SECTION 1
OVERVIEW

This section provides an overview of RCPU features and the PowerPC Architecture™ and summarizes the operation of the RCPU as a PowerPC™ implementation.

1.1 RCPU Overview

The RCPU is a single-issue, 32-bit implementation of the PowerPC architecture. The processor provides 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

The RCPU integrates four execution units: an integer unit (IU), a load/store unit (LSU), a branch processing unit (BPU), and a floating-point unit (FPU). The RCPU can issue one sequential (non-branch) instruction per clock cycle. In addition, the processor attempts to evaluate branch conditions ahead of time and execute branch instructions simultaneously with sequential instructions, often resulting in zero-cycle execution time for branch instructions. Instructions can complete out of order for increased performance; however, the processor makes execution appear sequential.

RCPU-based microcontrollers (MCUs) include an on-chip, four-Kbyte, two-way set-associative instruction cache (I-cache). The I-cache uses a least recently used (LRU) replacement algorithm.

RCPU-based MCUs include a number of features to aid in system debugging. Features implemented in the silicon include internal breakpoint comparators, internal bus visibility, program flow tracking, and a development port.

Figure 1-1 is a simplified block diagram of an RCPU-based MCU.

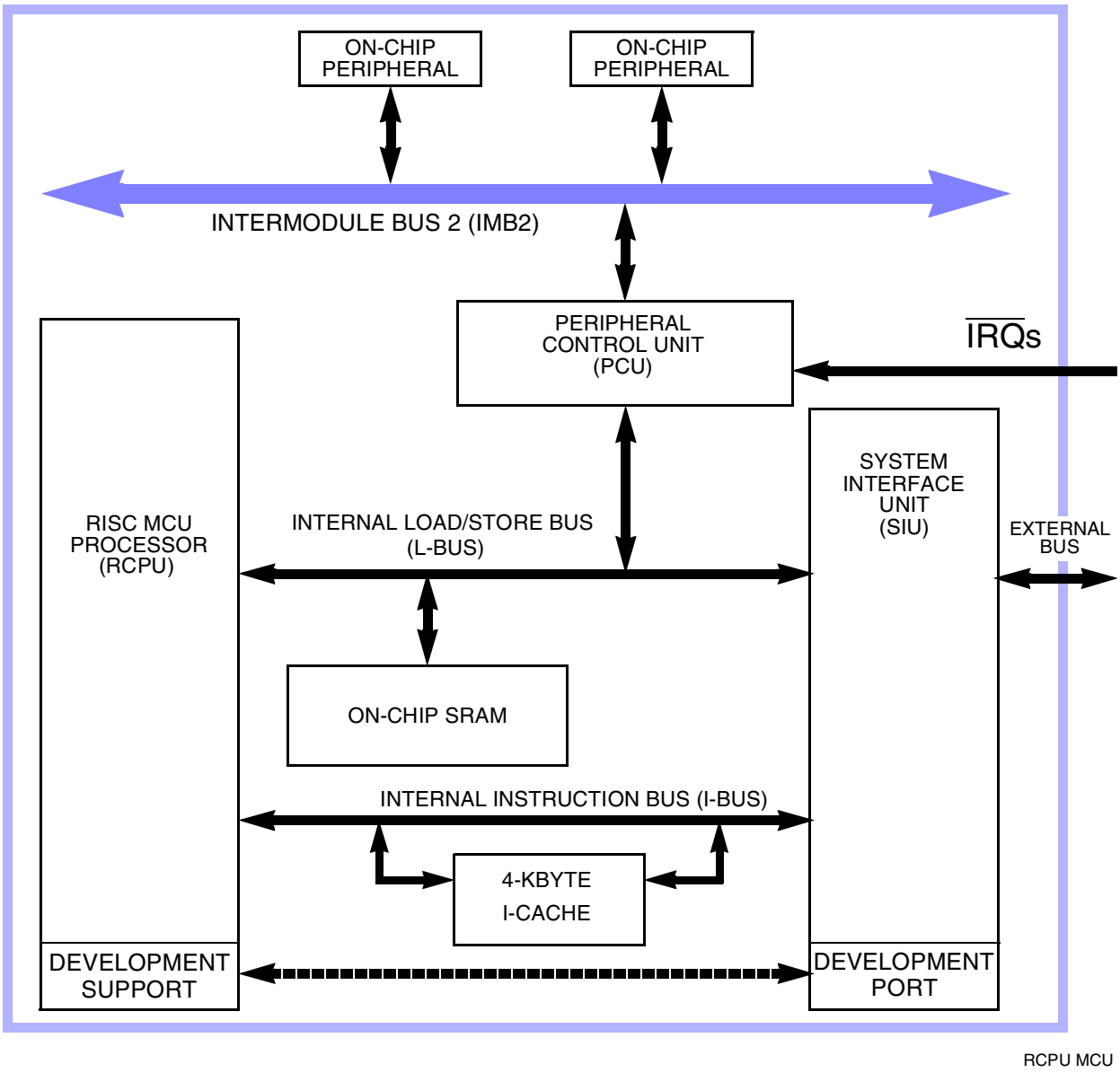

Figure 1-1 RCPU-Based Microcontroller

The following subsections describe the features of the RCPU, provide a block diagram showing the major functional units, and give an overview of RCPU operation.

