Chapter 1
Introduction

This chapter provides an introduction to LSI Logic's Enhanced Self-Embedding™ Processors, the LR33300 and the LR33310. These 32-bit processors are high-performance, highly integrated processors that implement the MIPS Reduced Instruction Set Computer (RISC) architecture. This chapter is divided into six sections:

- RISC Architectures
- The LR333x0 Enhanced Self-Embedding Processors
- Pipeline Architecture
- Memory System Hierarchy
- Comparison of the LR333x0 and the LR33000
- Compatibility with the R2000 and R3000 Microprocessors

1.1 RISC Architectures

Until recently, computer architectures had evolved with families of increasingly complex processors. Under market pressures to preserve existing software, Complex Instruction Set Computer (CISC) architectures evolved by the gradual addition of microcode and increasingly elaborate operations. The intent was to provide more support for high-level languages and operating systems, as semiconductor advances made it possible to fabricate more complex integrated circuits.

In recent years, RISC architectures have implemented a more sophisticated division of complexity between hardware, firmware, and software. RISC concepts emerged from statistical analysis of how software actually uses the resources of a processor. Dynamic measurement of system kernels and object modules generated by optimizing compilers shows an overwhelming predominance, even in the code for CISC machines, of the simplest instructions. Complex instructions are seldom used because microcode rarely provides the precise routines needed to support a variety of high-level language and system environments. RISC designs eliminate the

microcoded routines and turn the low-level control of the machine over to software.

The MIPS RISC design process is an iterative process that uses feedback to tune the design. The MIPS architecture started with the knowledge of earlier RISC efforts from Stanford University; the optimizing compilers were begun with the knowledge of those efforts. A base-level instruction set was proposed, and measurements were taken from simulations of code compiled with the existing optimizers. Proposals for additions to the instruction set were carefully weighed to verify that these additions actually improved performance. Any instruction added for performance reasons had to provide a verifiable 1% performance gain over a range of applications or else the instruction was rejected.

The result of this approach is an instruction set that is very well tuned for high-level language use. Every instruction is either structurally necessary (such as Restore From Exception) or can be generated by compilers. This instruction set approach stands in contrast to many other machines. Even machines labelled RISC often have user-level instructions or instruction mode combinations that are very difficult to reach from compiled languages.

Benefits of RISC Design

Some of the important benefits that result from the RISC design techniques are a result of the overall reduction in complexity: the simpler design allows both chip-area resources and human resources to be applied to features that enhance performance. Some benefits of RISC designs are:

- **Shorter Design Cycle** – The simplified architectures of RISC processors can be implemented more quickly. It is much easier to implement and debug a streamlined, simplified architecture with no microcode than a complex, microcoded architecture.
- **Smaller Chip Size** – The simplicity of RISC processors also frees scarce chip-area resources for performance-critical structures like larger register files, coprocessors, and fast multiply-divide units. These additional resources help these processors obtain an even greater performance edge.
- **User (Programmer) Benefits** – Simplicity in architecture also helps the user:
  - The uniform instruction set is easier to use.
  - Instruction count correlates more closely with cycle count, a fact that makes it easier to measure the true impact of code optimization activities.

Optimizing Compilers

The RISC architecture was designed so that compilers, not assembly language programmers, have an optimal working environment. The RISC philosophy assumes that high-level language (HLL) programming will be used. This philosophy is in contrast to the older CISC (Complex Instruction Set Computing) philosophy that was developed when assembly language programming was of primary importance.

The trend toward HLL instructions has led to an emphasis on the use of efficient compilers to convert HLL instructions to machine instructions. Primary measures of a compiler's efficiency are the compactness of the code it generates and the execution time of that code. Modern, optimizing compilers have evolved to provide great efficiency in the HLL-to-machine language translation.

Optimizing compilers and RISC architectures have a synergistic relationship. Compilers can perform their best job of optimization with a RISC architecture. RISC-type computers, in many cases, rely on compilers to obtain their full performance capabilities.

During the development of more efficient compilers, an analysis of instruction streams revealed that most time was spent executing simple instructions and performing load and store operations. The more complex instructions were used less frequently. It was also learned that compilers produced code that was often a narrow subset of the processor's architecture: complex instructions and features were not usable by compilers.

The more complex, powerful instructions are either difficult for the compiler to use or those instructions do not precisely fit the HLL requirements. A compiler prefers instructions that perform simple, well-defined operations with minimum side-effects. Since these characteristics are typical of a RISC instruction set, a natural match exists between RISC architectures and efficient, optimizing compilers. This match makes it easier for compilers to choose the most effective sequences of a machine's instructions to accomplish the tasks described by a high-level language.

