LSI LOGIC

LR2010 Floating-Point Accelerator Preliminary

Description

The LR2010 Floating-Point Accelerator (FPA) provides high-speed, floating-point capability for systems based on the LR2000 CPU. The organization of FPA architecture is similar to that of the CPU, allowing high-level language compilers to optimize both integer and floating-point performance. The LR2010, with associated system software, fully conforms to the requirements and recommendations of the ANSI/IEEE Standard 754-1985. The LR2010 connects seamlessly to the CPU. Since both units receive instructions in parallel, floating-point instructions can be initiated at the same single cycle rate as fixed-point instructions.

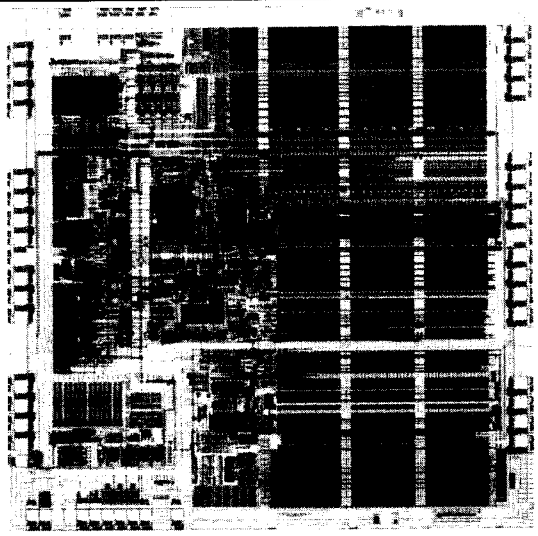
Features

- Fully compatible to ANSI/IEEE Standard 754-1985 floating-point arithmetic
- Supports single and double precision data formats
- High speed throughput, low latency
- Two speed versions

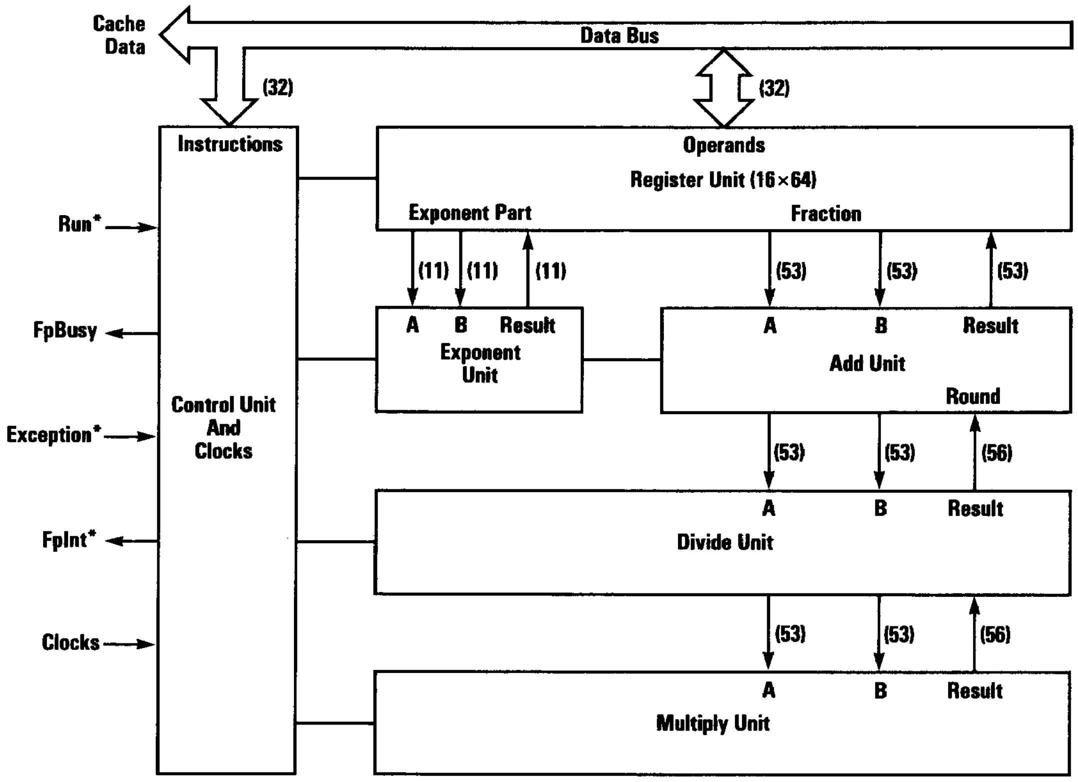
LR2010LC-12	12.5 MHz
LR2010LC-16	16.7 MHz

- Highly pipelined architecture coupled with optimizing compilers generates high throughput.
- Load/store oriented instruction set initiates floating-point instructions in a single cycle and overlaps execution with additional fixed or floating-point instructions.
- Status/control registers implemented to provide access to all IEEE Standard exception handling capability.
- Sixteen on-chip 64-bit registers individually accessible for flexible operation
- Complete instruction set
  - Single and double precision multiply, divide, add, subtract, negate, absolute value
  - Conversion to/from all supported formats
  - Comparison instructions derived from predicates named in IEEE Standard
- 84-pin ceramic leaded chip carrier
- LR2010 FPA performance floating-point benchmarks

■ Linpack	
– Single precision	4.8 MFlops
– Double precision	2.2 MFlops
■ Whetstone	
– Single precision	11.4 MWips
– Double precision	9.1 MWips
■ Livermore loops	
– Single precision	9.6×VAX 11/780
– Double precision	12.1×VAX 11/780
■ Spice	9.7×VAX 11/780
■ 256-Point FFT	23×VAX 11/780


LR2010 FPA Chip Photo

Block Diagram



Note: An asterisk * indicates an Active-LOW Signal.

Figure 1. Functional Block Diagram

Coprocessor Operation

The LR2010 FPA serves as a seamlessly integrated coprocessor in floating-point intensive LR2000-based systems. The FPA continually monitors the LR2000 instruction stream. If an instruction does not apply to the coprocessor, it is ignored. If an instruction does apply to the coprocessor, the FPA executes the instruction and transfers results and necessary exception data synchronously to the memory. The FPA performs three types of operations:

- Loads and stores
- Moves
- Two and three-register floating-point operations.