1.1.1 RCPU Features

Major features of the RCPU are as follows:

- High-performance microprocessor
  - — Single clock-cycle execution for most instructions
- Four independent execution units
  - — Independent load/store unit (LSU) for load and store operations
  - — Branch processor unit (BPU) featuring static branch prediction

  - — A 32-bit integer unit (IU)
  - — Floating-point unit (FPU) for both single- and double-precision operations (fully IEEE 754-compliant when used with software envelope)
- 32 general-purpose registers (GPRs) for integer operands
- 32 floating-point registers (FPRs) for single- or double-precision operands
- Facilities for enhanced system performance
  - — Programmable big- and little-endian byte ordering
  - — Atomic memory references
- In-system testability and debugging features through boundary-scan capability
- High instruction and data throughput
  - — Condition register (CR) look-ahead operations performed by BPU
  - — Branch-folding capability during execution (zero-cycle branch execution time)
  - — Programmable static branch prediction on unresolved conditional branches
  - — A pre-fetch queue that can hold up to four instructions, providing look-ahead capability
  - — Interlocked pipelines with feed-forwarding that control data dependencies in hardware
  - — Four-Kbyte instruction cache: two-way set-associative, LRU replacement algorithm
  - — Programmable static branch prediction on conditional branches

1.1.2 RCPU Block Diagram

Figure 1-2 provides a block diagram of the RCPU.