Family of Compilers

Many compiler products, especially on microprocessors, are cobbled together from various sources that do not necessarily fit together very well. Rather than treating each language's compiler as a separate entity, the MIPS "language family" approach shares common elements across compilers. Unique among RISC systems, the MIPS family of compilers (RISCompilers) offers both tight integration and broad language coverage. The suite of compilers:

- Provides front-ends for six languages (C, FORTRAN, Pascal, Ada, PL/1, COBOL)
- Uses a common intermediate language, thus offering a reasonable way to add language front-ends over time
- Shares all of the back-end optimization and code generation
- Uses the same object format and calling conventions
- Cleanly supports mixed language programs
- Supports debugging of all of the languages, including mixtures

The MIPS "language family" approach yields consistently high-quality compilers for all languages since common elements make up the majority of each of the language products. In addition, the ability to develop and execute multi-language programs is provided, which promotes flexibility in development, avoids recoding of proven program segments, and protects the user's investment in software. The common back-end also spreads optimizing and code generating improvements immediately across the family of RISCompilers, thereby reducing maintenance.

1.2 The LR333x0 Enhanced Self-Embedding Processors

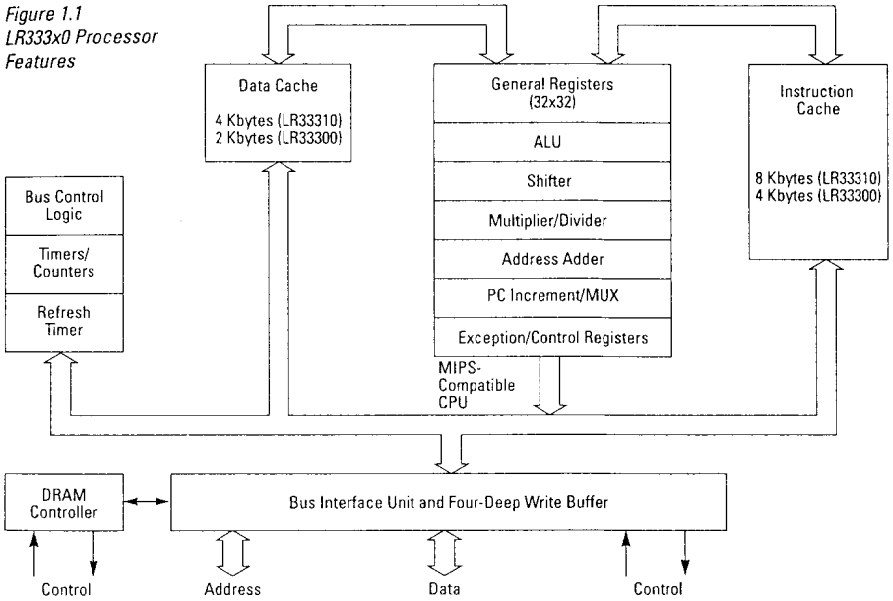
LSI Logic's LR333x0 Self-Embedding Processor is a single-chip microprocessor that has been optimized for embedded control applications. The LR333x0 consists of a MIPS-compatible CPU with on-chip instruction and data caches, a DRAM Controller, three counter/timers, and a sophisticated bus interface unit.

This section contains a description of the processor's general features, its register architecture, its instruction set, its system control processor, and its operating modes.

Processor General Features

The LR333x0 provides several features that make it especially suitable for embedded applications. Figure 1.1 is a block diagram of the processor; descriptions of the blocks follow.

Figure 1.1 LR333x0 Processor Features



MIPS CPU – The LR333x0 implements the high-performance MIPS architecture.

- **Instruction Set Compatibility** – The LR333x0 maintains user-code binary compatibility with the R2000, R3000, and R3000A microprocessors.
- **Efficient Pipelining** – The CPU's five-stage pipeline assists in obtaining an execution rate approaching one instruction per cycle. Pipeline stalls and exceptional events are handled precisely and efficiently.
- **Full 32-bit Operation** – The LR333x0 contains 32 32-bit registers. All instructions and addresses are 32 bits wide to provide a 4-Gbyte address space.

- **Enhanced Debug Features** – The LR333x0 includes hardware break-point on program counter and break-point on data address registers to ease software debugging, and it includes program trace support logic.

On-Chip Cache Memory – The LR33300 contains a 4-Kbyte instruction cache and a 2-Kbyte data cache. The LR33310 contains an 8-Kbyte instruction cache and a 4-Kbyte data cache. The processor can access both caches during a single clock cycle, which allows the processor to execute one instruction per cycle when executing instructions out of cache memory. The LR33310 I-Cache is two-way set associative. The other caches employ direct address mapping. The I-Cache and D-Cache support bus snooping to maintain cache coherency. The data cache uses a write-through technique to maintain coherency with main memory. The D-Cache may be reconfigured as Scratchpad RAM, direct mapped into any cacheable memory space.

On-Chip DRAM Control – The LR333x0 has an integral DRAM Controller that supports many popular DRAMs. Its features include support for page-mode DRAMs, $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh, DMA by separate I/O devices, synchronous DRAM support, and interleaving.