FPA Pipeline Architecture

The execution of a single LR2010 instruction consists of six primary steps:

IF	Instruction Fetch. The main processor calculates the instruction address required to read an instruction from the I-cache. No action is required of the FPA during this pipe stage since the main processor is responsible for address generation.
RD	The instruction is present on the data bus during phase 1 of this pipe stage. The FPA decodes the data and determines whether the instruction will be executed.
ALU	If the decoded instruction applies to the FPA, execution commences during this pipe stage.
MEM	If the instruction is a coprocessor load or store, the FPA captures or presents data during phase 2 of this pipe stage.
WB	The FPA uses this pipe stage to deal with exceptions.
FWB	During this stage the ALU writes results back to the register file. This stage is equivalent to the WB stage in the LR2000 processor.

The LR2010 architecture contains a pipeline similar to the LR2000 processor. The FPA pipeline contains six stages in contrast to the five-stage CPU, providing efficient coordination of exception responses between the FPA and the main processor. Such an architecture operates efficiently because different FPA resources (address and data bus accesses, ALU operations, register accesses, etc.) are utilized on a non-interfering basis. With the use of optimizing compilers to keep the pipeline full, the LR2010 achieves an instruction rate approaching one instruction per second.

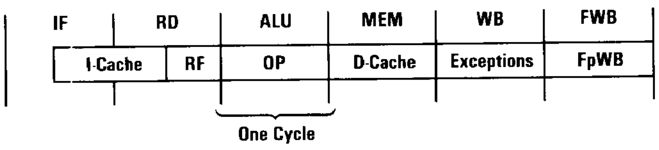


Figure 2. FPA Instruction Execution Sequence

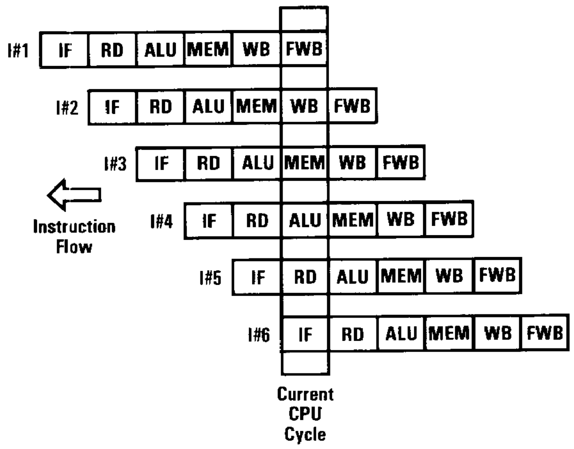


Figure 3. FPA Instruction Pipeline

Programming Model

The LR2010 contains sixteen 64-bit floating-point registers. These are intended to provide a sufficient number of floating-point registers to support allocation of scalar floating-point values and to permit overlapping execution and efficient scheduling of floating-point operations. Each register can hold one value of a single- or double-precision format floating-point number. Extended precision or quad precision floating-point formats can be accommodated by combining adjacent registers.

The coprocessor also contains control and status registers used primarily with diagnostic software, exception handling, state saving and restoring, and control of rounding modes.

The LR2010 FPA provides three types of registers shown in Figure 4.

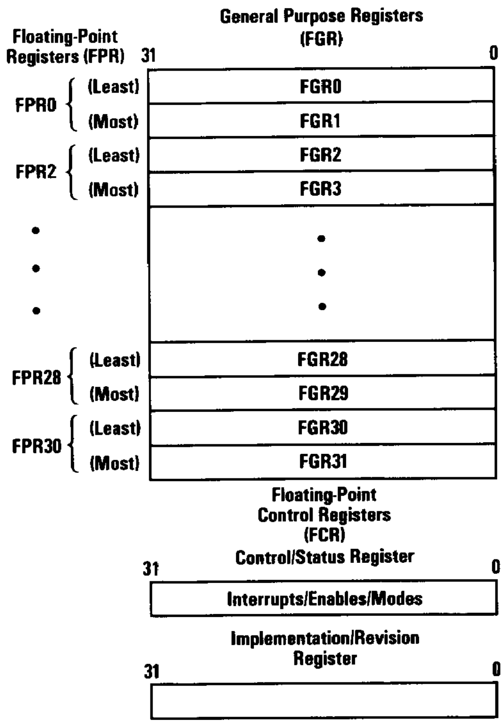


Figure 4. FPA Registers

Floating-point general purpose registers (FGR) are directly addressable, physical registers. The FPA provides thirty-two 32-bit FGRs individually accessable via move, load and store operations.

Table 1. Floating-Point General Registers

FGR Number	Usage
0	FPR 0 (Least)
1	FPR 0 (Most)
2	FPR 2 (Least)
3	FPR 2 (Most)
•	•
•	•
•	•
28	FPR 28 (Least)
29	FPR 28 (Most)
30	FPR 30 (Least)
31	FPR 30 (Most)

Floating-point registers (FPR) are logical registers used to store data values for floating-point operations. Each of the FPRs is 64 bits wide and is formed by concatenating two FGRs. The FPRs may hold either single- or double-precision format numbers. Only even-numbered addresses are used to address: odd-numbered register numbers are invalid. During single-precision operations only the even-numbered registers are used. Double-precision operations access general registers in pairs. For example, in a double-precision operation, selecting FPR0 addresses the adjacent floating-point general purpose registers FGR0 and FGR1.

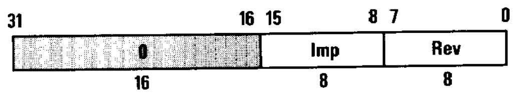
Floating-point control registers (FCR) are used for rounding mode control, exception handling, and state saving. LR2000 coprocessors, in general, can have up to 32 control registers. The FPA implements two: the control/status register (FCR31) and the implementation/revision (FCR0) register.