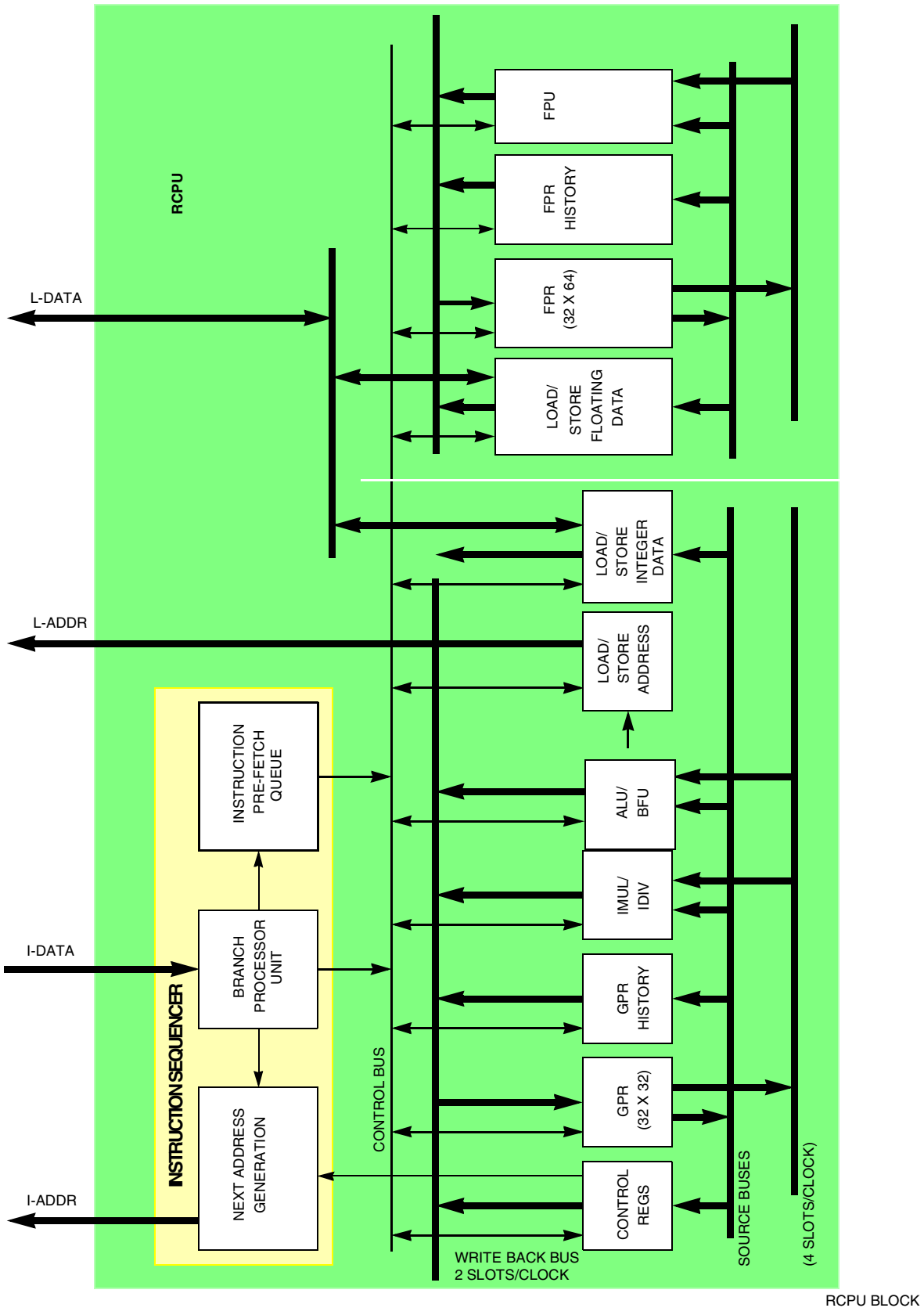

Figure 1-2 RCPU Block Diagram

1.1.3 Instruction Sequencer

The instruction sequencer provides centralized control over data flow between execution units and register files. The sequencer implements the basic instruction pipeline, fetches instructions from the memory system, issues them to available execution units, and maintains a state history so it can back the machine up in the event of an exception.

In addition, the sequencer implements all branch processor instructions, which include flow control and condition register instructions. Refer to **1.1.4.1 Branch Processing Unit (BPU)** for more details on the branch processing unit within the instruction sequencer.

The instruction sequencer fetches the instructions from the instruction cache into the instruction pre-fetch queue, which can hold up to four instructions. The processor uses branch folding (a technique of removing the branch instructions from the pre-fetch queue) in order to execute branches in parallel with execution of sequential instructions. Sequential (non-branch) instructions reaching the top of the instruction pre-fetch queue are issued to the execution units. Instructions may be flushed from the queue when an external interrupt is detected, a previous instruction causes an exception, or a branch prediction turns out to be incorrect.

All instructions, including branches, enter the history buffer along with processor state information that may be affected by the instruction's execution. This information is used to enable out-of-order completion of instructions together with the handling of precise exceptions. Instructions may be flushed from the machine when an exception is taken. Refer to **6.3 Precise Exception Model Implementation** and to **7.1 Instruction Flow** for additional information.