DMA Support – Many systems allow peripherals to load or store data in the main system memory without processor intervention in order to reduce the load on the processor. This process, called direct memory access (DMA), is managed by a DMA Controller. The LR333x0 can be configured to support DMA operations through the DRAM Configuration Register. The LR333x0 supports two DMA access types: BREQ/BGNT and DMAR/DMAC. Refer to Section 9.8, "Direct Memory Access (DMA) Transactions," for more information.

Bus Interface Unit (BIU) – The LR333x0's BIU provides a simple memory interface that is easily connected to I/O devices, supports 8-bit, 16-bit, and 32-bit PROMs, and includes a programmable wait-state generator. The BIU supports optional parity generation and checking. Checking may be suspended on a per-memory-access basis, and the LR333x0 provides the PERR output to signal parity errors to external logic.

On-Chip Counter/Timers – The LR333x0 includes three counter/timers: two 32-bit general-purpose timers (discussed in Chapter 7, "Counter/Timers") and a 16-bit refresh timer (discussed in Section 9.9, "Refresh Generator"). The refresh timer can support an optional off-chip DRAM Controller.

CPU Registers

The LR333x0 CPU provides 32 general-purpose 32-bit registers, a 32-bit Program Counter, and two 32-bit registers (HI and LO). The HI and LO registers hold the results of integer multiply and divide operations. The CPU registers are shown in Figure 1.2 and are described in Chapter 2, "Data and Registers." Notice that the figure does not contain a Program Status Word (PSW) register; the functions traditionally provided by a PSW register are instead provided by the *Status* and *Cause* registers incorporated within the system control coprocessor (CP0).

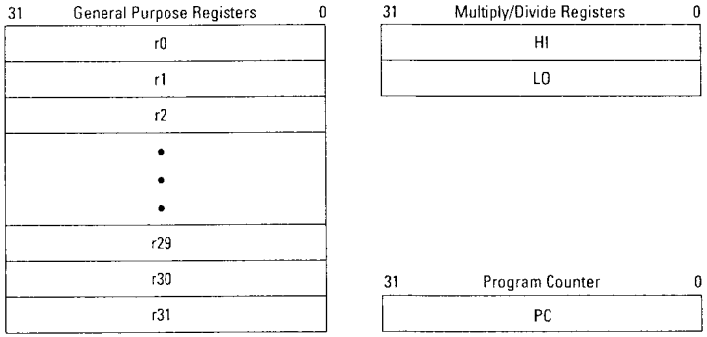
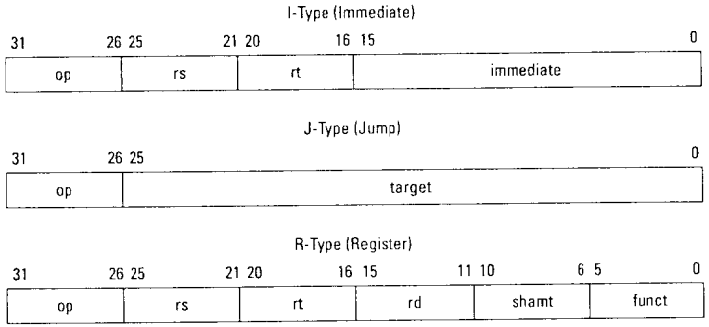


Figure 1.2 LR333x0 CPU Registers

Instruction Set Overview

All LR333x0 instructions are 32 bits long. As shown in Figure 1.3, instructions have three basic formats: *I-type* (immediate), *J-type* (jump), and *R-type* (register). Instruction decoding is simplified by this approach. More complicated (and less frequently used) operations and addressing modes can be synthesized by the compiler using sequences of simple RISC-type instructions. The assembler recognizes some CISC-type instructions for programming ease, but translates them into sequences of simple instructions.

Figure 1.3
LR333x0 Instruction Formats



The LR333x0 instruction set is divided into the following groups:

- *Load/Store* instructions are the only instructions that move data between memory and general registers. These instructions are I-type, since the only addressing mode supported is base register plus 16-bit, signed immediate offset.
- *Computational* instructions perform arithmetic, logical, and shift operations on values in registers. These instructions are either R-type (both operands and the result are registers) or I-type (one operand is a 16-bit immediate).
- *Jump* and *Branch* instructions change the control flow of a program. Jumps are always to a paged, absolute address formed by combining a 26-bit target with four bits of the program counter (J-type format, for subroutine calls), or 32-bit register byte addresses (R-type, for returns and dispatches). Branches have 16-bit offsets relative to the program counter (I-type). Jump and Link instructions save a return address in register *r31*.