Programming Model
(Continued)

The control/status register contains control and status data that can be accessed by instructions running in either kernel or user mode. It controls the arithmetic rounding mode, the enabling of exceptions, and exception status. Bit assignments are shown in Figure 5.

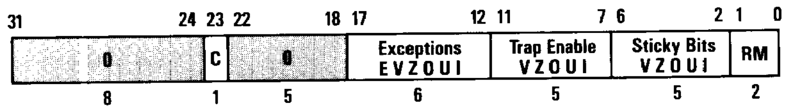
The bits in the control/status register can be set or cleared by writing to the register using a move control to coprocessor 1 (ctc1) instruction. The register must only be written to when the FPA is not actively executing floating-point operations. This can be assured by first reading the contents of the register to empty the pipeline. If a floating-point exception occurs as the pipeline empties, the exception is taken and the CFC1 instruction can be re-executed after the exception is serviced.

The FPA control register 0 (FCR0) contains values that define the implementation and revision number of the LR2010 FPA. This information can be used by diagnostic software to determine the coprocessor revision level. Only the low order bytes are defined. Bits 15 through 8 identify the implementation and bits 7 through 0 identify the revision number as shown in Figure 6.



Imp	Implementation: 0×10=LR2010.
Rev	Revision of FPA.
$\boxed{0}$	Unused; ignored on writes, zero when read.

Figure 6. Implementation/Revision Register



C	Condition bit. Set/cleared to reflect the result of a compare instruction and drives the FPA CpCond output signal.
Exceptions	These bits are set to indicate any exceptions that occurred during the most recent instructions.
Trap Enable	These bits enable assertion of the CpInt* signal if the corresponding exception bit is set during a floating-point operation.
Sticky Bits	These bits are set if an exception occurs and are reset only by explicity loading new settings into this register (with a move instruction).
RM	Rounding Mode. These two bits specify which of the four rounding modes is to be used by the FPA.
$\boxed{0}$	Reserved. Currently ignores writes, undefined when read.

Figure 5. Control/Status Register Bit Assignments

Floating-Point Formats

The LR2010 FPA supports both 32-bit single-precision and 64-bit double-precision IEEE Standard floating-point formats. The 32-bit format has a 24-bit signed magnitude fraction field and an 8-bit exception, as shown in Figure 7.

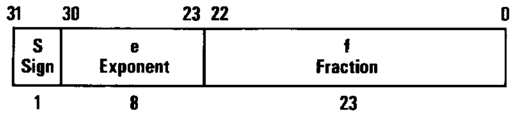


Figure 7. Single-Precision, Floating-Point Format

The 64-bit format has a 53-bit signed magnitude fraction field and an 11-bit exponent, as shown in Figure 8.

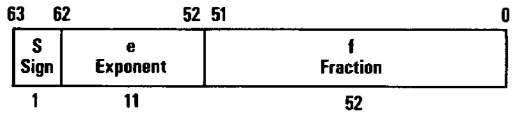


Figure 8. Double-Precision, Floating-Point Format

Floating-point representations in the LR2010 are composed of three fields:

1. A 1-bit sign: s
2. A biased exponent: e = E + bias
3. A fraction: f = .b1b2 . . . bp-1

The range of unbiased exponent E includes every integer between two values EMin and EMax inclusive, and also two other reserved values: EMin − 1 to encode ±0 and denormalized numbers, and EMax + 1 to encode ± ∞ and NaNs (Not-A-Number). For single- and double-precision formats, each representable non-zero value has just one encoding.

The value of a floating-point number is shown in Table 2.

Table 2. Equations for Calculating Values in Floating-Point Format

(1)	if $E = EMax + 1$ and $f \neq 0$, then v is NAN, regardless of s.
(2)	if $E = EMax + 1$ and $f = 0$, then $v = (-1)^S \infty$
(3)	if $EMin \leq E \leq EMax$, then $v = (-1)^S 2^E (1.f)$
(4)	if $E = EMin - 1$ and $f \neq 0$, then $v = (-1)^S 2^{EMin} (0.f)$
(5)	if $E = EMin - 1$ and $f = 0$, then $v = (-1)^S 0$

For all floating-point formats, if v is a NaN, the most significant bit of f determines whether the value is a signaling NaN or a quiet NaN. The most significant bit of f will be set for signaling NaN.

The values for the parameters described are shown in Table 3.

Table 3. Floating-Point Format Parameter Values

Parameter	Single	Double
P	24	53
EMax	+127	+1023
EMin	−126	−1022
Exponent Bias	+127	+1023
Exponent Width in Bits	8	11
Integer Bit	Hidden	Hidden
Fraction Width in Bits	23	52
Format Width in Bits	32	64

Number Definitions

The IEEE Standard 754-1985 specifies four varieties of numbers that must be represented: normalized numbers, denormalized numbers, infinity, and zero. The definition of each number type in the LR2010 follows:

Normalized Numbers

Most floating-point calculations are performed on normalized numbers. For single-precision operations, normalized numbers have a biased exponent that ranges from 1 to 254 (−126 to +127 unbiased) and a normalized fraction field, meaning that the leftmost (hidden) bit is one. In decimal notation this allows representation of a range of positive and negative values from approximately 10^{38} to 10^{-38}, with accuracy to seven decimal places.

Denormalized Numbers

Denormalized numbers have a zero exponent and a denormalized (hidden bit = 0) non-zero fraction field.

Infinity

Inifinity has an exponent of all ones and a fraction field equal to zero. Both positive and negative infinity are supported.

Zero

Zero has an exponent of zero, a hidden bit equal to zero, and a value of zero in the fraction field. Both +0 and −0 are supported.

Instruction Set Summary

The floating-point instructions supported by the LR2010 are all implemented using the coprocessor unit 1 (COP1) operation instructions of the LR2000 CPU instruction set. The basic operations performed by the CPU are:

- Load/store operations from/to the FPA registers
- Moves between the CPU and the FPA registers
- Computational instructions including floating-point add, subtract, multiply, divide and convert instructions
- Floating-point comparisons

Load, Store and Move Instructions

All movement of data between the LR2010 FPA and memory is accomplished by load word to coprocessor 1 (LWC1) and store word to coprocessor 1 (SWC1) instructions which reference a single 32-bit word of the FPAs general registers. These loads and stores are unformatted; no format conversions are performed and therefore no floating-point exceptions occur due to these operations.