An instruction retires from the machine after it finishes execution without exception and all preceding instructions have already retired from the machine.

Figure 1-3 illustrates the instruction flow in the RCPU.

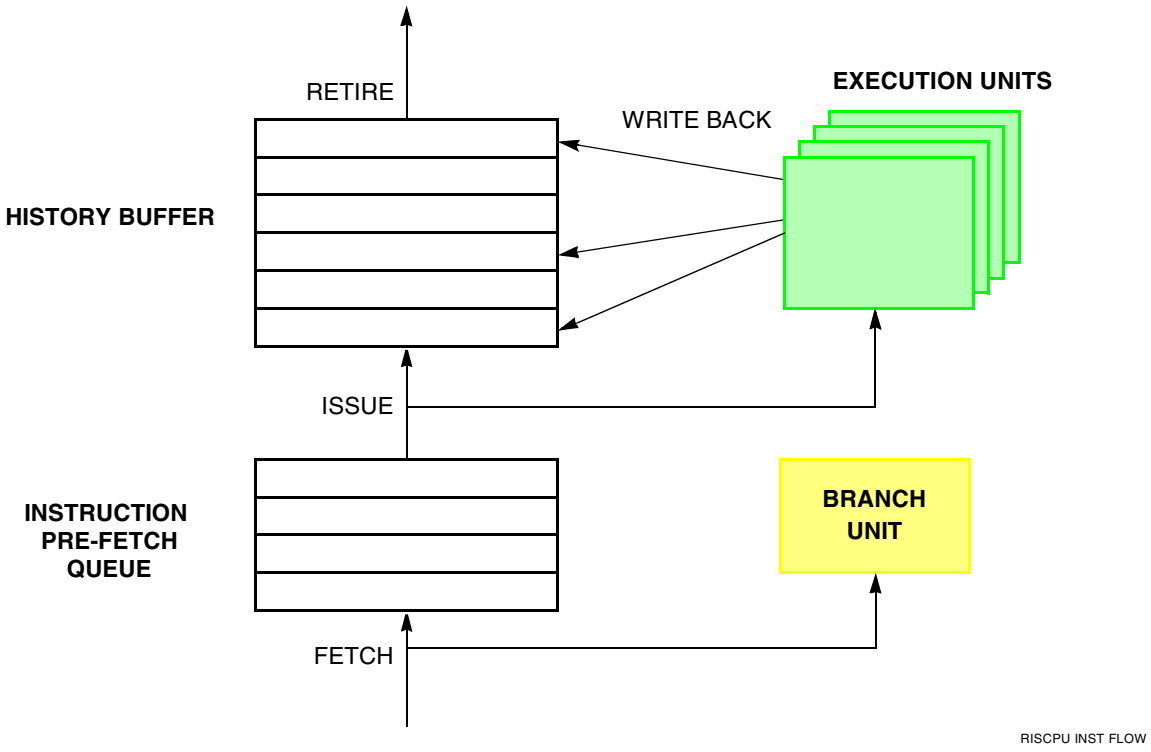

Figure 1-3 RCPU Instruction Flow

1.1.4 Independent Execution Units

The RCPU supports independent floating-point, integer, load/store, and branch processing execution units, making it possible to implement advanced features such as look-ahead operations. For example, since branch instructions do not depend on GPRs or FPRs, branches can often be resolved early, eliminating stalls caused by taken branches.

Table 1-1 summarizes the RCPU execution units.