Table 1.1
LR333x0 Instruction Summary

Op	*Description*
Load/Store Instructions	
LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned
LW	Load Word
LWL	Load Word Left
LWR	Load Word Right
SB	Store Byte
SH	Store Halfword
SW	Store Word
SWL	Store Word Left
SWR	Store Word Right
Arithmetic Instructions (ALU Immediate)	
ADDI	Add Immediate
ADDIU	Add Immediate Unsigned
SLTI	Set on Less Than Immediate
SLTIU	Set on Less Than Immediate Unsigned
ANDI	AND Immediate
ORI	OR Immediate
XORI	Exclusive OR Immediate
LUI	Load Upper Immediate
Arithmetic Instructions (3-Operand, Register-Type)	
ADD	Add
ADDU	Add Unsigned
SUB	Subtract
SUBU	Subtract Unsigned
SLT	Set on Less Than
SLTU	Set on Less Than Unsigned
AND	AND
OR	OR
XOR	Exclusive OR
NOR	NOR
Shift Instructions	
SLL	Shift Left Logical
SRL	Shift Right Logical
SRA	Shift Right Arithmetic
SLLV	Shift Left Logical Variable
SRLV	Shift Right Logical Variable
SRAV	Shift Right Arithmetic Variable
Multiply/Divide Instructions	
MULT	Multiply
MULTU	Multiply Unsigned
DIV	Divide
DIVU	Divide Unsigned
MFHI	Move From HI
MTHI	Move To HI
MFLO	Move From LO
MTLO	Move To LO
Jump and Branch Instructions	
J	Jump
JAL	Jump And Link
JR	Jump Register
JALR	Jump And Link Register
BEQ	Branch on Equal
BNE	Branch on Not Equal
BLEZ	Branch on Less than or Equal to Zero
BGTZ	Branch on Greater Than Zero
BLTZ	Branch on Less Than Zero
BGEZ	Branch on Greater than or Equal to Zero
BLTZAL	Branch on Less Than Zero And Link
BGEZAL	Branch on Greater than or Equal to Zero And Link
Special Instructions	
SYSCALL	System Call
BREAK	Breakpoint
Coprocessor Instructions	
BCzT	Branch on Coprocessor z True
BCzF	Branch on Coprocessor z False
System Control Coprocessor (CP0) Instructions	
MTC0	Move To CP0
MFC0	Move From CP0
RFE	Restore From Exception

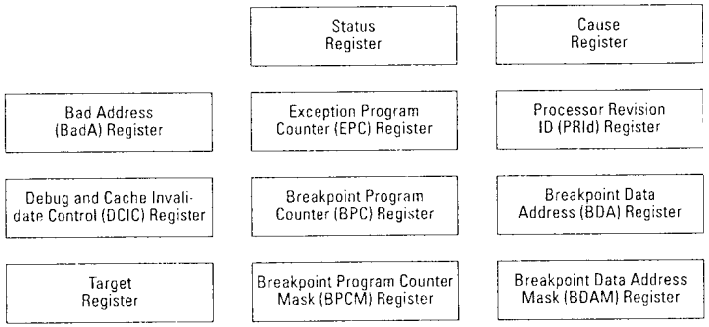
- *Coprocessor* instructions perform operations in the coprocessors, most of which are not implemented by the LR333x0. See Section 1.5, "Comparison of the LR333x0 and the LR33000," for details.
- *Coprocessor 0* instructions perform operations on the system control coprocessor (CP0) registers to manipulate the exception handling facilities of the processor.
- *Special* instructions perform a variety of tasks including movement of data between special and general registers, system calls, and breakpoint. These instructions are always R-type.

Table 1.1 lists the instruction set of the LR333x0 processor. A more detailed summary is provided in Chapter 3, "Instruction Set Summary," and a complete description of each instruction is contained in LSI Logic's *LR33000 Family Instruction Set Guide*.

System Control Coprocessor (CP0)

The LR333x0's system control coprocessor (CP0) supports exception handling functions of the LR333x0. Figure 1.4 summarizes the register set. Refer to Chapter 4, "Exception Processing," for descriptions of the exception-processing registers.