Instruction Set Summary (Continued)

Data may also be directly moved between the FPA and the LR2000 CPU by the move to coprocessor 1 (MTC1) and move from coprocessor 1 (MFC1) instructions. Like the floating-point load and store operations, these operations perform no format conversions and never cause floating-point exceptions. The load and move instructions have a latency of one instruction. Data being loaded from memory or the CPU into an FPA register is not available to the instruction that immediately follows the load instruction. Data becomes available to the second instruction following the load.

Table 4 summarizes the LR2010 load, store and move instructions.

Table 4. FPA Load, Store and Move Instruction Summary

Instruction	Format and Description
Load Word to FPA (Coprocessor 1)	*LWC1 ft,Offset(Base)* Sign-extend 16-bit *offset* and add to contents of CPU register *base* to form address. Load contents of addressed word into FPA general register *ft*.
Store Word from FPA (Coprocessor 1)	*SWC1 ft,Offset(Base)* Sign-extend 16-bit *offset* and add to contents of CPU register *base* to form address. Store 32-bit contents of FPA general register *ft* at addressed location.
Move Word to FPA (Coprocessor 1)	*MTC1 rt,fs* Move contents of CPU register *rt* into FPA register *fs*.
Move Word from FPA (Coprocessor 1)	*MFC1 rt,fs* Move contents of FPA general register *fs* into CPU register *rt*.
Move Control Word to FPA (Coprocessor 1)	*CTC1 rt,fs* Move contents of CPU register *rt* into FPA control register *fs*.
Move Control Word from FPA (Coprocessor 1)	*CFC1 rt,fs* Move contents of FPA control register *fs* into CPU register *rt*.

Instruction Set Summary (Continued)

Computational Instructions
Computational instructions perform arithmetic operations on floating-point values in registers. There are four categories of floating-point computational instruction:

- 3-operand register-type instructions that perform floating-point addition, subtraction, multiplication and division operations.
- 2-operand register-type instructions that perform floating-point absolute value, move and negate operations
- Convert instructions that perform conversions between the various formats
- Compare instructions that perform comparisons of the contents of two registers and set or clear a condition flag based on the result of the comparison.

Table 5 summarizes the computational instructions. The fmt term appended to the instruction op code is the data format specifier: s specifies single-precision binary floating point, d specifies double-precision binary floating point, and w specifies fixed point. When fmt is single precision or fixed point, the odd register of the destination is undefined.

Table 5. FPA Computational Instruction Summary

Instruction	Format and Description
Floating-Point Add	*ADD.fmt fd,fs,ft* Interpret contents of FPA registers *fs* and *ft* in specified format *(fmt)* and add arithmetically. Place rounded result in FPA register *fd*.
Floating-Point Subtract	*SUB.fmt fd,fs,ft* Interpret contents of FPA registers *fs* and *ft* in specified format *(fmt)* and arithmetically subtract *ft* from *fs*. Place result in FPA register *fd*.
Floating-Point Multiply	*MUL.fmt fd,fs,ft* Interpret contents of FPA registers *fs* and *ft* in specified format *(fmt)* and arithmetically multiply *ft* and *fs*. Place result in FPA register *fd*.
Floating-Point Divide	*DIV.fmt fd,fs,ft* Interpret contents of FPA registers *fs* and *ft* in specified format *(fmt)* and arithmetically divide *fs* by *ft*. Place rounded result in register *fd*.
Floating-Point Absolute Value	*ABS.fmt fd,fs* Interpret contents of FPA register *fs* in specified format *(fmt)* and take arithmetic absolute value. Place result in FPA register *fd*.
Floating-Point Move	*MOV.fmt fd,fs* Interpret contents of FPA register *fs* in specified format *(fmt)* and copy into FPA register *fd*.
Floating-Point Negate	*NEG.fmt fd,fs* Interpret contents of FPA register *fs* in specified format *(fmt)* and take arithmetic negation. Place result in FPA register *fd*.
Floating-Point Convert to Single FP Format	*CVT.S.fmt fd,fs* Interpret contents of FPA register *fs* in specified format *(fmt)* and arithmetically convert to the single binary floating-point format. Place rounded result in FPA register *fd*.
Floating-Point Convert to Double FP Format	*CVT.D.fmt fd,fs* Interpret contents of FPA register *fs* in specified format *(fmt)* and arithmetically convert to the double binary floating-point format. Place rounded result in FPA register *fd*.
Floating-Point Convert to Single Fixed-Point Format	*CVT.W.fmt fd,fs* Interpret contents of FPA register *fs* in specified format *(fmt)* and arithmetically convert to the single fixed-point format. Place result in FPA register *fd*.
Floating-Point Compare	*C.cond.fmt fs,ft* Interpret contents of FPA registers *fs* and *ft* in specified format *(fmt)* and arithmetically compare. The result is determined by the comparison and the specified condition *(cond)*. After a one instruction delay, the condition is available for testing by the CPU with the *branch on floating-point coprocessor condition (BC1T, BC1F)* instructions.

LSI LOGIC

Instruction Set Summary (Continued)

Floating-Point Relational Operations

The floating-point compare instructions (C.fmt.cond) interpret the contents of two FPA registers (fs, ft) in the specified format (fmt) and arithmetically compare them. The result is based on the comparison and the conditions (cond) specified in the instruction. Table 6 lists the conditions that can be specified for the compare instruction and Table 7 summarizes the floating-point relational operations that may be performed.

Table 7 is derived from a similar table in the IEEE Standard and describes 26 predicates named in the standard. The table also includes six additional predicates to round out the set of possible predicates based on a condition tested by a comparison. Four mutually exclusive relations are possible: less than, greater than, equal, and unordered. Note that invalid operations occur only when the comparisons include the less-than and greater-than characters but not the unordered character in the ad hoc form of the predicate.

