MPR620TSU-01
(IBM Order Number)

MPC620/D
(Motorola Order Number)

10/94

PowerPC™

Advance Information

PowerPC 620™ RISC Microprocessor Technical Summary

This document provides an overview of the PowerPC 620 microprocessor. It includes the following:

- Part 1, “Overview,” provides a summary of 620 features.
- Part 2, “PowerPC 620 Microprocessor Hardware Implementation,” provides details about the 620 hardware implementation. This includes descriptions of the 620’s execution units, cache implementation, memory management units (MMUs), and system interface.
- Part 3, “PowerPC 620 Microprocessor Execution Model,” provides a description of the 620 execution model. This section includes information about the programming model, instruction set, exception model, and instruction timing.
- Part 4, “Power Management and Performance Monitor,” discusses the power management feature and performance monitor facility of the 620.

In this document, the terms “PowerPC 620 Microprocessor” and “620” are used to denote a microprocessor from the PowerPC Architecture™ family. The PowerPC 620 microprocessors are available from IBM as PPC620 and from Motorola as MPC620.

IBM Microelectronics

MOTOROLA

Part 1 Overview

This section describes the features of the 620, provides a block diagram showing the major functional units, and describes briefly how those units interact.

The 620 is an implementation of the PowerPC™ family of reduced instruction set computer (RISC) microprocessors. The 620 implements the PowerPC architecture as it is specified for 64-bit addressing, which provides 64-bit effective (logical) addresses, integer data types of 8, 16, 32, and 64 bits, and floating-point data types of 32 and 64 bits (single-precision and double-precision). The 620 is software compatible with the 32-bit versions of the PowerPC microprocessor family.

The 620 is a superscalar processor capable of issuing four instructions simultaneously. As many as six instructions can finish execution in parallel. The 620 has six execution units that can operate in parallel:

- Floating-point unit (FPU)
- Branch processing unit (BPU)
- Load/store unit (LSU)
- Three integer units (IUs):
  - — Two single-cycle integer units (SCIUs)
  - — One multiple-cycle integer unit (MCIU)

This parallel design, combined with the PowerPC architecture's specification of uniform instructions that allows for rapid execution times, yields high efficiency and throughput. The 620's rename buffers, reservation stations, dynamic branch prediction, and completion unit increase instruction throughput, guarantee in-order completion, and ensure a precise exception model. (Note that the PowerPC architecture specification refers to all exceptions as interrupts.)

The 620 has separate memory management units (MMUs) and separate 32-Kbyte on-chip caches for instructions and data. The 620 implements a 128-entry, two-way set-associative translation lookaside buffer (TLB) for instructions and data, and provides support for demand-paged virtual memory address translation and variable-sized block translation. The TLB and the cache use least-recently used (LRU) replacement algorithms.

The 620 has a 40-bit address bus, and can be configured with either a 64- or 128-bit data bus. The 620 interface protocol allows multiple masters to compete for system resources through a central external arbiter. Additionally, on-chip snooping logic maintains data cache coherency for multiprocessor applications. The 620 supports single-beat and burst data transfers for memory accesses and memory-mapped I/O accesses.

The 620 uses an advanced, 3.3-V CMOS process technology and is compatible with 3.3-V CMOS devices.

1.1 PowerPC 620 Microprocessor Features

This section summarizes features of the 620's implementation of the PowerPC architecture.

Figure 1 provides a block diagram showing features of the 620. Note that this is a conceptual block diagram intended to show the basic features rather than an attempt to show how these features are physically implemented on the chip.

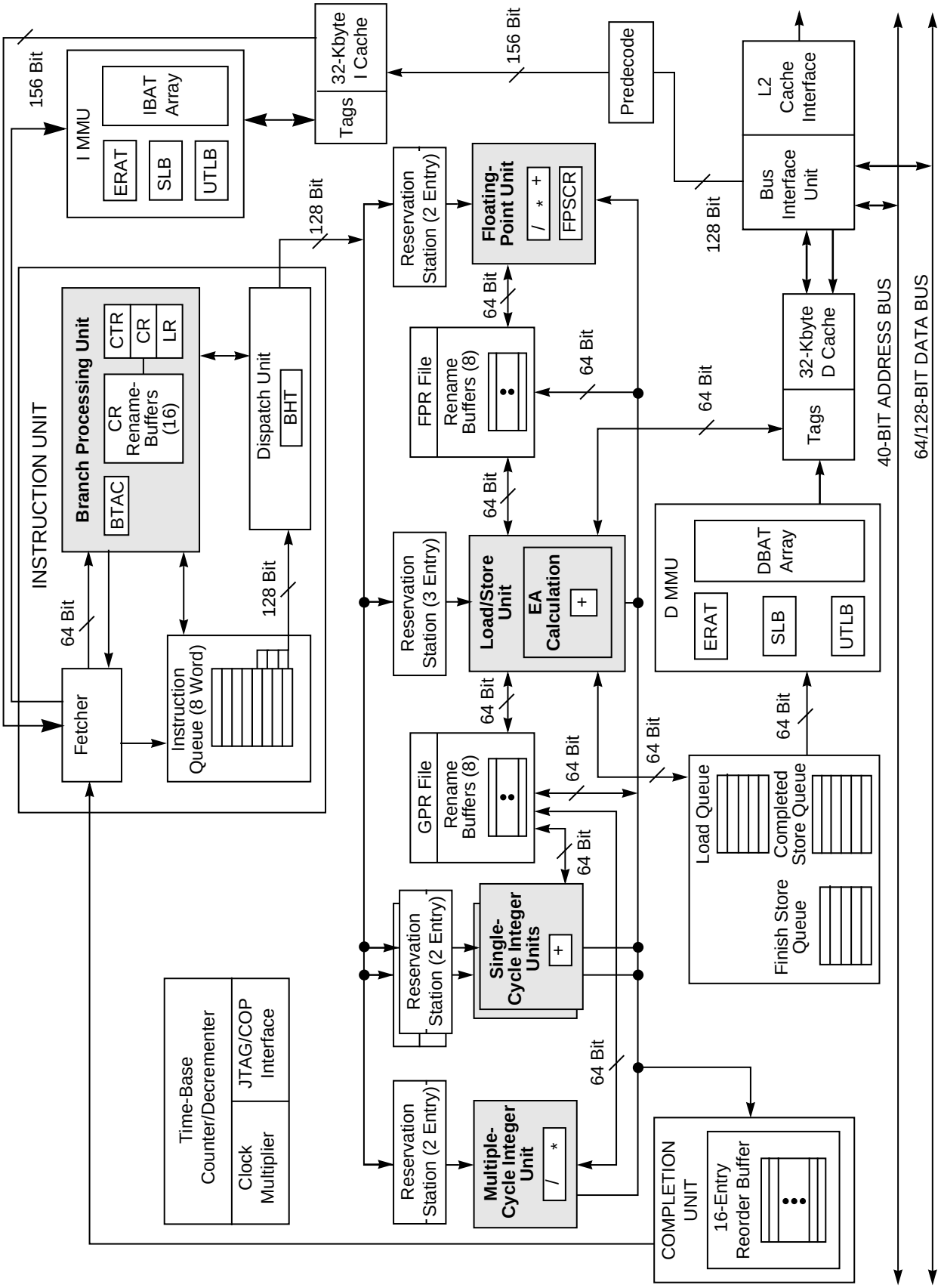


Figure 1. Block Diagram

Major features of the 620 are as follows:

- High-performance, superscalar microprocessor
  - — As many as four instructions can be issued per clock.
  - — As many as six instructions can start executing per clock (including three integer instructions).
  - — Single clock cycle execution for most instructions.
- Six independent execution units and two register files
  - — BPU featuring dynamic branch prediction
    - – Speculative execution through four branches
    - – 256-entry fully-associative branch target address cache (BTAC)
    - – 2048-entry branch history table (BHT) with two bits per entry indicating four levels of prediction—not-taken, strongly not-taken, taken, strongly taken
  - — Two single-cycle IUs (SCIUs) and one multiple-cycle IU (MCIU)
    - – Instructions that execute in the SCIU take one cycle to execute; most instructions that execute in the MCIU take multiple cycles to execute.
    - – Each SCIU has a two-entry reservation station to minimize stalls.
    - – The MCIU has a two-entry reservation station and provides early exit (three cycles) for 16 x 32-bit and overflow operations.
    - – Thirty-two GPRs for integer operands.
    - – Eight rename buffers for GPRs.
  - — Three-stage floating-point unit (FPU)
    - – Fully IEEE 754-1985 compliant FPU for both single- and double-precision operations
    - – Supports non-IEEE mode for time-critical operations
    - – Fully pipelined, single-pass double-precision design
    - – Hardware support for denormalized numbers
    - – Two-entry reservation station to minimize stalls
    - – Thirty-two 64-bit FPRs for single- or double-precision operands
    - – Eight rename buffers for FPRs
  - — Load/store unit (LSU)
    - – Three-entry reservation station to minimize stalls
    - – Single-cycle, pipelined cache access
    - – Dedicated adder that performs EA calculations
    - – Performs alignment and precision conversion for floating-point data
    - – Performs alignment and sign extension for integer data
    - – Five-entry pending load queue that provides load/store address collision detection
    - – Five-entry finished store queue
    - – Six-entry completed store queue
    - – Supports both big- and little-endian modes