Figure 1.4 The CP0 Exception-Handling Registers

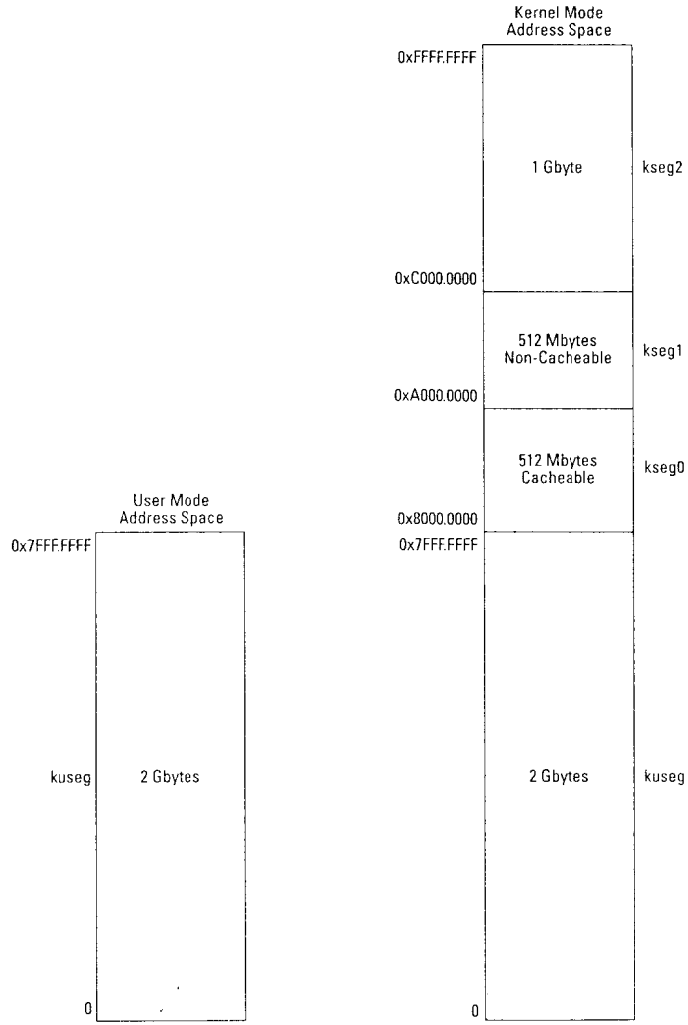


Operating Modes

To facilitate the separation of user and supervisory software, the LR333x0 has two operating modes: *user* and *kernel*. Normally, the processor operates in the user mode until an exception is detected, which forces it into the kernel mode. It remains in the kernel mode until a Restore From Exception (RFE) instruction is executed.

Figure 1.5 shows the address space for the two operating modes.

Figure 1.5 LR333x0 Address Space



User Mode – In user mode, a single, uniform address space (kuseg) of 2 Gbytes is available.

Kernel Mode – Four separate segments are defined in kernel mode:

- *kuseg* – References to this 2-Gbyte segment are treated just like user mode references, thus streamlining kernel access to user data. The cacheability of data and instructions is controlled on a per access basis by the $\overline{\text{CACHD}}$ signal.
- *kseg0* – References to this 512-Mbyte segment use cache memory and are hard-mapped to the first 512 Mbytes of memory. The cacheability of data and instructions is controlled on a per access basis by the $\overline{\text{CACHD}}$ signal.
- *kseg1* – References to this 512-Mbyte segment do not use the cache and are hard-mapped into the same 512-Mbyte segment of memory space as *kseg0*.
- *kseg2* – References to this 1-Gbyte segment are not mapped and the cacheability of data and instructions is controlled on a per access basis by the $\overline{\text{CACHD}}$ signal.

1.3 Pipeline Architecture

The execution of a single LR333x0 instruction consists of five primary steps or *pipestages*:

Step 1. **IF** – Fetch the instruction from the instruction cache (I-Cache).

Step 2. **RD** – Read any required operands from CPU registers while decoding the instruction.

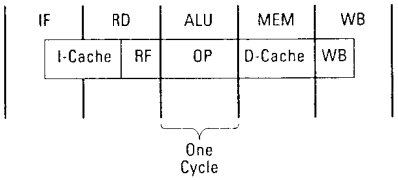
Step 3. **ALU** – Perform the required arithmetic or logical operation on instruction operands.

Step 4. **MEM** – Access memory from the data cache (D-Cache).

Step 5. **WB** – Write results back to register file.

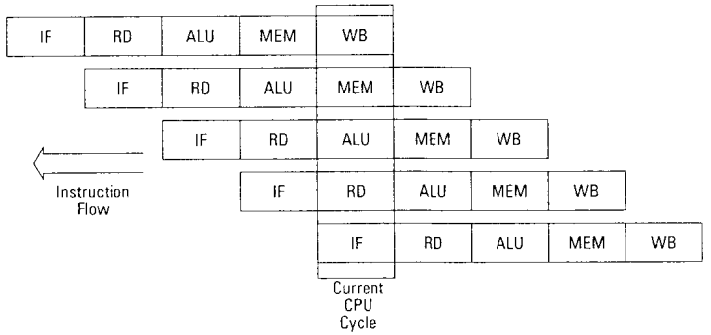
Each of these steps requires approximately one CPU cycle as shown in Figure 1.6 (parts of some operations overlap into another cycle while other operations require only 1/2 cycle).

Figure 1.6 Instruction Execution Sequence



The LR333x0 uses a five-stage pipeline to achieve an instruction execution rate approaching one instruction per CPU cycle. Thus execution of five instructions at a time are overlapped as shown in Figure 1.7.

Figure 1.7 LR333x0 Instruction Pipeline



This pipeline operates efficiently because different CPU resources (address and data bus accesses, ALU operations, register accesses, and so on) do not interfere with each other. Refer to Chapter 3, "Instruction Set Summary," for a detailed discussion of the instruction pipeline.

1.4 Memory System Hierarchy

A primary goal of systems employing RISC techniques is to achieve an instruction execution rate of one instruction per CPU cycle. To achieve this goal the MIPS architecture incorporates a number of RISC techniques including a compact and uniform instruction set, a deep instruction pipeline (as described above), utilization of optimizing compilers, and separate data and instruction caches (Harvard architecture).