Table 1-1 RCPU Execution Units

Unit	Description
Branch processor unit (BPU)	Includes the implementation of all branch instructions.
Load/store unit (LSU)	Includes implementation of all load and store instructions, whether defined as part of the integer processor or the floating-point processor.
Integer unit (IU)	Includes implementation of all integer instructions except load/store instructions. This module includes the GPRs (including GPR history and scoreboard) and the following subunits: • The IMUL-IDIV unit includes the implementation of the integer multiply and divide instructions. • The ALU-BFU unit includes implementation of all integer logic, add and subtract instructions, and bit field instructions.
Floating-point unit (FPU)	Includes the FPRs (including FPR history and scoreboard) and the implementation of all floating-point instructions except load and store floating-point instructions.

1.1.4.1 Branch Processing Unit (BPU)

The branch processor unit executes all branch instructions defined in the PowerPC architecture, including flow control and condition register instructions.

The BPU is implemented as part of the instruction sequencer. The BPU performs condition register look-ahead operations on conditional branches. The BPU looks through 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. (Refer to the discussion of the BO field in **4.6 Flow Control Instructions**.) Therefore, when an unresolved conditional branch instruction is encountered, the processor pre-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-accessible registers: the link register (LR), the count register (CTR), and the condition register (CR). The BPU calculates the return pointer for subroutine calls and saves it into the LR. 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 independent from execution of integer and floating-point instructions.

1.1.4.2 Integer Unit (IU)

The integer unit executes all fixed-point (integer) processor instructions defined by

the PowerPC architecture, except for fixed-point load and store instructions, which are implemented by the load/store unit. The IU consists of two execution units:

- The IMUL-IDIV executes the integer multiply and divide instructions.
- The ALU-BFU unit executes all integer logic, add, and subtract instructions, and bit-field instructions.

The IU includes the integer exception register (XER) and the general-purpose register file. These registers are described in **SECTION 2 REGISTERS**.

1.1.4.3 Floating-Point Unit (FPU)

The floating-point unit executes all the floating-point processor instructions defined by the PowerPC architecture, except for floating-point load and store instructions, which are executed by the load/store unit.

The floating-point unit contains a double-precision multiply array, the floating-point status and control register (FPSCR), and the FPRs. The multiply-add array allows the processor to efficiently implement floating-point operations such as multiply, multiply-add, and divide.

The RCPU depends on a software envelope to fully implement the IEEE floating-point specification. Overflows, underflows, NaNs, and denormalized numbers cause floating-point assist exceptions that invoke a software routine to deliver (with hardware assistance) the correct IEEE result. Refer to **6.11.10 Floating-Point Assist Exception (0x00E00)** for additional information.

To accelerate time-critical operations and make them more deterministic, the RCPU provides a mode of operation that avoids invoking the software envelope and attempts to deliver results in hardware that are adequate for most applications, if not in strict conformance with IEEE standards. In this mode, denormalized numbers, NaNs, and IEEE-invalid operations are treated as legitimate, returning default results rather than causing floating-point assist exceptions.

1.1.4.4 Load/Store Unit (LSU)

The load/store unit handles all integer and floating-point load and store instructions, including unaligned and string accesses. LSU instructions transfer data between the integer and floating-point register files and the chip-internal load/store bus (L-bus). The load/store unit is implemented as an independent execution unit so that stalls in the memory pipeline do not cause the master instruction pipeline to stall (unless there is a data dependency). The unit is fully pipelined so that memory instructions of any size may be issued on back-to-back cycles.

There is a 32-bit wide data path between the load/store unit and the integer register file and a 64-bit wide data path between the load/store unit and the floating-point register file.

The LSU interfaces with the external bus interface for all instructions that access memory. Addresses are formed by adding the source one register operand speci-

fied by the instruction (or zero) to either a source two register operand or to a 16-bit, immediate value embedded in the instruction.

1.1.5 Instruction Cache

RCPU-based MCUs contain a 4-Kbyte, two-way set associative instruction cache. The cache is organized into 128 sets, two lines per set, and four words per line. Cache lines are aligned on four-word boundaries in memory.