- Rename buffers
  - Eight GPR rename buffers
  - Eight FPR rename buffers
  - Sixteen condition register (CR) rename buffers

  The 620 rename buffers are described in Section 2.1.6, “Rename Buffers.”
- Completion unit
  - Retires an instruction from the 16-entry reorder buffer when all instructions ahead of it have been completed and the instruction has finished execution
  - Guarantees sequential programming model (precise exception model)
  - Monitors all dispatched instructions and retires them in order
  - Tracks unresolved branches and removes speculatively executed, dispatched, and fetched instructions if branch is mispredicted
  - Retires as many as four instructions per clock
- Separate on-chip instruction and data caches (Harvard architecture)
  - 32-Kbyte, eight-way set-associative instruction and data caches; data cache is 2-way interleaved
  - LRU replacement algorithm
  - 64-byte (sixteen word) cache block size
  - Physically indexed; physical tags
  - Cache write-back or write-through operation programmable on a per page or per block basis
  - Instruction cache can provide four instructions per clock; data cache can provide two words per clock.
  - Caches can be disabled in software
  - Parity checking performed on both caches
  - Data cache coherency (MESI) maintained in hardware
  - Interprocessor broadcast of cache control instructions
  - Instruction cache coherency maintained in software
- On-chip L2 cache interface
  - L2 cache is a unified instruction and data secondary cache with ECC.
  - L2 cache is direct mapped, physically indexed, and physically tagged.
  - L2 data cache is inclusive of L1; L2 instruction cache is not inclusive of L1.
  - L2 cache capacity is configurable from 1 Mbyte to 128 Mbyte.
  - Independent user-configurable PLL provides L2 interface clock.
  - L2 cache interface supports GTL or CMOS SRAMs.
- Separate memory management units (MMUs) for instructions and data
  - Address translation facilities for 4-Kbyte page size, variable block size, and 256-Mbyte segment size
  - Independent 64-entry fully-associative effective-to-real address translation (ERAT) cache with invalid-first replacement algorithm for instructions and data
  - Unified instruction and data translation lookaside buffer (TLB)

  - — TLB is 128-entry and two-way set-associative.
  - — 20-entry CAM segment lookaside buffer (SLB) with FIFO replacement algorithm
  - — Sixteen segment registers that provide support for 32-bit memory management
  - — SLB, TLB, and ERAT cache miss handling performed by 620 hardware
  - — Hardware update of page frame table reference and change bits
  - — Hardware broadcast of TLB and control instructions
  - — Separate IBATs and DBATs (four each) also defined as SPRs
  - — 64-bit effective addressing
  - — 80-bit virtual addressing
  - — 40-bit physical memory address for up to one terabyte
- Bus interface features include the following:
  - — Selectable processor-to-bus clock frequency ratios (2:1, 3:1, and 4:1)
  - — A 64- and 128-bit split-transaction external data bus with burst transfers
  - — Explicit address and data bus tagging
  - — Pended (split) read protocol
  - — Pipelined snoop response, fixed boot-time latency
  - — 620 bus is crossbar compatible.
  - — Additional signals and signal redefinition for direct-store operations
- Multiprocessing support features include the following:
  - — Hardware enforced, four-state cache coherency protocol (MESI) for data cache. Bits are provided in the instruction cache to indicate only whether a cache block is valid or invalid.
  - — Data cache coherence for L1, L2, and external L3 cache is fully supported by 620 hardware.
  - — Snoop operations take priority over processor access to L1 and L2 cache.
  - — Instruction cache coherence is software controlled.
  - — Load/store with reservation instruction pair provided for atomic memory references, semaphores, and other multiprocessor operations.
- Power management
  - — Doze mode allows bus snooping activity to continue while the 620 is in a reduced power state. When a wake-up signal is asserted, the 620 performs snoop activities, and after completing snoop responses (cache block state changes, bus intervention, or cache block pushes) automatically returns to the power saving doze state.
  - — Nap mode supports full shut down of the 620.
  - — Operating voltage is 3.3 ± 0.3 V.
- Performance monitor can be used to help in debugging system designs and improving software efficiency, especially in multiprocessor systems.
- In-system testability and debugging features provided through JTAG boundary-scan capability.

Part 2 PowerPC 620 Microprocessor Hardware Implementation

This section provides an overview of the 620's hardware implementation, including descriptions of the functional units, shown in Figure 2, the cache implementation, MMU, and the system interface.

Note that Figure 2 provides a more detailed block diagram than that presented in Figure 1—showing the additional data paths that contribute to the improved efficiency in instruction execution and more clearly indicating the relationships between execution units and their associated register files.

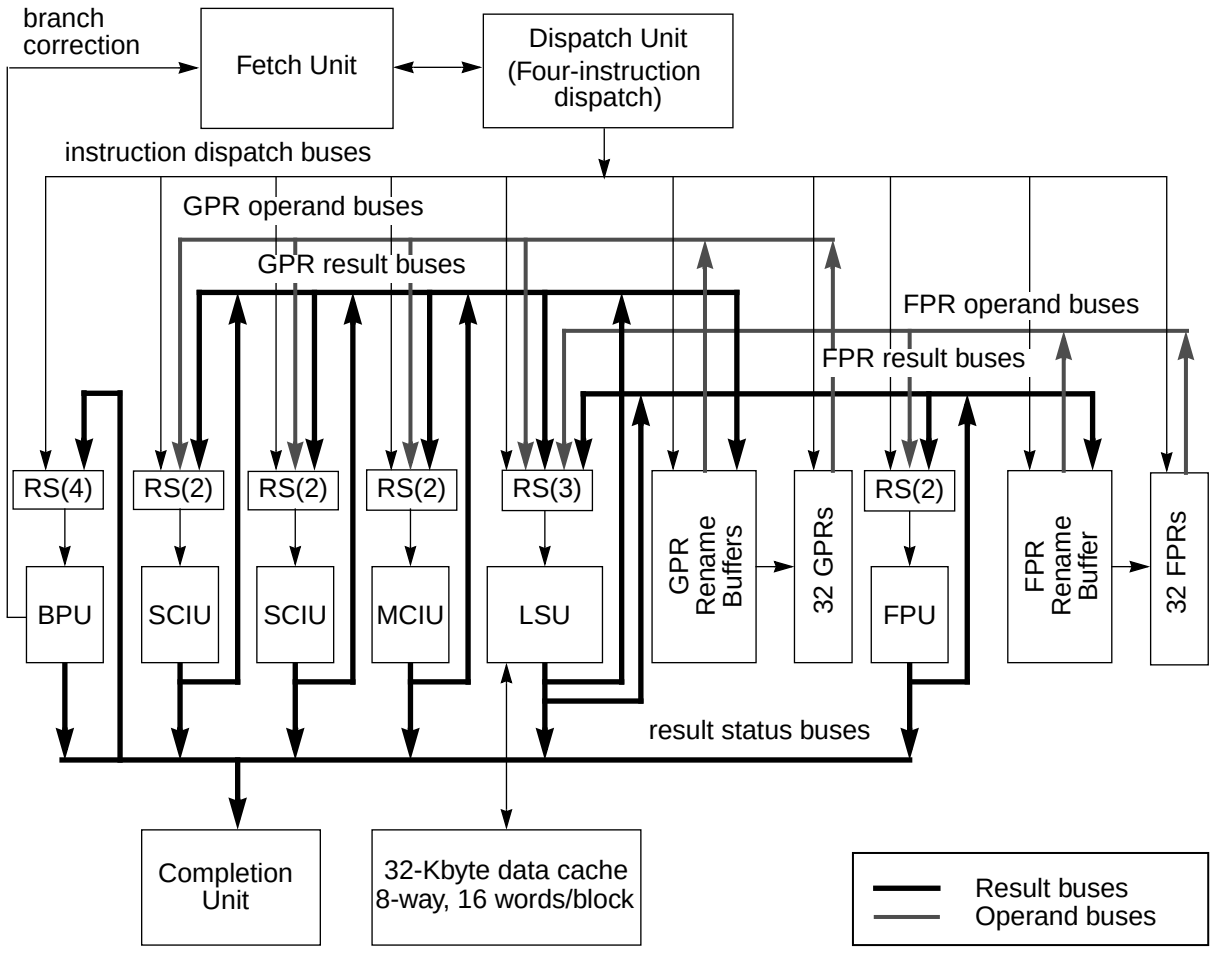


Figure 2. Block Diagram—Internal Data Paths

2.1 Instruction Flow

Several units on the 620 ensure the proper flow of instructions and operands and guarantee the correct update of the architectural machine state. These units include the following:

- Predecode unit—Provides logic to decode instructions and determine what resources are required for execution.
- Fetch unit—Using the next sequential address or the address supplied by the BPU when a branch is predicted or resolved, the fetch unit supplies instructions to the eight-word instruction buffer.

- Dispatch unit—The dispatch unit dispatches instructions to the appropriate execution unit. During dispatch, operands are provided to the execution unit (or reservation station) from the register files, rename buffers, and result buses.
- Branch processing unit (BPU)—In addition to providing the fetcher with predicted target instructions when a branch is predicted (and a mispredict-recovery address if a branch is incorrectly predicted), the BPU executes all condition register logical and flow control instructions.
- Completion unit—The completion unit retires executed instructions in program order and controls the updating of the architectural machine state.

2.1.1 Predecode Unit

The instruction predecode unit provides the logic to decode the instructions and categorize the resources that will be used, source operands, destination registers, execution registers, and other resources required for execution. The instruction stream is predecoded on its way from the bus interface unit to the instruction cache.