Branch on FPA Condition Instructions

Table 8 summarizes the two branch on FPA (coprocessor unit 1) condition instructions that can be used to test the result of the FPA compare instructions. The term delay slot, described in the table, refers to the instruction immediately following the branch instruction.

Table 6. Relational Mnemonic Definitions

Mnemonic	Definition	Mnemonic	Definition
F	False	T	True
UN	Unordered	OR	Ordered
EQ	Equal	NEQ	Not Equal
UEQ	Unordered or Equal	OLG	Ordered or Less Than or Greater Than
OLT	Ordered Less Than	UGE	Unordered or Greater Than or Equal
ULT	Unordered or Less Than	OGE	Ordered Greater Than
OLE	Ordered Less Than or Equal	UGT	Unordered or Greater Than
ULE	Unordered or Less Than or Equal	OGT	Ordered Greater Than
SF	Signaling False	ST	Signaling True
NGLE	Not Greater Than or Less Than or Equal	GLE	Greater Than, or Less Than or Equal
SEQ	Signaling Equal	SNE	Signaling Not Equal
NGL	Not Greater Than or Less Than	GL	Greater Than or Less Than
LT	Less Than	NLT	Not Less Than
NGE	Not Greater Than or Equal	GE	Greater Than or Equal
LE	Less Than or Equal	NLE	Not Less Than or Equal
NGT	Not Greater Than	GT	Greater Than

Instruction Set Summary (Continued)

Table 7. Floating-Point Relational Operators

Predicates			Relations				Invalid Operation Exception if Unordered
Condition Mnemonic	**Ad Hoc**	**FORTRAN**	**Greater Than**	**Less Than**	**Equal**	**Unordered**	
F	false		F	F	F	F	no
UN	?		F	F	F	T	no
EQ	=	.EQ.	F	F	T	F	no
UEQ	?=	.UE.	F	F	T	T	no
OLT	NOT(?>=)	.NOT. .UG.	F	T	F	F	no
ULT	?<	.UL.	F	T	F	T	no
OLE	NOT(?>)	.NOT. .UG.	F	T	T	F	no
ULE	?<=	.ULE.	F	T	T	T	no
OGT	NOT(?<=)	.NOT. .ULE	T	F	F	F	no
UGT	?>	.UGT.	T	F	F	T	no
OGE	NOT(?<)	.NOT. .UL.	T	F	T	F	no
UGE	?>=	.UGE.	T	F	T	T	no
OLG	NOT(?=)		T	T	F	F	no
NEQ	NOT(=)	.NE.	T	T	F	T	no
OR	NOT(?)		T	T	T	F	no
T	true		T	T	T	T	no
SF			F	F	F	F	yes
NGLE	NOT(<=>)	.NOT. .LEG.	F	F	F	T	yes
SEQ			F	F	T	F	yes
NGL	NOT(<>)	.NOT. LG.	F	F	T	T	yes
LT	<	.LT.	F	T	F	F	yes
NGE	NOT(>=)	.NOT. .GE.	F	T	F	T	yes
LE	<=	.LE.	F	T	T	F	yes
NGT	NOT(>)	.NOT. .GT.	F	T	T	T	yes
GT	>	.GT.	T	F	F	F	yes
NLE	NOT(<=)	.NOT. .LE.	T	F	F	T	yes
GE	>=	.GE.	T	F	T	F	yes
NLT	NOT(<)	.NOT. .LT.	T	F	T	T	yes
GL	<>	.LG.	T	T	F	F	yes
SNE			T	T	F	T	yes
GLE	<=>	.LEG.	T	T	T	F	yes
ST			T	T	T	T	yes

Table 8. Branch on FPA Condition Instructions

Instruction	Format and Description
Branch on FPA True	*BC1T* Compute a branch target address by adding address of instruction in the delay slot and the 16-bit *offset* (shifted left two bits and sign-extended to 32 bits). Branch to the target address (with a delay of one instruction) if the FPAs CpCond signal is true.
Branch on FPA False	*BC1F* Compute a branch target address by adding address of instruction in the delay slot and the 16-bit *offset* (shifted left two bits and sign-extended to 32 bits). Branch to the target address (with a delay of one instruction if the FPAs CpCond signal is false.

Instruction Execution Times

Unlike the LR2000 which executes nearly all its instructions in a single cycle, the time to execute an FPA instruction ranges from 1 cycle to 19 cycles. Figure 9 illustrates the number of cycles required to execute each of the FPA instructions. The cycles of an instruction's execution time that are darkly shaded require exclusive access to an FPA resource that precludes concurrent use by another instruction. With the exception of loads and stores, other FPA instructions cannot be overlapped during these cycles. Those instruction cycles that are lightly shaded place minimal demands on FPA resources and may be overlapped (with some exceptions) to obtain simultaneous execution without stalling the pipeline.