A cache access cycle begins with an instruction request from the instruction unit in the processor. In case of a cache hit, the instruction is delivered to the instruction unit. In the case of a cache miss, the cache initiates a burst read cycle on the I-bus with the address of the requested instruction. The first word received from the bus is the requested instruction. The cache forwards this instruction to the instruction unit of the CPU as soon as it is received from the I-bus. A cache line is then selected to receive the data which will be coming from the bus. An LRU replacement algorithm is used to select a line when no empty lines are available.

Each cache line can be used as an SRAM, thus allowing the application to lock critical code segments that need fast and deterministic execution time.

The instruction cache is described in **SECTION 5 INSTRUCTION CACHE**.

1.1.6 Instruction Pipeline

The RCPU is a pipelined processor. A pipelined processor is one in which the processing of an instruction is broken down into discrete stages; in the RCPU, these stages are dispatch, execute, writeback, and retirement (described below). Because instruction processing is broken into a series of steps, an instruction does not require the entire resources of the processor. 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.

The RCPU instruction pipeline has four stages:

1. The dispatch stage is implemented using a distributed mechanism. The central dispatch unit broadcasts the instruction to all units. In addition, scoreboard information (regarding data dependencies) is broadcast to each execution unit. Each execution unit decodes the instruction. If the instruction is not implemented, a program exception is taken. If the instruction is legal and no data dependency is found, the instruction is accepted by the appropriate execution unit, and the data found in the destination register is copied to the history buffer. If a data dependency exists, the machine is stalled until the dependency is resolved.
2. In the execute stage, each execution unit that has an executable instruction executes the instruction (perhaps over multiple cycles).
3. In the writeback stage, the execution unit writes the result to the destination register and reports to the history buffer that the instruction is completed.
4. In the retirement stage, the history buffer retires instructions in architectural order. An instruction retires from the machine if it completes execution with

no exceptions and if all instructions preceding it in the instruction stream have finished execution with no exceptions. As many as six instructions can be retired in one clock.

The history buffer maintains the correct architectural machine state. An exception is taken only when the instruction is ready to be retired from the machine. When an exception is taken, all instructions following the excepting instruction are canceled, (i.e., the values of the affected destination registers are restored using the values saved in the history buffer during the dispatch stage).

Figure 1-4 illustrates the basic instruction pipeline timing. Refer to **SECTION 7 INSTRUCTION TIMING** for more detailed timing illustrations.