2.1.2 Fetch Unit

The fetch unit provides instructions to the four-entry instruction queue by accessing the on-chip instruction cache. Typically, the fetch unit continues fetching sequentially as many as four instructions at a time.

The address of the next instruction to be fetched is determined by several conditions, which are prioritized as follows:

1. Detection of an exception. Instruction fetching begins at the exception vector.
2. The BPU recovers from an incorrect prediction when a branch instruction is in the execute stage. Undispatched instructions are flushed and fetching begins at the correct target address.
3. The BPU recovers from an incorrect prediction when a branch instruction is in the dispatch stage. Undispatched instructions are flushed and fetching begins at the correct target address.
4. A fetch address is found in the BTAC. As a cache block is fetched, the branch target address cache (BTAC) and the branch history table (BHT) are searched with the fetch address. If it is found in the BTAC, the target address from the BTAC is the first candidate for being the next fetch address.
5. If none of the previous conditions exist, the instruction is fetched from the next sequential address.

2.1.3 Dispatch Unit

The dispatch unit provides the logic for dispatching the predecoded instructions to the appropriate execution unit. For many branch instructions, these decoded instructions along with the bits in the BHT, are used during the decode stage for branch correction. The dispatch logic also resolves unconditional branch instructions and predicts conditional branch instructions using the branch decode logic, BHT, and values in the CTR.

The 2048-entry BHT provides two bits per entry, indicating four levels of dynamic prediction—strongly not-taken, not-taken, taken, and strongly taken. The history of a branch's direction is maintained in these two bits. Each time a branch is taken, the value is incremented (with a maximum value of three meaning strongly-taken); when it is not taken, the bit value is decremented (with a minimum value of zero meaning strongly not-taken). If the current value predicts taken and the next branch is taken again, the BHT entry then predicts strongly taken. If the next branch is not taken, the BHT then predicts taken.

The dispatch logic also allocates each instruction to the appropriate execution unit. A reorder buffer (ROB) entry in the completion unit is allocated for each instruction, and dependency checking is done between the instructions in the dispatch queue. The rename buffers are searched for the operands as the operands are fetched from the register file. Operands that are written by other instructions ahead of this one in the dispatch queue are given the tag of that instruction's rename buffer; otherwise, the rename buffer or register file supplies either the operand or a tag. As instructions are dispatched, the fetch unit is notified that the dispatch queue can be updated with more instructions.

2.1.4 Branch Processing Unit (BPU)

The BPU is used for branch instructions and condition register logical operations. All branches, including unconditional branches, are placed in reservation stations until conditions are resolved and they can be executed. At that point, branch instructions are executed in order—the completion unit is notified whether the prediction was correct.

The BPU also executes condition register logical instructions, which flow through the reservation station like the branch instructions.

2.1.5 Completion Unit

The completion unit retires executed instructions from the reorder buffer (ROB) in the completion unit and updates register files and control registers. The completion unit recognizes exception conditions and discards any operations being performed on subsequent instructions in program order. The completion unit can quickly remove instructions from a mispredicted branch, and the dispatch unit begins dispatching from the correct path.

The instruction is retired from the reorder buffer when it has finished execution and all instructions ahead of it have been completed. The instruction's result is written into the appropriate register file and is removed from the rename buffers at or after completion. At completion, the 620 also updates any other resource affected by this instruction. Several instructions can complete simultaneously. Most exception conditions are recognized at completion time.

2.1.6 Rename Buffers

To avoid contention for a given register location, the 620 provides rename registers for storing instruction results before the completion unit commits them to the architected register. Eight rename registers are provided for the GPRs, eight for the FPRs, and eight for the condition register. GPRs are described in Section 3.2.1, "General-Purpose Registers (GPRs)," FPRs are described in Section 3.2.2, "Floating-Point Registers (FPRs)," and the condition register is described in Section 3.2.3, "Condition Register (CR)."

When the dispatch unit dispatches an instruction to its execution unit, it allocates a rename register for the results of that instruction. The dispatch unit also provides a tag to the execution unit identifying the result that should be used as the operand. When the proper result is returned to the rename buffer, it is latched into the reservation station. When all operands are available in the reservation station, execution can begin.

The completion unit does not transfer instruction results from the rename registers to the registers until any speculative branch conditions preceding it in the completion queue are resolved and the instruction itself is retired from the completion queue without exceptions. If a speculatively executed branch is found to have been incorrectly predicted, the speculatively executed instructions following the branch are flushed from the completion queue and the results of those instructions are flushed from the rename registers.

2.2 Execution Units

The following sections describe the 620's arithmetic execution units— two single-cycle IUs, multiple cycle IU, and FPU. When the reservation station sees the proper result being written back, it will grab it directly from one of the result buses. Once all operands are in the reservation station for an instruction, it is eligible to be executed. Reservation stations temporarily store dispatched instructions that cannot be executed until all of the source operands are valid.

2.2.1 Integer Units (IUs)

The two single-cycle IUs (SCIUs) and one multiple-cycle IU (MCIU) execute all integer instructions. These are shown in Figure 1 and Figure 2. The results generated by the IUs are put on the result buses that are connected to the appropriate reservation stations and rename buffers. Each IU has a two-entry reservation station to reduce stalls. The reservation station can receive instructions from the dispatch unit and operands from the GPRs, the rename buffers, or the result buses.

Each SCIU consists of three single-cycle subunits—a fast adder/comparator, a subunit for logical operations, and a subunit for performing rotates, shifts, and count-leading-zero operations. These subunits handle all one-cycle arithmetic instructions; only one subunit can execute an instruction at a time.

The MCIU consists of a 64-bit integer multiplier/divider. The MCIU executes **mfspr** and **mtspr** instructions, which are used to read and write special-purpose registers. The MCIU can execute an **mtspr** or **mfspr** instruction at the same time that it executes a multiply or divide instruction. These instructions are allowed to complete out of order.

2.2.2 Floating-Point Unit (FPU)

The FPU, shown in Figure 1 and Figure 2, is a single-pass, double-precision execution unit; that is, both single- and double-precision operations require only a single pass, with a latency of three cycles.

As the dispatch unit issues instructions to the FPU's two reservation stations, source operand data may be accessed from the FPRs, the floating-point rename buffers, or the result buses. Results in turn are written to the floating-point rename buffers and to the reservation stations and are made available to subsequent instructions. The three reservation stations provided by the FPU support out-of-order execution of floating-point instructions.

2.2.3 Load/Store Unit (LSU)

The LSU, shown in Figure 1 and Figure 2, transfers data between the data cache and the result buses, which route data to other execution units. The LSU supports the address generation and handles any alignment for transfers to and from system memory. The LSU also supports cache control instructions and load/store multiple/string instructions.

The LSU includes a 64-bit adder dedicated for EA calculation. Data alignment logic manipulates data to support aligned or misaligned transfers with the data cache. The LSU's load and store queues are used to buffer instructions that have been executed and are waiting to be completed. The queues are used to monitor data dependencies generated by data forwarding and out-of-order instruction execution ensuring a sequential model.

The LSU allows load instructions to precede store instructions in the reservation stations. Data dependencies resulting from the out-of-order execution of loads before stores to addresses with the same low-order 12 bits in the effective address are resolved when the store instruction is completed. If an out-of-order load operation is found to have an address that matches a previous store, the instruction pipeline is flushed, and the load instruction will be refetched and re-executed.

The LSU does not allow the following operations to be speculatively performed on unresolved branches:

- Store operations
- Loading of noncacheable data or cache miss operations
- Loading from direct-store segments

2.3 Memory Management Units (MMUs)

The primary functions of the MMUs 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 direct-store accesses, and to provide access protection on blocks and pages of memory.

The PowerPC MMUs 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.

Address translations are enabled by setting bits in the MSR—MSR[IR] enables instruction address translations and MSR[DR] enables data address translations.

The 620's MMUs support up to one heptabyte (2^{80}) of virtual memory and one terabyte (2^{40}) of physical memory. The MMUs support block address translations, direct-store segments, and page translation of memory segments. Referenced and changed status are maintained by the processor for each page to assist implementation of a demand-paged virtual memory system.

Separate but identical translation logic is implemented for data accesses and for instruction accesses. The 620 implements a two-stage translation mechanism; the first stage consists of independent 64-entry content addressable address translation caches for instructions and data, and the second stage consists of a shared 128-entry, two-way set-associative translation lookaside buffer (TLB). If a TLB miss occurs during the second-stage address translation, memory segment lookup is assisted by a 20-entry content addressed segment lookaside buffer (SLB). Sixteen segment registers are provided by binary decode of 16 of the 20 SLBs for the execution of software compiled for 32-bit PowerPC microprocessors.

2.4 Cache Implementation

The PowerPC architecture does not define hardware aspects of cache implementations. For example, whereas the 620 implements separate data and instruction caches (Harvard architecture), other processors may use a unified cache, or no cache at all. The PowerPC architecture defines the unit of coherency as a cache block, which for the 620 is a 64-byte (sixteen-word) line.

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
- Guarded memory (prevents access for speculative execution)

2.4.1 Instruction Cache

The 620's 32-Kbyte, eight-way set-associative instruction cache is physically indexed. Within a single cycle, the instruction cache provides up to four instructions. Instruction cache coherency is not maintained by hardware.