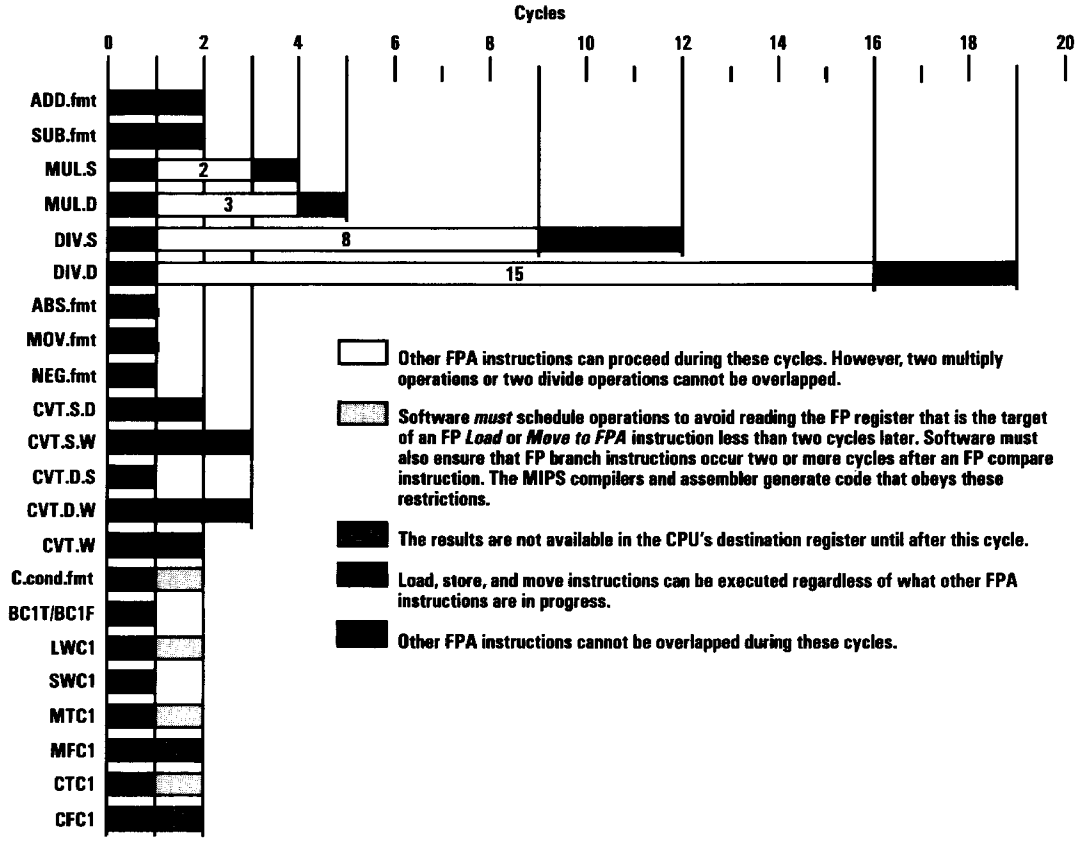


Figure 9. FPA Instruction Execution Times

Overlapping FPA Instructions

Figure 10 illustrates the overlapping of several FPA (and non-FPA) instructions. In this example, the first instruction requires 12 total cycles for execution but only the first cycle and the last three cycles inhibit simultaneous execution of other instructions. Similarly, the second instruction (MUL.S) has two cycles in the middle of its total of four required cycles that can be used to advance the execution of the third and fourth instructions.

Although processing of a single instruction consists of six pipe stages, the FPA does not require that the instruction actually be completed in six cycles to avoid stalling the pipeline. If a subsequent instruction does not require the resources being used by a preceding instruction and has no data dependencies on uncompleted instructions, then execution continues.

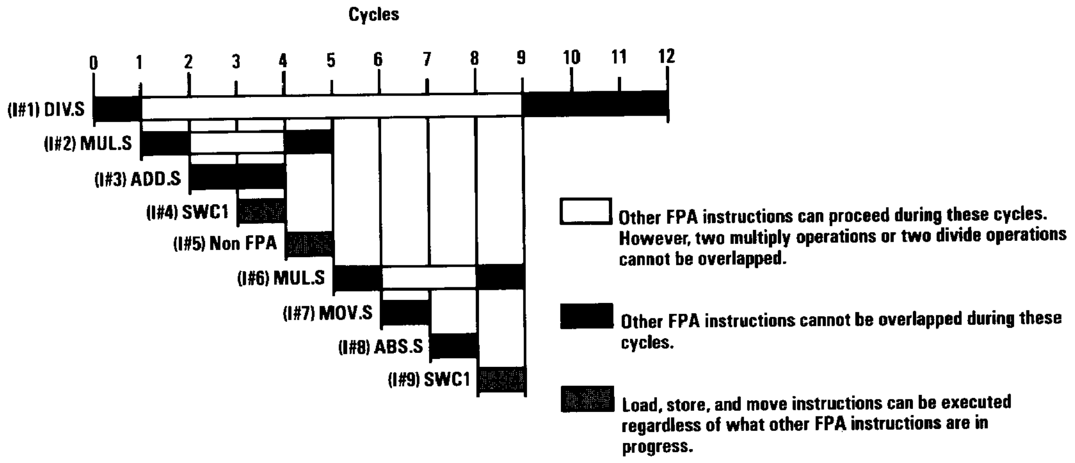

Cycles
0
1
2
3
4
5
6
7
8
9
10
11
12
(I#1) DIV.S
(I#2) MUL.S
(I#3) ADD.S
(I#4) SWC1
(I#5) Non FPA
(I#6) MUL.S
(I#7) MOV.S
(I#8) ABS.S
(I#9) SWC1
Other FPA instructions can proceed during these cycles. However, two multiply operations or two divide operations cannot be overlapped.
Other FPA instructions cannot be overlapped during these cycles.
Load, store, and move instructions can be executed regardless of what other FPA instructions are in progress.

Figure 10. Overlapping FPA Instructions

Floating-Point Exceptions

Floating-point exceptions occur when the FPA cannot handle the results of a floating-point operation in a normal way. The FPA responds by either generating an interrupt or setting a status flag. The control status register previously described contains a trap enable bit for each exception type that determines whether an exception will cause the FPA to initiate a trap or set a status flag. If a trap is taken, the FPA remains in the state found at the beginning of the operation and a software handling routine is executed. If no trap is taken, an appropriate value is written into the FPA destination register and execution continues.

The FPA supports the five IEEE exceptions — inexact (I), overflow (O), underflow (U), divide by zero (Z), and invalid (V) — with exception bits, trap enables and sticky bits. The LR2010 FPA adds a sixth exception type, unimplemented operation (E), to be used in those cases where a software implementation must be employed to conform to the MIPS floating-point architecture. The unimplemented operation exception has no trap enable or sticky bit. Whenever this exception occurs, an unimplemented exception trap is taken (if the FP interrupt input to the LR2000 is enabled).

Figure 11 shows the control/status register associated with the five IEEE exceptions (V,Z,O,I,U). When an exception occurs, the corresponding exception and sticky bits are set. If the corresponding trap enable bit is set, the FPA generates an interrupt to the LR2000 processor and subsequent exception processing allows a trap to be taken.