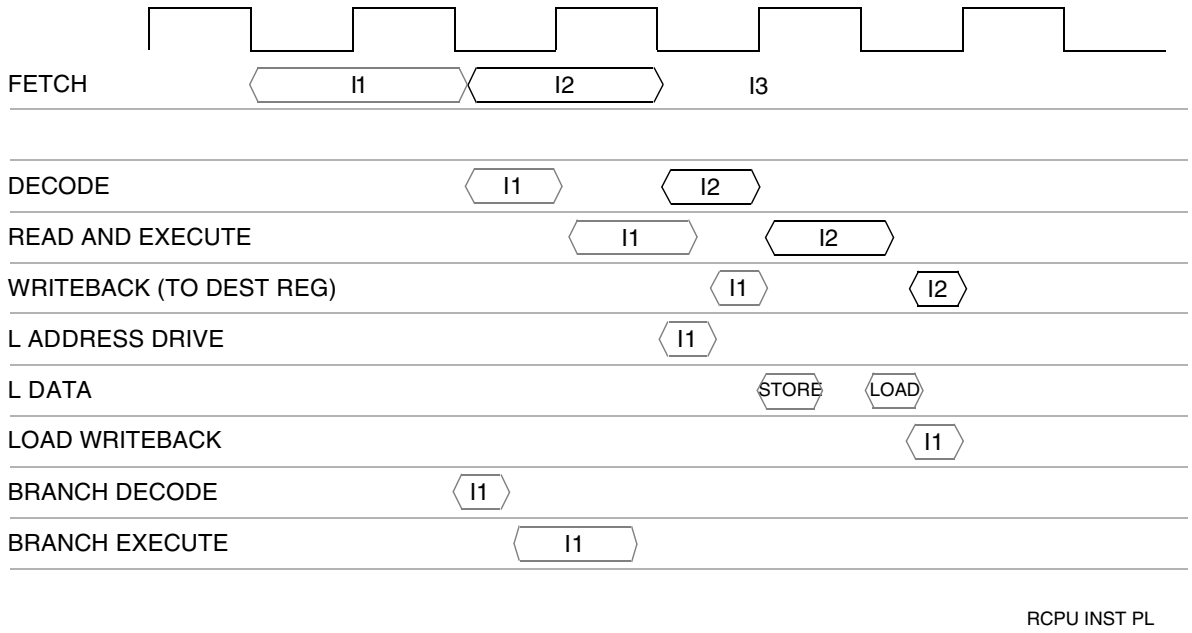

Figure 1-4 Basic Instruction Pipeline

1.1.7 Development Support

Development tools are used by a microcomputer system developer to debug the hardware and software of a target system. These tools are used to give the developer some control over the execution of the target program and to allow the user to debug the program by observing its execution. In-circuit emulators, bus state analyzers, and software monitors are the most frequently used debugging tools. The RCPU supports the use of development tools by providing internal breakpoint comparators, internal bus visibility, program flow tracking, and a development port.

For details on development support, refer to **SECTION 8 DEVELOPMENT SUPPORT**.

1.2 Levels of the PowerPC Architecture

The PowerPC architecture consists of three layers. Adherence to the PowerPC architecture can be measured in terms of which of the following levels of the architecture are implemented:

- PowerPC user instruction set architecture (UISA) — 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 a uniprocessor environment.
- PowerPC virtual environment architecture (VEA) — Describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the VEA also adhere to the UISA, but may not necessarily adhere to the OEA.
- PowerPC operating environment architecture (OEA) — Defines the memory management model, supervisor-level registers, synchronization requirements, and the exception model. Implementations that conform to the OEA also adhere to the UISA and the VEA.

1.3 The RCPU as a PowerPC Implementation

This subsection describes the RCPU as a member of the PowerPC processor family.

1.3.1 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 memory and on-chip registers.

PowerPC processors have two levels of privilege: supervisor mode of operation (typically used by the operating system) and 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. Each PowerPC processor also has its own unique set of implementation-specific registers.

The RCPU is a 32-bit implementation of the PowerPC architecture. In the RCPU, the time base and FPRs are 64 bits; all other registers are 32 bits.

The following sections summarize the PowerPC registers that are implemented in the RCPU. Refer to **SECTION 2 REGISTERS** for detailed descriptions of PowerPC registers. In addition, for descriptions of the I-cache control registers, refer to **SECTION 5 INSTRUCTION CACHE**. For details on development-support registers, refer to **SECTION 8 DEVELOPMENT SUPPORT**.

1.3.1.1 General-Purpose Registers (GPRs)

The processor provides 32 user-level, general-purpose registers (GPRs). The GPRs serve as the data source or destination for all integer instructions and provide addresses for all memory-access instructions.

1.3.1.2 Floating-Point Registers (FPRs)

The processor also provides 32 user-level 64-bit floating-point registers. 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. The floating-point register file can only be accessed by the FPU.

1.3.1.3 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.

1.3.1.4 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.

1.3.1.5 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 are saved when an exception is taken and restored when the exception handling completes.

1.3.1.6 Special-Purpose Registers (SPRs)

The processor provides several 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**.

1.3.1.7 User-Level SPRs

The following SPRs are accessible by user-level software:

- The link register (LR) can be used to provide the branch target address and to hold the return address after branch and link instructions.
- The count register (CTR) is decremented and tested automatically as a result of branch-and-count instructions.
- The integer exception register (XER) contains the integer carry and overflow bits and two fields for the load string and compare byte indexed (**lscbx**) instruction. The XER is 32 bits wide in all implementations.