The PowerPC architecture defines a special set of instructions for managing the instruction cache. The instruction cache can be invalidated entirely or on a cache-block basis. The instruction cache can be disabled and invalidated by setting the HID0[16] and HID0[20] bits, respectively.

2.4.2 Data Cache

The 620's data cache is a 32-Kbyte, eight-way set-associative cache. It is a physically-indexed, nonblocking, write-back cache with hardware support for reloading on cache misses. Within one cycle, the data cache provides double-word access to the LSU.

The data cache tags are dual-ported, so the process of snooping does not affect other transactions on the system interface. If a snoop hit occurs in the same cache set as a load or store access, the LSU is blocked internally for one cycle to allow the 16-word block of data to be copied to the write-back buffer.

The 620 data cache supports the four-state MESI (modified/exclusive/shared/invalid) protocol to ensure cache coherency.

These four states indicate the state of the cache block as follows:

- Modified (M)—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 (E)—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 (S)—This cache block holds valid data that is identical to this address in system memory and at least one other caching device.
- Invalid (I)—This cache block does not hold valid data.

Like the instruction cache, the data cache can be invalidated all at once or on a per cache block basis. The data cache can be disabled and invalidated by setting the HID0[17] and HID0[21] bits, respectively.

Each cache line contains 16 contiguous words from memory that are loaded from a 16-word boundary (that is, bits A58–A63 of the logical addresses are zero); thus, a cache line never crosses a page boundary. Accesses that cross a page boundary can incur a performance penalty.

The organization of the cache is shown in Figure 3.

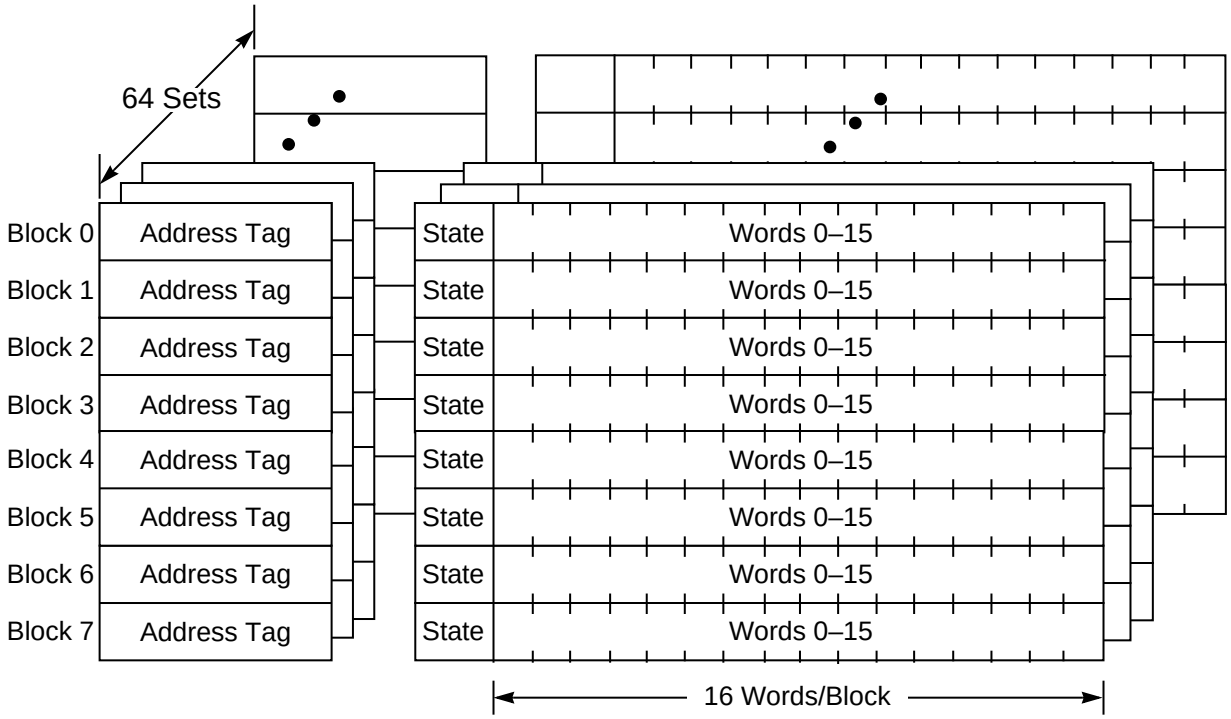


Figure 3. Cache Unit Organization

2.5 Level 2 Cache Interface

The 620 provides an integrated L2 cache controller that supports L2 configurations from 1 Mbyte to 128 Mbyte, using the same block size (64 bytes) as the internal L1 caches. The 620's L2 cache is a direct-mapped, error-correction-code (ECC) protected, unified instruction and data secondary cache that supports the use of single- and double-register synchronous static RAMs. The L2 cache interface supports a wide variety of static RAM access speeds by means of a boot-time configurable subsynchronous interface that is configurable for either CMOS or GTL logic levels. An external coprocessor can also be connected to the 620 through the L2 cache interface.

The L2 cache interface generates 9 bits of ECC for the 128 bits of data in a cache block, and 6 bits of ECC for the tag and coherency state of the block. The ECC allows the correction of single-bit errors, and the detection of double-bit errors. Uncorrectable errors detected by the L2 cache interface will generate a machine check exception. The ECC capability of the L2 cache interface can be configured in three modes; always-corrected mode, never-corrected mode, and automatic mode. In always-corrected mode, ECC is generated for write operations, and always corrected on read operations, resulting in constant L2 read access latency. In never-corrected mode, ECC generation, checking, and correction are disabled. In the automatic mode, ECC is generated during write operations, and read operations are corrected only when errors are detected, thereby increasing read latency only when correctable errors are detected.

2.6 System Interface/Bus Interface Unit (BIU)

The 620 provides a versatile bus interface that allows a wide variety of system design options. The interface includes a 144-bit data bus (128 bits of data and 16 bits of parity), a 43-bit address bus (40 bits of address and 3 bits of parity), and sufficient control signals to allow for a variety of system-level optimizations. The 620 uses one-beat, four-beat, and eight-beat data transactions (depending on whether the 620 is configured with a 64- or 128-bit data bus), although it is possible for other bus participants to perform longer data transfers. The 620 clocking structure supports processor-to-bus clock ratios of 2:1, 3:1, and 4:1 as described in Section 2.7, “Clocking.”

The system interface is specific for each PowerPC processor implementation. The 620 system interface is shown in Figure 4.

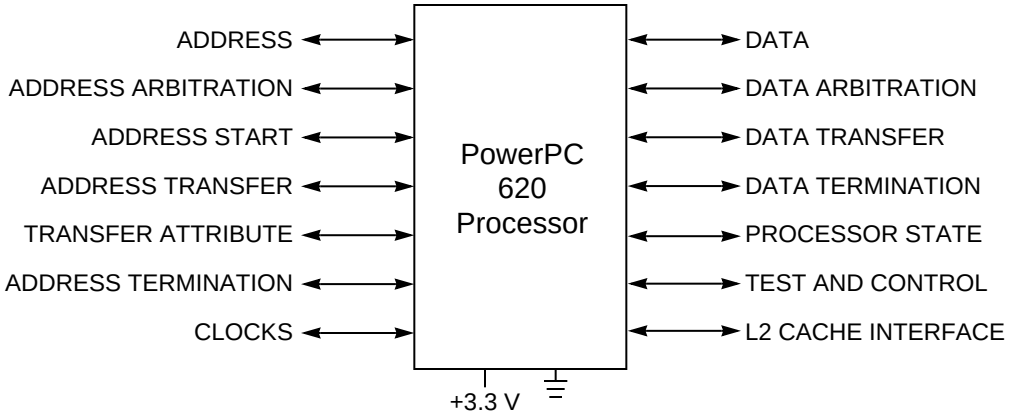


Figure 4. System Interface

Four-beat (or eight-beat, if in 64-bit data bus mode) burst-read memory operations that load a 16-word cache block into one of the on-chip caches are the most common bus transactions in typical systems, followed by burst-write memory operations, direct-store operations, and single-beat (noncacheable or write-through) memory read and write operations. Additionally, there can be address-only operations, data-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.

Memory accesses can occur in single-beat or four-beat burst data transfers. The address and data buses are independent for memory accesses to support pipelining and split transactions, and all bus operations are explicitly tagged through the use of 8-bit tags for addresses and data. The 620 supports bus pipelining and out-of-order split-bus transactions.

Typically, memory accesses are weakly-ordered. Sequences of operations, including load and store string/multiple instructions, do not necessarily complete in the same order in which they began—maximizing the efficiency of the bus without sacrificing coherency of the data. The 620 allows load operations to precede store operations (except when a dependency exists, of course). In addition, the 620 provides a separate queue for snoop push operations so these operations can access the bus ahead of previously queued operations. The 620 dynamically optimizes run-time ordering of load/store traffic to improve overall performance.

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 620 to be integrated into systems that use 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 caches and TLBs, and support for a secondary cache. The PowerPC architecture provides the load/store with reservation instruction pair (**lwarx**/**stwcx.**) for atomic memory references and other operations useful in multiprocessor implementations.

The following sections describe the 620 bus support for memory and direct-store operations. Note that some signals perform different functions depending upon the addressing protocol used.