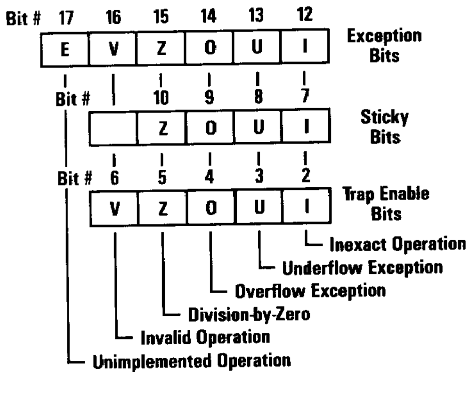


Figure 11. Control/Status Register Exception/ Sticky/Trap Enable Bits

Floating-Point Exceptions (Continued)

Exception Trap Processing

When a floating-point exception trap is taken, the LR2000s cause register indicates that an external interrupt is the cause of the exception and the LR2000s EPC (exception program counter) contains the address of the instruction that caused the exception trap.

For each IEEE Standard exception, a sticky-bit status flag is provided that is set on the occurrence of the corresponding condition with no corresponding exception trap signaled. The sticky bits may be reset by writing a new value into the control/status register and may be saved and restored by software.

When no exception trap is signaled, a default action is taken by the FPA which provides a substitute value for the original exceptional result of the floating-point operation. The default action depends on the type of exception and, in the case of overflow, the current rounding mode. Table 10 lists the default action taken by the FPA for each of the IEEE exceptions.

The FPA internally detects eight different conditions that can cause exceptions. When the FPA encounters one of these situations it will cause either an IEEE exception or an unimplemented operation (E) exception. Table 9 lists the exception-causing situations.

The following sections describe the conditions that cause the FPA to generate each of its six exceptions and details the FPAs response to each of these situations.

Table 9. FPA Exception Situations

FPA Internal Result	IEEE Standard	Trap Enabled	Trap Disabled	Note
Inexact Result	I	I	I	Loss of accuracy
Exponent Overflow	O I*	O I	O I	Normalized exponent > EMax
Divide by Zero	Z	Z	Z	Zero is (exponent = EMin − 1, mantissa = 0)
Overflow on Convert	V	V	E	Source out of integer range
Signaling NaN Source	V	V	E	Quiet NaN source produces quiet NaN result
Invalid Operation	V	V	E	0/0 etc.
Exponent Underflow	U	E	E	Normalized exponent < EMin
Denormalized Source	None	E	E	Exponent = EMin − 1 and mantissa < > 0

*Standard specifies inexact exception on overflow only if overflow trap is disabled.

Table 10. FPA Exception Default Actions

Exception		Rounding Mode	Default Action (No Exception Trap Signaled)
V	Invalid Operation	—	Supply a quiet NaN.
Z	Division by Zero	—	Supply a properly signed ∞.
O	Overflow	RN	Modify overflow values to ∞ with the sign of the intermediate result.
		RZ	Modify overflow values to the format's largest finite number with the sign of the intermediate result.
		RP	Modify negative overflows to the format's most negative finite number. Modify positive overflows to + ∞.
		RM	Modify positive overflows to the format's largest finite number. Modify negative overflows to − ∞.
U	Underflow	—	Generate an unimplemented exception.
I	Inexact	—	Supply a rounded result.

Floating-Point Exceptions (Continued)

Inexact Exception (I)

The FPA generates this exception if the rounded result of an operation is not exact or if it overflows.

The FPA usually examines the operands of floating-point operations before execution actually begins to determine (based on the exponent values of the operands) if the operation can *possibly* cause an exception. If there is a possibility of an instruction causing an exception trap, then the FPA uses the coprocessor stall mechanism previously described. It is impossible, however, for the FPA to predetermine if an instruction will produce an inexact result. Therefore, if inexact exception traps are enabled, the FPA uses the coprocessor stall mechanism to execute all floating-point operations that require more than one cycle. Since this mode of execution can impact performance, inexact exception traps should be enabled only when necessary.

Trap Enabled Results: If inexact exception traps are enabled, the result register is not modified and the source registers are preserved.

Trap Disabled Results: The rounded or overflowed result is delivered to the destination register if no other software trap occurs.

Underflow Exception (U)

The FPA never generates an underflow exception and never sets the *U* bit in either the *exceptions* field or *sticky* field of the control/status register. If the FPA detects a condition that could be either an underflow or a loss of accuracy, it generates an unimplemented exception.

Overflow Exception (O)

The overflow exception is signaled when what would have been the magnitude of the rounded floating-point result, were the exponent range unbounded, is larger than the destination format's largest finite number. (This exception also sets the inexact exception and sticky bits.)

Trap Enabled Results: The result register is not modified, and the source registers are preserved.

Trap Disabled Results: The result, when no trap occurs, is determined by the rounding mode and the sign of the intermediate result (as listed in Table 10).

Division-by-Zero Exception (Z)

The division-by-zero exception is signaled on a divide operation if the divisor is zero and the dividend is a finite non-zero number.

Trap Enabled Results: The result register is not modified, and the source registers are preserved.

Trap Disabled Results: The result, when no trap occurs, is a correctly signed infinity.

Invalid Operation Exception (V)

The invalid operation exception is signaled if one or both of the operands are invalid for an implemented operation. The invalid operations are:

1. Addition or subtraction: magnitude subtraction of infinities, such as: $(+\infty) - (+\infty)$
2. Mutliplication: $0 \times \infty$, with any signs
3. Division: $0 \div 0$, or $\infty \div \infty$, with any signs
4. Conversion of a floating-point number to a fixed-point format when an overflow, or operand value of infinity or NaN, precludes a faithful representation in that format
5. Comparison of predicates involving $<$ or $>$ without ?, when the operands are "unordered"
6. Any arithmetic operation on a signaling NaN. Note that a move (MOV) operation is not considered to be an arithmetic operation, but that ABS and NEG are considered to be arithmetic operations and will cause this exception if one or both operands is a signaling NaN.