These features combine to provide excellent performance; however, separate data and instruction caches boost performance only if the processor can access both in the same cycle. In the R2000 and R3000 microprocessors, single-cycle access to both caches is provided by multiplexing two memory accesses on a single bus, which makes the cache interface for

R2000/R3000-based systems difficult to design and expensive, especially for cost-sensitive embedded applications.

On-Chip Data and Instruction Cache Memories – The LR333x0 addresses the cache interface problem by incorporating the data and instruction caches on the same chip as the processor. As shown in Figure 1.1, the LR333x0 has separate internal data and instruction buses so that the processor can access both caches in a single cycle. Because the LR333x0 has separate data and instruction buses, the processor can obtain data and instructions at the cycle rate of the CPU.

The cache memories hold instructions and data that are repetitively accessed by the CPU (for example, within a program loop) and thus reduce the number of references that must be made to the slower main memory. Although smaller than the caches commonly used for reprogrammable applications, embedded applications can be tuned to run performance-critical code from the on-chip caches, thus providing excellent performance at a much lower cost.

Integral Write and Read Buffers – To ensure data consistency, all data that are written to the data cache must also be written out to main memory. To relieve the processor of this responsibility and thus eliminate processor stalls during write accesses, the LR333x0 includes a four-word-deep write buffer. The write buffer captures data (and associated addresses) output by the processor and ensures that the data are passed on to main memory.

The LR333x0 implements instruction streaming. The LR333x0 contains a one-word-deep read buffer, which enables the processor to stream regardless of the refill speed. The read buffer allows the CPU to access the instructions as they are being placed into the cache.

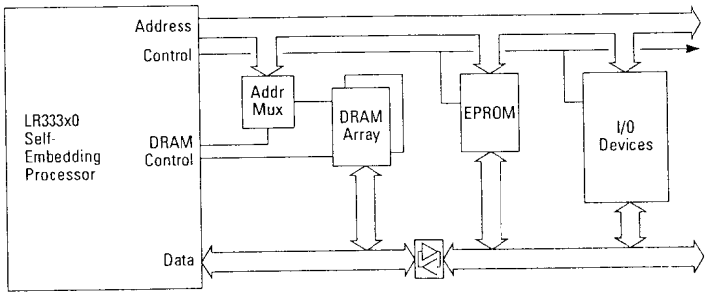
Simple Memory Interface – By bringing the cache memories on-chip, LSI Logic has simplified the interface to off-chip memory subsystems and peripherals. The LR333x0's memory interface consists of separate 32-bit address and 32-bit data buses that can be connected directly to many devices, including I/O controller chips, EPROMs, SRAMs, DRAMs, and SDRAMs.

Integral DRAM Controller – To further simplify the designer's task, the LR333x0 incorporates a DRAM Controller. With the addition of an address multiplexer, the LR333x0 can directly support standard, page-mode DRAMs. Through the DRAM Configuration Register, the LR333x0 can be configured to support synchronous DRAMs and provide interleaving support.

The combination of a simple general-purpose memory interface and the ability to directly support DRAMs makes it possible for designers to construct embedded systems that provide very high performance at a low system cost. Figure 1.8 illustrates such a system.

The DRAM Controller can be disabled and an off-chip controller used for systems with special needs.

Figure 1.8 LR333x0 Subsystem Interfaces



1.5 Comparison of the LR333x0 and the LR33000

The LR33300 and LR33310 contain various enhancements over the LR33000. This section compares the features of the processors and lists the differences in the processors's pin functions.

Features Comparison

Table 1.2 compares the features of the three processors. Brief descriptions of some of the new features in the LR333x0 follows the table.

Table 1.2
LR333x0 and LR33000
Comparison

	LR33000	*LR33300*	*LR33310*
Operating Frequency (Sustained Performance)	25, 33, 40 MHz (11, 14, 17 MIPS)	20, 25 MHz (11, 14 MIPS)	33, 40, 50 MHz (21, 26, 32 MIPS)
Technology	LCB007K	LCB310K	LCB310K
D-Cache Size	1K Direct Mapped	2K Direct Mapped	4K Direct Mapped
I-Cache Size	8K Direct Mapped	4K Direct Mapped	8K 2-Way Set Associative
Integral Write Buffer	One Word Deep	Four Words Deep	Four Words Deep
Interrupt Polarity	Active High	Programmable	Programmable
Snooping	D-Cache	I-Cache and D-Cache	I-Cache and D-Cache
Refresh Timer On/Off Control	Always On	On/Off Control	On/Off Control
Block Refill Operation	Starts at Word 0	Starts at Missed Word	Starts at Missed Word
Branch Taken Indicator	No	Yes	Yes
Wait-State Address Spaces	2	5	5
Clock Edge for Data and Parity Ready on Reads	Rising Edge	Programmable	Programmable
Dead Cycle Between Transactions	Yes	Programmable	Programmable
Read Priority	No	Programmable	Programmable
Lockable I-Cache	No	No	Yes
Scratchpad RAM Option	No	Yes	Yes
Synchronous DRAM Support	No	Yes	Yes
Programmable Interleaving Support	No	Yes	Yes
SRAM Address Space	No	Yes	Yes
Instruction Streaming	No	Programmable	Programmable
Load Scheduling	No	Programmable	Programmable
Byte and Halfword Gathering	No	Yes	Yes
Byte and Halfword Steering	No	Yes	Yes
Multiply Instruction Execution Time	12 Cycles	4, 7, or 12 Cycles	4, 7, or 12 Cycles