2.6.1 Memory Accesses

Memory accesses allow transfer sizes of 8, 16, 24, 32, 64, or 128 bits in one bus clock cycle. Data transfers occur in either single-beat, four-beat, or eight-beat burst transactions. A single-beat transaction transfers as much as 128 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 block (64 bytes), are initiated when a block in the cache is read from or written to memory. Additionally, the 620 supports address-only transactions used to invalidate entries in other processors' TLBs and caches, and data-only transactions in which modified data is provided by a snooping device during a read operation to both the bus master and the memory system.

Typically I/O accesses are performed using the same protocol as memory accesses.

2.6.2 Signals

The 620's signals are grouped as follows:

- Address arbitration signals—The 620 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 620 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 reset and hard reset signals. These signals are used to interrupt and, under various conditions, to reset the processor.

- Processor state signal—This signal is used to indicate the state of the reservation coherency bit.
- Miscellaneous signals—These signals are used in conjunction with such resources as secondary caches and the time base facility.
- 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) on all internal memory arrays.
- 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, $\overline{\text{DBG}}$ (data bus grant) and $\overline{\text{EATS}}$ (early address 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–AP2 (address bus parity signals) and DT0–DT7 (data tag signals) are referred to as asserted when they are high and negated when they are low.

2.6.3 Signal Configuration

Figure 5 illustrates the logical pin configuration of the 620, showing how the signals are grouped.

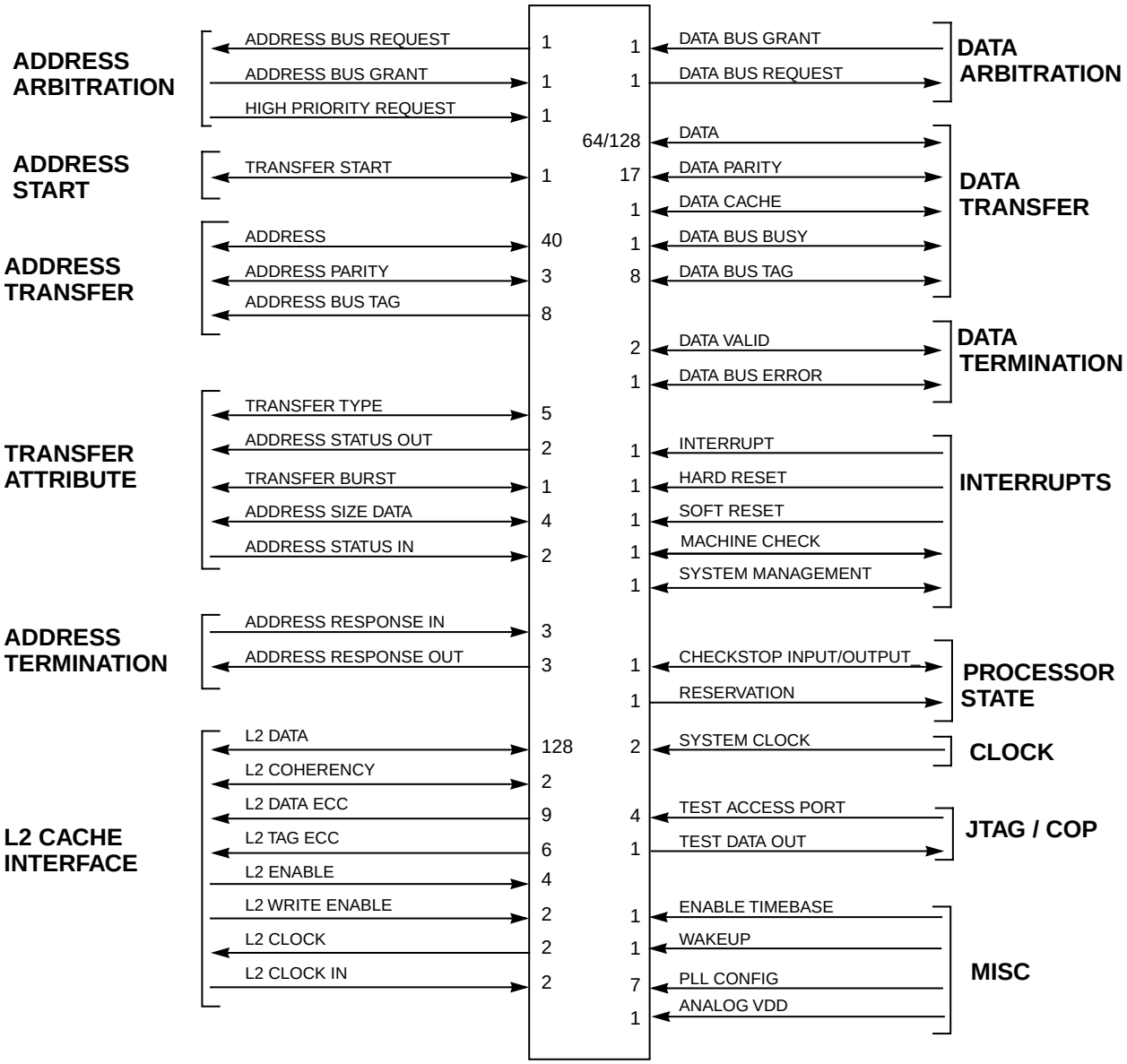


Figure 5. PowerPC 620 Microprocessor Signal Groups

2.7 Clocking

The 620 has a phase-locked loop (PLL) that generates the internal processor clock. The input, or reference signal, to the PLL is the bus clock. The feedback in the PLL guarantees that the processor clock is phase locked to the bus clock, regardless of process variations, temperature changes, or parasitic capacitances. The PLL also ensures a 50% duty cycle for the processor clock.

The 620 supports the following processor-to-bus clock frequency ratios—2:1, 3:1, and 4:1, although not all ratios are available for all frequencies. Table 1 shows the supported processor frequencies for different bus frequencies.

Table 1. Supported Processor/Bus Frequency Ratios

Bus Frequency (MHz)	Supported Processor/Bus Clock Ratios		
	2:1	3:1	4:1
9.4–14.0	No	No	Yes
12.5–18.8	No	Yes	Yes
16.7–25.0	No	Yes	No
18.8–28.0	Yes	No	Yes
25.0–37.5	Yes	Yes	Yes
33.3–50.0	No	Yes	No
37.5–56.3	Yes	No	No
50.0–75.0	Yes	No	No

Part 3 PowerPC 620 Microprocessor Execution Model

This section describes the following characteristics of the 620's execution model:

- The PowerPC architecture
- The 620 register set and programming model
- The 620 instruction set
- The 620 exception model
- Instruction timing on the 620

3.1 Levels of the PowerPC Architecture

The PowerPC architecture is derived from the IBM POWER Architecture™ (Performance Optimized with Enhanced RISC 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.

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. For example, if a processor adheres to the virtual environment architecture, it is assumed that it meets the user instruction set architecture specification.

- PowerPC user instruction set architecture (UISA)—The UISA defines the level of the architecture to which user-level software must conform. The UISA defines the base user-level instruction set, user-level registers, data types, memory conventions, and the memory and programming models seen by application programmers. Note that the PowerPC architecture refers to user level as problem state.

- PowerPC virtual environment architecture (VEA)—The VEA, which is the smallest component of the PowerPC architecture, defines additional user-level functionality that falls outside typical user-level software requirements. The VEA describes the memory model for an environment in which multiple processors or other devices can access external memory, and defines aspects of the cache model and cache control instructions from a user-level perspective. The resources defined by the VEA are particularly useful for managing resources in an environment in which other processors and other devices can access external memory.

  Implementations that conform to the PowerPC VEA also adhere to the UISA, but may not necessarily adhere to the OEA.

- PowerPC operating environment architecture (OEA)—The OEA defines supervisor-level resources typically required by an operating system. The OEA defines the PowerPC memory management model, supervisor-level registers, and the exception model. Note that the PowerPC architecture refers to the supervisor level as privileged state.

  Implementations that conform to the PowerPC OEA also conform to the PowerPC UISA and VEA.

The 620 complies with all three levels of the PowerPC architecture. Note that the PowerPC architecture defines additional instructions for 64-bit data types. PowerPC processors are allowed to have implementation-specific features that fall outside, but do not conflict with, the PowerPC architecture specification. Examples of features that are specific to the 620 include the performance monitor and nap mode.

The 620 is a high-performance, superscalar PowerPC implementation of the PowerPC architecture. Like other PowerPC processors, it adheres to the PowerPC architecture specifications but also has additional features not defined by the architecture. These features do not affect software compatibility. 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 in the 620 allow compilers to maximize parallelism and instruction throughput. Compilers that take advantage of the flexibility of the PowerPC architecture can additionally optimize instruction processing of the PowerPC processors.