- The time base (TB) can be read at the user privilege level. A separate SPR number is provided for writing to the time base. Writes to the time base can occur only at the supervisor privilege level.

NOTE

While these registers are defined as SPRs and can be accessed by using the **mtspr** and **mfspr** instructions, they (except for the time base) are typically accessed implicitly.

1.3.1.8 Supervisor-Level SPRs

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

- The data access exception (DAE)/source instruction service register (DSISR) defines the cause of data access and alignment exceptions.
- The data address register (DAR) 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. The DEC frequency is provided as a subdivision of the processor clock frequency.
- The machine status save/restore register 0 (SRR0) is used by the processor to save 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 used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed.
- General SPRs, SPRG[0:3], are provided for operating system use.
- The processor version register (PVR) is a read-only register that identifies the version (model) and revision level of the PowerPC processor.
- The time base (TB) can be written to only at the supervisor privilege level. Separate SPR numbers are provided for reading and writing to the time base.

The following supervisor-level SPRs are implementation-specific to the RCPU:

- The EIE, EID, and NRI are provided to facilitate exception processing.
- Cache control SPRs allow system software to control the operation of the instruction cache.
- Development support SPRs allow development-system software control over the on-chip development support.
- The floating-point exception cause register (FPECR) is a 32-bit internal status and control register used to assist the software emulation of floating-point operations.

1.3.2 Instruction Set and Addressing Modes

The following subsections describe the PowerPC instruction set and addressing modes and summarize the instructions implemented in the RCPU.

1.3.2.1 PowerPC Instruction Set

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 simplify instruction pipelining. In addition, each instruction is defined in a way that simplifies pipelined implementations and allows maximum realization of instruction-level parallelism.

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
- 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 cache management.
  - — Move to/from special-purpose register instructions
  - — Synchronize
  - — Instruction synchronize
- Memory control instructions. These instructions provide control of the instruction cache.
  - — Instruction cache block invalidate

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision and double-precision 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 pro-

vides 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.

1.3.2.2 PowerPC Addressing Modes

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

Note that with register indirect with immediate index addressing, the offset can be equal to zero.

Refer to **4.1.2 Addressing Modes and Effective Address Calculation** for additional information.

1.3.2.3 RCPU Instruction Set

The RCPU instruction set is defined as follows:

- The RCPU supports all 32-bit PowerPC UISA required instructions.
- The RCPU supports the following PowerPC VEA instructions: **eieio**, **icbi**, **isync**, and **mftb**.
- The RCPU supports the following PowerPC OEA instructions: **mfmsr**, **mfspr**, **mtmsr**, **mtspr**, **rfi**, and **sc**. (Note that **mtspr**, **mfspr**, and **sc** are also defined in the UISA architecture.)
- The RCPU does not provide any implementation-specific instructions not defined in the PowerPC architecture.
- The RCPU implements the following instruction which is defined as optional in the PowerPC architecture: **stfiwx**.

An attempt to execute a PowerPC optional instruction that is not implemented in hardware causes the RCPU to take the implementation dependent software emulation exception.

For additional information on the PowerPC architecture, refer to *PowerPC Microprocessor Family: the Programming Environments*, MPCFPE/AD (Motorola order number).

1.3.3 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, the address of the instruction to be executed after control is returned to the original program and the contents of the machine state register are saved to the save/restore registers (SRR0 and SRR1). Program control then passes from user to supervisor level, and software continues execution at an address (exception vector) predetermined for each exception.

Although multiple exception conditions can map to a single exception vector, the specific condition can be determined by examining a register associated with the exception — for example, the DAE/DSISR and the FPSCR. Specific exception conditions can be explicitly enabled or disabled by software.

While exception conditions may be recognized out of order, they are handled strictly in order. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream are allowed to complete. Any exceptions caused by those instructions are handled in order.

Unless a catastrophic condition causes a non-maskable 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 the processor can recover the machine state following an exception.

For additional information on exception handling, refer to **SECTION 6 EXCEPTIONS**.