Load Scheduling – This feature determines whether a load is implemented immediately or is delayed depending on whether the load data is required yet. The LR33000 stalls in the MEM stage of the operation until the load data is available. If there are no data dependencies in the CPU pipeline, the LR333x0 does not stall in the MEM stage. Refer to the subsection entitled "Load Scheduling" on page 3-18 for more information.

Instruction Streaming – This feature is implemented during instruction block refills. Streaming starts at the missed word in a block refill. During streaming, the LR333x0 both fills the instruction cache with the instruction and uses that instruction. Refer to Section 10.5, "Instruction Streaming," for more information.

Read Priority – The LR333x0 has an option that gives load operations priority over store operations. Refer to the description of the RDPRI field in Section 6.2, "BIU/Cache Configuration Register," for information on enabling this feature.

Optimized Block Refill – For all block refills, the LR33000 starts the refill operation at word 0 of the block. The LR333x0 starts refills on the missed address. For data block refills, the LR333x0 counts upward from the missed word to the last word of the block and wraps around to get the remaining entries in the block. For instruction block refills, the LR333x0 counts upward from the missed word to the last word of the block and stops. Refer to Section 10.4, "Block-Fetch Transactions," for more detailed information.

Dead Cycle Removal – The LR33000 added a dead cycle at the end of every transaction. The LR333x0 provides the option of removing that dead cycle, thus improving performance between transactions. Refer to the description of the NOPAD field in Section 6.2, "BIU/Cache Configuration Register," for information on enabling this feature.

Branch Interpretation Support – In the LR33000, when the cause of an exception is in the branch delay slot, the preceding branch instruction must be interpreted to determine where to resume normal execution. In the LR333x0, the branch taken bit in the Cause Register indicates whether the branch is taken or not. The Target Address Register in the LR333x0 holds the return address. The exception handler just loads this address into a register and jumps to that location. Refer to the subsection entitled "Cause Register" on page 4-8 and the subsection entitled "Target Address Register" on page 4-10 for more information.

Pin Comparison

An LR333x0 can directly replace an LR33000 in an existing system. Table 1.3 lists the pins that have different functions between the LR333x0 and the LR33000.

Table 1.3 LR33000/LR333x0 Pin Comparison

LR33000 Pin	*LR333x0 Pin*	*Description*
THIT	$\overline{\text{INIT8}}$	$\overline{\text{INIT8}}$ is read at reset to initialize the 8WIDE bit in the SPEC0 Wait-State Configuration Register. A LOW on this input at reset enables byte gathering in the PROM space. This input has an internal pullup resistor for LR33000 compatibility.
TDONE	$\overline{\text{INIT16}}$	$\overline{\text{INIT16}}$ is read at reset to initialize the 16WIDE bit in the SPEC0 Wait-State Configuration Register. A LOW on this input at reset enables halfword gathering in the PROM space. This input has an internal pullup resistor for LR33000 compatibility.
I$TST	VSS	This pin is tied LOW.
D$TST	INTMASK	INTMASK allows the external system to mask any interrupts going to the CPU.
BRTKN	BRTKN	The LR333x0 BRTKN is an input during reset. The value on BRTKN is read at reset to initialize the WAITENA bit in the SPEC0 Wait-State Configuration Register. If BRTKN is HIGH, then the LR333x0 generates internal ready signals, otherwise, the external system generates the ready signal. This input has an internal pullup resistor for LR33000 compatibility.
$\overline{\text{DMAR}}$	$\overline{\text{DMAR}}$	The LR333x0 $\overline{\text{DMAR}}$ is also used as a DMA request signal when the bus is granted. The internal DRAM Controller starts a DRAM transaction as if $\overline{\text{AS}}$ had been asserted.
INT[5:0]	INT[5:0]	The LR333x0 interrupt pins have programmable polarity. The INTP bit in the BIU/Cache Configuration Register determines whether the polarity is negative or positive.
DP[3:0]	DP[3:0]	When the synchronous DRAM mode is enabled in the LR333x0, DP[3:0] act as SDRAM output enables.

Note When replacing an LR33000 with an LR333x0, pin 158 must be tied to VDD.