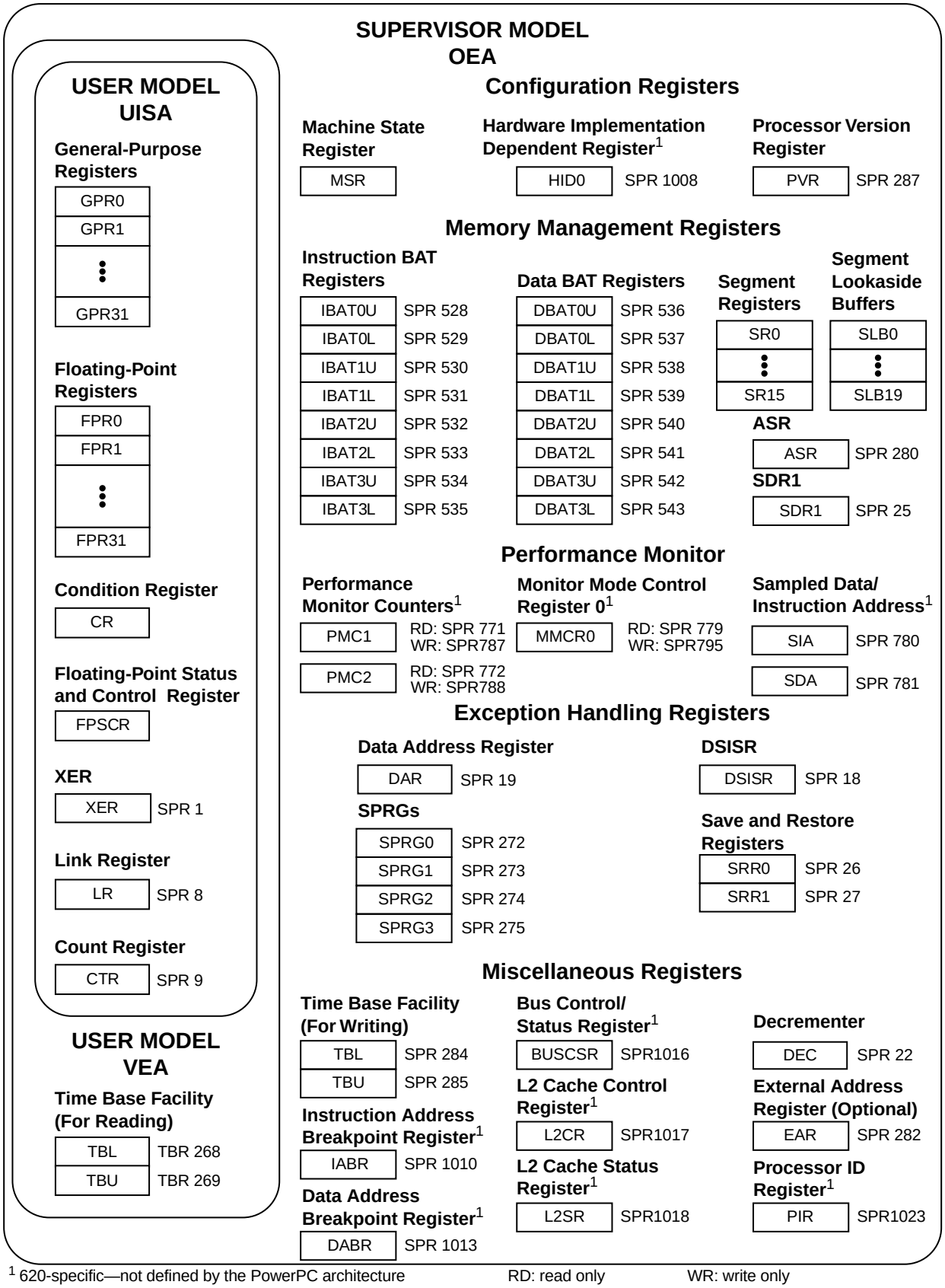
3.2 Registers and Programming Model

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.

During normal execution, a program can access the registers, shown in Figure 6, 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 implicitly as the part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

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

Figure 6 shows the registers implemented in the 620, indicating those that are defined by the PowerPC architecture and those that are 620-specific.



1 620-specific—not defined by the PowerPC architecture RD: read only WR: write only

Figure 6. Programming Model—PowerPC 620 Microprocessor Registers

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 application software). As shown in Figure 6, the programming model incorporates 32 GPRs, 32 FPRs, special-purpose registers (SPRs), and several miscellaneous registers. Note that each PowerPC implementation has its own unique set of implementation-dependent registers that are typically used for debugging, configuration, and other implementation-specific operations.

Some registers are accessible only by supervisor-level software. 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 in supervisor mode.

The following sections summarize the PowerPC registers that are implemented in the 620.

3.2.1 General-Purpose Registers (GPRs)

The PowerPC architecture defines 32 user-level, general-purpose registers (GPRs). These registers are 32 bits wide in 32-bit PowerPC implementations and 64 bits wide in 64-bit PowerPC implementations. The 620 also has eight GPR rename buffers, which provide a way to buffer data intended for the GPRs, reducing stalls when the results of one instruction are required by a subsequent instruction. The use of rename buffers is not defined by the PowerPC architecture, and they are transparent to the user with respect to the architecture. The GPRs and their associated rename buffers serve as the data source or destination for instructions executed in the IUs.

3.2.2 Floating-Point Registers (FPRs)

The PowerPC architecture also defines 32 floating-point registers (FPRs). These 64-bit registers typically are used to provide source and target operands for user-level, floating-point instructions. The 620 has eight FPR rename buffers that provide a way to buffer data intended for the FPRs, reducing stalls when the results of one instruction are required by a subsequent instruction. The rename buffers are not defined by the PowerPC architecture. The FPRs and their associated rename buffers can contain data objects of either single- or double-precision floating-point formats.

3.2.3 Condition Register (CR)

The CR is a 32-bit user-level register that consists of eight 4-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. The 620 also has 16 CR rename buffers, which provide a way to buffer data intended for the CR. The rename buffers are not defined by the PowerPC architecture.

3.2.4 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.

3.2.5 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 are saved when an exception is taken and restored when the exception handling completes. The 620 implements the MSR as a 64-bit register that provides a superset of the 32-bit functionality.

3.2.6 Segment Registers (SRs)

For memory management, 32-bit PowerPC implementations use sixteen 32-bit segment registers (SRs). The 620 provides 16 segment registers for use when executing programs compiled for 32-bit PowerPC microprocessors.

3.2.7 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 SPR instructions, **mtspr** and **mfspr**.

3.2.7.1 User-Level SPRs

The following 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 64 bits wide.
- Count register (CTR)—The CTR is decremented and tested automatically as a result of branch and count instructions. The CTR is 64 bits wide.
- XER—The 32-bit XER contains the integer carry and overflow bits.
- The time base registers (TBL and TBU) can be read by user-level software, but can be written to only by supervisor-level software.

3.2.7.2 Supervisor-Level SPRs

The 620 also contains SPRs that can be accessed only by supervisor-level software. These registers consist of the following:

- The 32-bit data DSISR defines the cause of DSI and alignment exceptions.
- The data address register (DAR) is a 64-bit register that holds the address of an access after an alignment or DSI exception.
- The decrementer register (DEC) is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay. In the 620, the decrementer frequency is equal to the bus clock frequency (as is the time base frequency).
- The 32-bit SDR1 register specifies the page table format used in logical-to-physical address translation for pages.
- The machine status save/restore register 0 (SRR0) is a 64-bit register that is used by the 620 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 64-bit register used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed.
- SPRG0–SPRG3 registers are 64-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 In Word Indexed (**eciwx**) and External Control Out 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.

- The time base registers (TBL and TBU) together provide a 64-bit time base register. The registers are implemented as a 64-bit counter, with the least-significant bit being the most frequently incremented. The PowerPC architecture defines that the time base frequency be provided as a subdivision of the processor clock frequency. In the 620. the time base frequency is equal to the bus clock frequency (as is the decrementer frequency). Counting is enabled by the Time Base Enable ($\overline{\text{TBE}}$) signal.
- The address space register (ASR) is a 64-bit register that holds the physical address of the segment table. The segment table defines the set of memory segments that can be addressed.
- 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 620 includes the following registers not defined by the PowerPC architecture:

- Instruction address breakpoint register (IABR)—This register can be used to cause a breakpoint exception to occur if a specified instruction address is encountered.
- Data address breakpoint register (DABR)—This register can be used to cause a breakpoint exception to occur if a specified data address is encountered.
- Hardware implementation-dependent register 0 (HID0)—This register is used to control various functions within the 620, such as enabling checkstop conditions, and locking, enabling, and invalidating the instruction and data caches.
- Bus control and status register (BUSCSR)—This register controls the setting of various bus operational parameters, and provides read-only access to bus control values set at system boot time.
- L2 cache control register (L2CR)—The L2 cache control register provides controls for the operation of the L2 cache interface, including the ECC mode desired, size of the L2 cache, and the selection of GTL or CMOS interface logic.
- L2 cache status register (L2SR)—The L2 cache status register contains all ECC error information for the L2 cache interface.
- Processor identification register (PIR)—The PIR is a supervisor-level register that has a right-justified, 4-bit field that holds a processor identification tag used to identify a particular 620. This tag is used to identify the processor in multiple-master implementations.
- Performance monitor counter registers (PMC1 and PMC2). The counters are used to record the number of times a certain event has occurred.
- Monitor mode control register 0 (MMCR0)—This is used for enabling various performance monitoring interrupt conditions and establishes the function of the counters.
- Sampled instruction address and sampled data address registers (SIA and SDA)—These registers hold the addresses for instruction and data used by the performance monitoring interrupt.

Note that while it is not guaranteed that the implementation of HID registers is consistent among PowerPC processors, other processors may be implemented with similar or identical HID registers.

3.3 Instruction Set and Addressing Modes

The following subsections describe the PowerPC instruction set and addressing modes.

