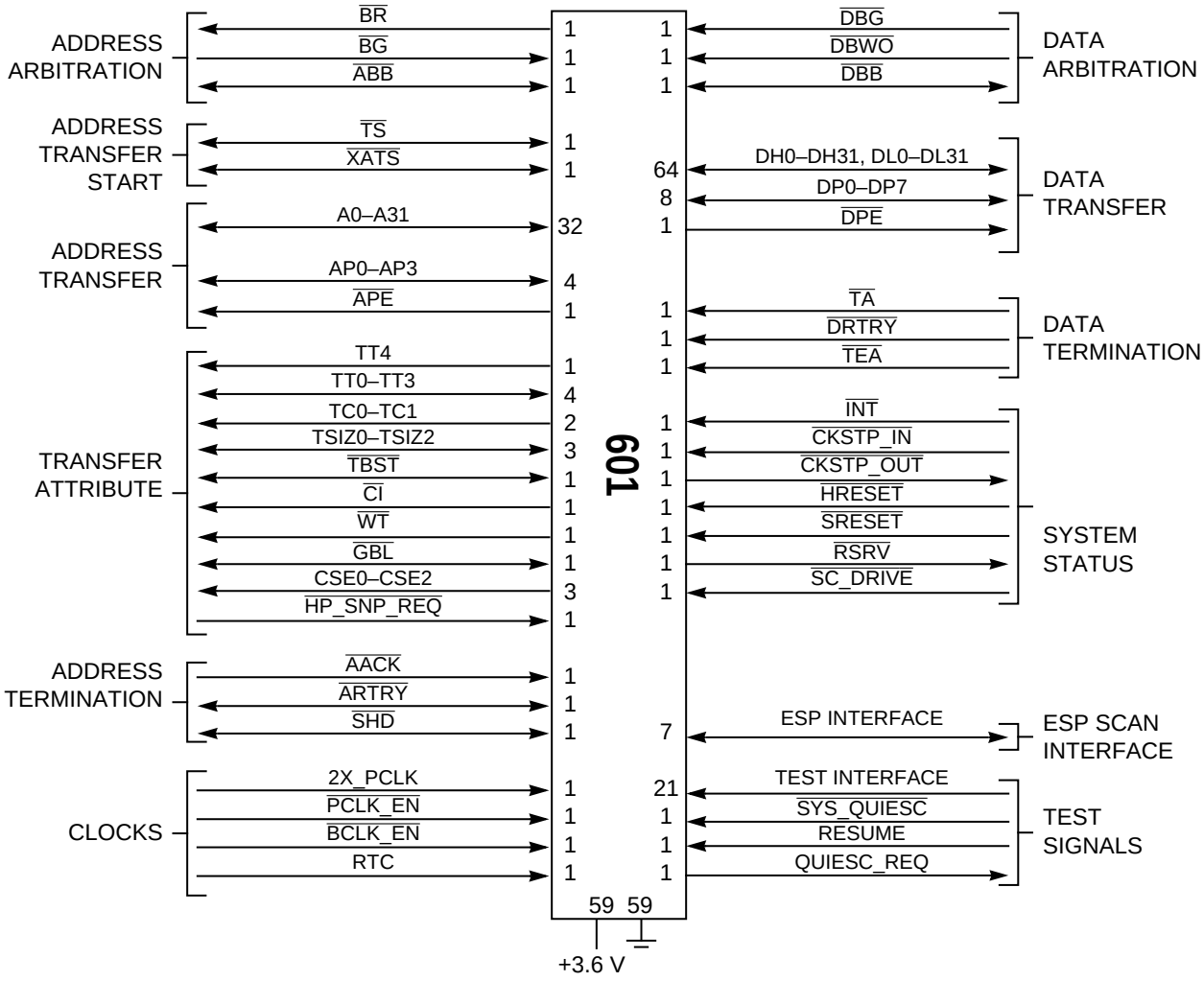


Figure 7. PowerPC 601 Microprocessor Signal Groups

3.8.4.1 Real-Time Clock

The real-time clock (RTC) facility, which is specific to the 601, provides a high-resolution measure of real time to provide time of day and date with a calendar range of 136.19 years. The RTC consists of two registers—the RTC upper (RTCU) register and the RTC lower (RTCL) register. The RTCU register maintains the number of seconds from a point in time specified by software. The RTCL register counts nanoseconds. The contents of either register may be copied to any GPR.

Motorola Literature Distribution Centers:
USA: Motorola Literature Distribution, P.O. Box 20912, Phoenix, Arizona 85036.
EUROPE: Motorola Ltd., European Literature Centre, 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd., 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.
ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd., Silicon Harbour Centre, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.
Technical Information: Motorola Inc. Semiconductor Products Sector Technical Responsiveness Center; (800) 521-6274.
Document Comments: FAX (512) 891-2638, Attn: RISC Applications Engineering.

IBM Microelectronics:
USA: IBM Microelectronics, Mail Stop A25/862-1, PowerPC Marketing, 1000 River Street, Essex Junction, VT 05452-4299; Tel.: (800) PowerPC [(800) 769-3772]; FAX (800) POWERfax [(800) 769-3732].
EUROPE: IBM Microelectronics, PowerPC Marketing, Dept. 1045, 224 Boulevard J.F. Kennedy, 91105 Corbeil-Essonnes CEDEX, France; Tel. (33) 1-60-88 5167; FAX (33) 1-60-88 4920.
JAPAN: IBM Microelectronics, PowerPC Marketing, Dept., R0260, 800 Ichimiyake, Yasu-cho, Yasu-gun, Shinga-ken, Japan 520-23; Tel. (81) 775-87-4745; FAX (81) 775-87-4735.

IBM Microelectronics

MOTOROLA