Register Comparison

Table 1.4 lists the register differences between the LR333x0 and the LR33000.

Table 1.4 LR33000/LR333x0 Register Comparison

Register	*Description*
Breakpoint PC Mask Register (BPCM)	New register in the LR333x0.
Breakpoint Data Address Mask Register (BDAM)	New register in the LR333x0.
Target Address Register (TAR)	New register in the LR333x0.
BIU/Cache Control Register	New register in the LR333x0.
Cause Register	Bit 30 is reserved in the LR33000. Bit 30 is Branch Taken in the LR333x0.
Status Register	Bit 16 is reserved in the LR33000. Bit 16 is Isolate Cache in the LR333x0.
Processor Revision ID Register (PRId)	This register is hardwired to zero in the LR33000. For the LR333x0, the value in this register contains an implementation and a revision field.
DRAM Configuration Register	New register in the LR333x0.
Configuration Register	This 32-bit register in the LR333x0 was removed from the LR333x0.
Refresh Timer Initial Count (RTIC) Register	The RTIC Register in the LR33000 contains a 12-bit count field. The RTIC Register in the LR333x0 contains a 16-bit count field and a timer disable bit.
Timer Initial Count (TIC) Registers	The timer count field in the TIC Registers in the LR33000 are 24 bits wide. The timer count field in the LR333x0 are expanded to 32 bits.
Debug and Cache Invalidate Control (DCIC) Register	Bits 13 and 12 in the LR33000 DCIC Register are the D-Cache and I-Cache enable bits. These bits are reserved in the LR333x0 DCIC Register.
Wait-State Configuration Registers	New registers in the LR333x0.

1.6 Compatibility with the R2000 and R3000 Microprocessors

The LR333x0 Self-Embedding Processor provides instruction set compatibility with the R2000 and R3000 microprocessors. Therefore, software developed for the R2000 and R3000 can be easily ported to systems based on the LR333x0, significantly reducing software development time and resource requirements.

However, the LR333x0 does not support external coprocessors and does not implement the R2000/R3000 memory management unit (MMU). In

addition, the internal cache memories are managed differently and two hardware breakpoint registers have been added to facilitate software debugging. These differences are further described below.

Coprocessor Support

The LR333x0 does not support external coprocessors. To prevent unpredictable results should a coprocessor instruction be decoded, all external coprocessors should be disabled by setting the appropriate coprocessor usability bits in CP0's Status Register to zero. When the coprocessors are disabled in this way, the detection of a coprocessor instruction causes a Coprocessor Unusable Exception. Table 1.2 lists the unsupported coprocessor instructions.

The LR333x0 provides four coprocessor condition signals that software can test using the Branch on Coprocessor z True and Branch on Coprocessor z False (BCzT and BCzF) instructions. This facility allows software to directly monitor hardware that is connected to the condition signals. The corresponding coprocessor must be enabled to use the test instruction. See Chapter 3, "Instruction Set Summary," for a description of the BCzT and BCzF instructions and Chapter 8, "Signal Definitions," for a description of the condition signals.

Table 1.5 Unimplemented Instructions

Op	*Description*
Coprocessor Instructions	
LWCz	Load Word from Coprocessor
SWCz	Store Word to Coprocessor
MTCz	Move To Coprocessor[1]
MFCz	Move From Coprocessor[1]
CTCz	Move Control To Coprocessor
CFCz	Move Control From Coprocessor
COPz	Coprocessor Operation
System Control Coprocessor (CP0) Instructions	
TLBR	Read Indexed TLB entry
TLBWI	Write Indexed TLB entry
TLBWR	Write Random TLB entry
TLBP	Probe TLB for matching entry

1. The LR333x0 implements this instruction for CP0 when in kernel mode.

Memory Management Unit Support

The LR333x0 does not implement the R2000/R3000 memory management unit, which includes the Translation Lookaside Buffer (TLB). Consequently, the LR333x0 does not implement any of the instructions designed to maintain and use the TLB. If the processor detects a TLB instruction, it causes a Reserved Instruction Exception. Table 1.2 lists the unimplemented instructions. In addition, the LR333x0 does not implement the TLB registers in CP0: EntryHi, EntryLo, Index, Random, and Context. References to those registers cause a Reserved Instruction Exception.

The System Control Processor: CP0

In addition to the removal of the TLB registers, the LR333x0's implementation of CP0 includes additional debug and control registers, which are summarized below. Chapter 4, "Exception Processing," contains a detailed description of CP0.

Debug Registers – The LR333x0 includes program counter and data address breakpoint registers to simplify software debugging. These registers have been incorporated into CP0, and software accesses them using Move from Coprocessor and Move to Coprocessor (MFC0 and MTC0) instructions.

Debug and Cache Invalidate Control Register – This register contains the enable and status bits for the LR333x0's debug mechanism and the control bits for the Cache Controller's invalidate mechanism. (The LR333x0 does not implement the cache control bits, which are located in the R2000/R3000 Status Register.)