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 Instruction Set

The 620 implements the entire PowerPC instruction set (for 64-bit implementations) and most optional PowerPC instructions. The PowerPC instructions can be loosely grouped into the following general categories:

- Integer instructions—These include computational and logical instructions.
  - — Integer arithmetic instructions
  - — Integer compare instructions
  - — Logical instructions
  - — Integer rotate and shift instructions
- Floating-point instructions—These include floating-point computational instructions, as well as instructions that affect the FPSCR. Floating-point instructions include the following:
  - — Floating-point arithmetic instructions
  - — Floating-point multiply/add instructions
  - — Floating-point rounding and conversion instructions
  - — Floating-point compare instructions
  - — Floating-point move instructions
  - — Floating-point status and control instructions
  - — Optional floating-point instructions (listed with the optional instructions below)

  The 620 supports all IEEE 754-1985 floating-point data types (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software exception routines.

  The PowerPC architecture also supports a non-IEEE mode, controlled by a bit in the FPSCR. In this mode, denormalized numbers, NaNs, and some IEEE invalid operations are not required to conform to IEEE standards and can execute faster. Note that all single-precision arithmetic instructions are performed using a double-precision format. The floating-point pipeline is a single-pass implementation for double-precision products. A single-precision instruction using only single-precision operands in double-precision format performs the same as its double-precision equivalent.
- Load/store instructions—These include integer and floating-point load and store instructions.
  - — Integer load and store instructions
  - — Integer load and store multiple instructions
  - — Integer load and store string instructions
  - — Floating-point load and store
- 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
  - — System call and **rfi** instructions
  - — Condition register logical instructions

- Synchronization instructions—The PowerPC architecture defines instructions for memory synchronizing, especially useful for multiprocessing:
  - — Load and store with reservation instructions—These UISA-defined instructions provide primitives for synchronization operations such as test and set, compare and swap, and compare memory.
  - — The Synchronize instruction (**sync**)—This UISA-defined instruction is useful for synchronizing load and store operations on a memory bus that is shared by multiple devices.
  - — The Enforce In-Order Execution of I/O instruction (**eieio**)—The **eieio** instruction, defined by the VEA, can be used instead of the **sync** instruction when only memory references seen by I/O devices need to be ordered. The 620 implements **eieio** as a barrier for all storage accesses to the BIU, but not as a barrier for all instructions like the implementation of the **sync** instruction.
- Processor control instructions—These instructions are used for synchronizing memory accesses and managing caches, TLBs, segment registers and SLBs. These instructions include move to/from special-purpose register instructions (**mtspr** and **mfspr**).
- Memory/cache control instructions—These instructions provide control of caches, TLBs, segment registers, and SLBs.
  - — User- and supervisor-level cache instructions
  - — Segment lookaside buffer management instructions
  - — Segment register manipulation instructions
  - — Translation lookaside buffer management instructions
- Optional instructions—the 620 implements the following optional instructions:
  - — The **eciwx**/**ecowx** instruction pair
  - — TLB invalidate entry instruction (**tlbie**)
  - — TLB Synchronize instruction (**tlbsync**)
  - — SLB invalidate entry instruction (**slbie**)
  - — SLB invalidate all instruction (**slbia**)
  - — Optional graphics instructions:
    - – Store Floating-Point as Integer Word Indexed (**stfiwx**)
    - – Floating Reciprocal Estimate Single (**fres**)
    - – Floating Reciprocal Square Root Estimate (**frsqrte**)
    - – Floating Square Root Single (**fsqrts**)
    - – Floating Square Root Double (**fsqrt**)
    - – Floating Select (**fsel**)

Note that this grouping of the instructions does not indicate which execution unit executes a particular instruction or group of instructions.

Integer instructions operate on byte, half-word, word, and double-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, word, and double-word operand loads and stores between memory and a set of 32 GPRs. It also provides for word and double-word operand loads and stores between memory and a set of 32 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 specific store 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 64-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 64-bit unsigned binary arithmetic. A carry from bit 0 is ignored.

3.4 Exception Model

The following subsections describe the PowerPC exception model and the 620 implementation, respectively.

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 various registers and the processor begins execution at an address (exception vector) predetermined for each exception and the processor changes to 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 DSISR and the FPSCR. Additionally, specific 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 PowerPC processor may recognize exception conditions out of order, exceptions are handled 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 must be handled first. Likewise, exceptions that are asynchronous and precise are recognized when they occur (unless they are masked) and the reorder buffer is drained. The address of next instruction to be executed is saved in SRR0 so execution can resume at the proper place when the exception handler returns control to the interrupted process.

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. 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 or machine check exception or to an instruction-caused exception in the exception handler.

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.
- Synchronous, imprecise—The PowerPC architecture defines two imprecise floating-point exception modes, recoverable and nonrecoverable. The 620 treats the imprecise, recoverable and imprecise, nonrecoverable modes as the precise mode.
- Asynchronous—The OEA portion of the PowerPC architecture defines two types of asynchronous exceptions:
  - Asynchronous, maskable—The PowerPC architecture defines the external interrupt and decrementer interrupt which are maskable and asynchronous exceptions. In the 620, and in many PowerPC processors, the hardware interrupt is generated by the assertion of the Interrupt ($\overline{\text{INT}}$) signal, which is not defined by the architecture. In addition, the 620 implements one additional interrupt, the system management interrupt, which performs similarly to the external interrupt, and is generated by the assertion of the System Management Interrupt ($\overline{\text{SMI}}$) signal.

    When these exceptions occur, their handling is postponed until all instructions, and any exceptions associated with those instructions, complete execution.
  - Asynchronous, nonmaskable—There are two nonmaskable asynchronous exceptions that are imprecise: system reset and machine check exceptions. Note that the OEA portion of the PowerPC architecture, which defines how these exceptions work, does not define the causes or the signals used to cause these exceptions. These exceptions may not be recoverable, or may provide a limited degree of recoverability for diagnostic purposes.

The PowerPC architecture defines two bits in the machine state register (MSR)—FE0 and FE1—that determine how floating-point exceptions are handled. There are four combinations of bit settings, of which the 620 implements two. These are as follows:

- Ignore exceptions mode—In this mode, the instruction dispatch logic feeds the FPU as fast as possible and the FPU uses an internal pipeline to allow overlapped execution of instructions. In this mode, floating-point exception conditions return a predefined value instead of causing an exception.
- Precise interrupt mode—This mode includes both the precise mode and imprecise recoverable and nonrecoverable modes defined in the PowerPC architecture. In this mode, a floating-point instruction that causes a floating-point exception brings the machine to a precise state. In doing so, the 620 takes floating-point exceptions as defined by the PowerPC architecture.

The 620 exception classes are shown in Table 2.

Table 2. Exception Classifications

Type	Exception
Asynchronous/nonmaskable	Machine check System reset
Asynchronous/maskable	External interrupt Decrementer System management interrupt (not defined by the PowerPC architecture)
Synchronous/precise	Instruction-caused exceptions
Synchronous/imprecise	Floating-point exceptions

The 620's exceptions, and a general description of conditions that cause them, are listed in Table 3.

Table 3. Overview of 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 the soft reset or hard reset signal.
Machine check	00200	A machine check exception is signaled by the assertion of a qualified $\overline{\text{DERR}}$ indication on the 620 bus, or the machine check interrupt ($\overline{\text{MCP}}$) signal. If MSR[ME] is cleared, the processor enters the checkstop state when one of these signals is asserted. Note that MSR[ME] is cleared when an exception is taken. The machine check exception is also caused by parity errors on the address or data bus, in the instruction or data caches, or L2 ECC errors. The assertion of the $\overline{\text{DERR}}$ signal is determined by load and store operations initiated by the processor; however, it is expected that the $\overline{\text{DERR}}$ signal would be used by a memory controller to indicate that a memory parity error or an uncorrectable memory ECC error has occurred. Note that the machine check exception is imprecise with respect to the instruction that originated the bus operation.

Table 3. Overview of Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
DSI	00300	The cause of a DSI exception can be determined by the bit settings in the DSISR, listed as follows: 0 Set if a load or store instruction results in a direct-store program exception; otherwise cleared. 1 Set if the translation of an attempted access is not found in the primary table entry group (PTEG), or in the secondary PTEG, 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; otherwise cleared. 5 If SR[T] = 1, set by an **eciwx**, **ecowx**, **lwarx**, or **stwcx**. 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. 10 Set if the segment table search fails to find a translation for the effective address; otherwise cleared. 11 Set if **eciwx** or **ecowx** is used and EAR[E] is cleared.
ISI	00400	An ISI exception is caused when an instruction fetch cannot be performed for any of the following reasons: • The effective address cannot be translated. That is, there is a page fault for this portion of the translation, so an ISI exception must be taken to retrieve the translation from a storage device such as a hard disk drive. • The fetch access is to a direct-store segment. • The fetch access violates memory protection. If the key bits (Ks and Kp) 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 external exception signal, $\overline{\text{INT}}$, is asserted. This signal is expected to remain asserted until the exception handler begins execution. Once the signal is detected, the 620 stops dispatching instructions and waits for all dispatched instructions to complete. Any exceptions associated with dispatched instructions are taken before the interrupt is taken.
Alignment	00600	An alignment exception is caused when the processor cannot perform a memory access for the following reasons: An integer load or store double word is not word aligned. A floating-point load, store, **lmw**, **stmw**, **lwarx**, **stwcx.**, **eciwx**, or **ecowx** instruction is not word-aligned. A **dcbz** instruction refers to a page that is marked either cache-inhibited or write-through. A **dcbz** instruction has executed when the 620 data cache is locked or disabled. An **lmw**, **stmw**, **lswi**, **lswx**, **stswi**, or **stswx** instruction is issued in little-endian mode. A floating-point instruction access to a direct-store segment.

Table 3. Overview of 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 exceptions—A floating-point enabled exception condition causes an exception when FPSCR[FEX] is set and depends on the values in MSR[FE0] and MSR[FE1]. 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 or when execution of an optional instruction not provided in the specific implementation 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 user privilege bit, MSR[PR], is set. This exception is also generated for **mtspr** or **mfspr** with an invalid SPR field if SPR[0] = 1 and MSR[PR] = 1. • 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.
Reserved	00A00–00BFF	—
System call	00C00	A system call exception occurs when a System Call (**sc**) instruction is executed.
Trace	00D00	Either the MSR[SE] = 1 and any instruction (except **rfi**) successfully completed or MSR[BE] = 1 and a branch instruction is completed.
Floating-point assist	00E00	Defined by the PowerPC architecture, but not required in the 620.
Reserved	00E10–00EFF	—
Performance monitoring interrupt	00F00	The performance monitoring interrupt is a 620-specific exception and is used with the 620 performance monitor, described in Section 4.2, “Performance Monitor.” The performance monitoring facility can be enabled to signal an exception when the value in one of the performance monitor counter registers (PMC1 or PMC2) goes negative. The conditions that can cause this exception can be enabled or disabled in the monitor mode control register 0 (MMCR0). Although the exception condition may occur when the MSR EE bit is cleared, the actual interrupt is masked by the EE bit and cannot be taken until the EE bit is set.
Reserved	01000–012FF	—

Table 3. Overview of Exceptions and Conditions (Continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Instruction address breakpoint	01300	An instruction address breakpoint exception occurs when the address (bits 0 to 29) in the IABR matches the next instruction to complete in the completion unit, and the IABR enable bit IABR[30] is set.
System management interrupt	01400	A system management interrupt is caused when MSR[EE] = 1 and the $\overline{\text{SMI}}$ input signal is asserted. This exception is provided for use with the nap mode, which is described in Section 4.1, “Power Management.”
Reserved	01500–02FFF	Reserved, implementation-specific exceptions. These are not implemented in the 620.

3.5 Instruction Timing

As shown in Figure 7, the common pipeline of the 620 has five stages through which all instructions must pass. Some instructions occupy multiple stages simultaneously and some individual execution units have additional stages. For example, the floating-point pipeline consists of three stages through which all floating-point instructions must pass.

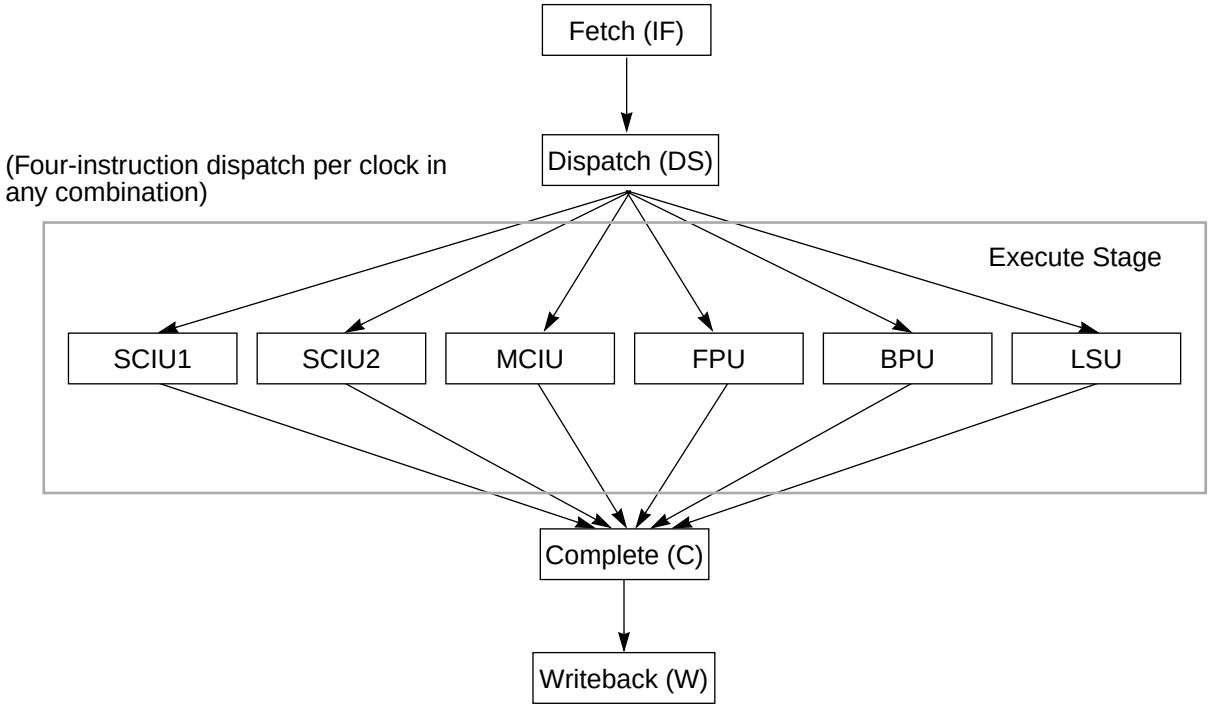


Figure 7. Pipeline Diagram

The common pipeline stages are as follows:

- Instruction fetch (IF)—During the IF stage, the fetch unit loads the decode queue (DEQ) with instructions from the instruction cache and determines from what address the next instruction should be fetched.
- Instruction dispatch (DS)—During the dispatch stage, the decoding that is not time-critical is performed on the instructions provided by the previous ID stage. Logic associated with this stage determines when an instruction can be dispatched to the appropriate execution unit. At the end of the DS stage, instructions and their operands are latched into the execution input latches or into the unit's reservation station. Logic in this stage allocates resources such as the rename registers and reorder buffer entries.
- Execute (E)—While the execution stage is viewed as a common stage in the 620 instruction pipeline, the instruction flow is split among the six execution units, some of which consist of multiple pipelines. An instruction may enter the execute stage from either the dispatch stage or the execution unit's dedicated reservation station.

  At the end of the execute stage, the execution unit writes the results into the appropriate rename buffer entry and notifies the completion stage that the instruction has finished execution.

  The execution unit reports any internal exceptions to the completion stage and continues execution, regardless of the exception. Under some circumstances, results can be written directly to the target registers, bypassing the rename buffers.
- Complete (C)—The completion stage ensures that the correct machine state is maintained by monitoring instructions in the completion buffer and the status of instruction in the execute stage.

  When instructions complete, they are removed from the reorder buffer (ROB). Results may be written back from the rename buffers to the register as early as the complete stage. If the completion logic detects an instruction containing exception status or if a branch has been mispredicted, all subsequent instructions are cancelled, any results in rename buffers are discarded, and instructions are fetched from the correct instruction stream.
- Writeback (W)—The writeback stage is used to write back any information from the rename buffers that was not written back during the complete stage. The CR, CTR, and LR are updated during the writeback stage.

All instructions are fully pipelined except for divide operations and some integer multiply operations. The integer multiplier is a three-stage pipeline. SPR and divide operations can execute in the MCIU in parallel with multiply operations.

The floating-point pipeline has three stages. All floating-point instructions are fully pipelined except for divide and square root operations.

Part 4 Power Management and Performance Monitor

The following sections discuss the power management feature and performance monitor facility of the 620.

4.1 Power Management

The 620 provides two power-saving modes (referred to as the doze mode and nap mode), in which all internal processing and bus operations are suspended. Doze mode allows the 620 to respond to snoop operations, and then automatically return to the power saving doze mode. The nap mode requires the caches to be flushed or invalidated so snoop operations do not require the 620 to respond.

Software initiates both the doze and nap modes by setting the MSR[POW] bit. After this bit is set, the 620 suspends instruction dispatch and waits for all activity in progress, including active and pending bus transactions, to complete. The 620 then powers down the internal clocks. In the case of the nap mode, software also flushes or invalidates the contents of the data cache.

When the 620 is in doze mode, all internal activity stops except for decrementer, time base, and interrupt logic, and the 620 does not snoop bus activity unless the system asserts the WAKE-UP input signal. The 620 will then awaken to complete all pending snoop operations, and then will return to the doze mode.

When the 620 is in nap mode, there is nothing in the caches subject to snoop activity. Assertion of the wake-up signal will cause the 620 to awaken, process a snoop miss in the data cache, and return to the nap mode.

Doze and nap mode is exited (clocks resume and MSR[POW] cleared) when any asynchronous interrupt is detected.

4.2 Performance Monitor

The 620 incorporates a performance monitor facility that system designers can use to help bring up, debug, and optimize software performance, especially in multiprocessing systems. The performance monitor is a software-accessible mechanism that provides detailed information concerning the dispatch, execution, completion, and memory access of PowerPC instructions.

The monitor mode control register 0 (MMCR0) can be used to specify the conditions for which a performance monitoring interrupt is taken. For example, one such condition is associated with one of the counter registers (PMC1 or PMC2) incrementing until the most significant bit indicates a negative value. Additionally, the sampled instruction address and sampled data address registers (SIA and SDA) are used to hold addresses for instruction and data related to the performance monitoring interrupt.

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