Software may simulate this exception for other operations that are invalid for the given source operands. Examples of these operations include IEEE-specified functions implemented in software, such as remainder: x REM y, where y is zero or x is infinite; conversion of a floating-point number to a decimal format whose value causes an overflow or is infinity or NaN; and transcendental functions, such as $\ln(-5)$ or $\cos^{-1}(3)$.

Trap Enabled Results: The original operand values are undisturbed.

Trap Disabled Results: The FPA always signals an unimplemented exception because it does not create the NaN that the IEEE Standard specifies should be returned under these circumstances.

Floating-Point Exceptions (Continued)

Unimplemented Operation Exception (E)
The FPA generates this exception when it attempts to execute an instruction with an OpCode (bits 31–26) or format code (bits 24–21) which has been reserved for future use.

This exception is not maskable: the trap is always enabled. When an unimplemented operation is signaled, an interrupt is sent to the LR2000 processor so that the operation can be emulated in software. When the operation is emulated in software, any of the IEEE exceptions may arise; these exceptions must, in turn, be simulated.

This exception is also generated when any of the following exceptions are detected by the FPA:

- Extended and quad precision
- Square root
- Denormalized operand
- Not-a-number (NaN) operand
- Invalid operation with trap disabled
- Denormalized result
- Underflow

Trap Enabled Results: The original operand values are undisturbed.

Trap Disabled Results: This trap cannot be disabled.

Saving and Restoring State

Thirty-two coprocessor load or store instructions will save or restore the FPAs floating-point register state in memory. The contents of the control/status register can be saved using the "move to/from coprocessor control register" instructions (CTC1/CFC1). Normally, the control/status register contents are saved first and restored last.

If the control/status register is read when the coprocessor is executing one or more floating-point instructions, the instructions in progress (in the pipeline) are completed before the contents of the register are moved to the main processor. If an exception occurs during one of the in-progress instructions, that exception is written into the control/status register exceptions field.

Note that the exceptions field of the control/status register holds the results of only one instruction: the FPA examines source operands before an operation is initiated to determine if the instruction can possibly cause an exception. If an exception is possible, the FPA executes the instruction in "stall" mode to ensure that no more than one instruction at a time is executed that might cause an exception.

All of the bits in the exceptions field can be cleared by writing a zero value to this field. This permits restarting of normal processing after the control/status register state is restored.

Pin Descriptions

(**Note:** an asterisk * indicates an Active-LOW signal)

Data (31:0)
(I/O) A multiplexed 32-bit bus used for instruction and data transfers on phase 1 and phase 2, respectively.

Data P(3:0)
(O) A 4-bit bus containing even parity over the data bus. Parity is generated by the FPC on stores.

Run*
(I) Input to the FPC which indicates whether the processor-coprocessor system is in the run or stall state.

Exception*
(I) INput to the FPC which indicates exception related status information.

FpBusy
(O) Signal to the CPU indicating a request for a coprocessor busy stall.

FpCond
(O) Signal to the CPU indicating the result of the last comparison operation.

FpInt*
(O) Signal to the CPU indicating that a floating-point exception has occurred for the current FPC instruction.

Reset*
(I) Synchronous initialization input used to distinguish the processor-FPC synchronization period from the execution period. Reset* must be synchronized by the leading edge of SysOut from the CPU.

PLLOn*
(I) Input which during the reset period determines whether the phase lock mechanism is enabled and during the execution period determines the output timing model.

FpPresent*
(O) Output which is pulled to ground through an impedance of approximately 0.5K Ω. By providing an external pullup on this line, an indication of the presence or absence of the FPC can be obtained.

Clk2×Sys
(I) A double-frequency clock input used for generating FpSysOut*.

Clk2×Smp
(I) A double-frequency clock input used to determine the sample point for data coming into the FPC.

Clk2×Rd
(I) A double-frequency clock input used to determine the disable point for the data drivers.

Clk2×Phi
(I) A double-frequency clock input used to determine the position of the internal phases; phase 1 and phase 2.

FpSysOut*
(O) Synchronization clock from the FPC.

FpSysIn*
(I) Input used to receive the synchronization clock from the FPC.

FpSync*
(I) Input used to receive the synchronization clock from the CPU.

LR2010
Floating-Point
Accelerator
Preliminary

Pin Assignments

Table 11. FPC Pinout 84-Pin Quad J-Lead CerPak

Pin Name	Pin Number	Pin Name	Pin Number
Data(0)	33	FpSync*	23
Data(1)	34	Reset*	22
Data(2)	35	PllOn*	28
Data(3)	36	Run*	66
Data(4)	39	Exception*	67
Data(5)	40	FpInt*	68
Data(6)	41	FpBusy	69
Data(7)	42	FpCond	70
Data(8)	44	VCC0	7
Data(9)	45	VCC1	15
Data(10)	46	VCC2	24
Data(11)	47	VCC3	26
Data(12)	50	VCC4	29
Data(13)	51	VCC5	31
Data(14)	52	VCC6	38
Data(15)	53	VCC7	49
Data(16)	76	VCC8	55
Data(17)	77	VCC9	57
Data(18)	78	VCC10	61
Data(19)	79	VCC11	63
Data(20)	82	VCC12	72
Data(21)	83	VCC13	75
Data(22)	84	VCC14	81
Data(23)	1	Gnd0	6
Data(24)	3	Gnd1	16
Data(25)	4	Gnd2	25
Data(26)	5	Gnd3	27
Data(27)	8	Gnd4	30
Data(28)	9	Gnd5	32
Data(29)	10	Gnd6	37
Data(30)	11	Gnd7	48
Data(31)	14	Gnd8	54
DataP(0)	43	Gnd9	56
DataP(1)	73	Gnd10	60
DataP(2)	2	Gnd11	62
DataP(3)	17	Gnd12	71
Clk2xSys	19	Gnd13	74
Clk2xSmp	20	Gnd14	80
Clk2xRd	12	Resvd0	58
Clk2xPhi	21	Resvd1	64
FpSysIn*	13	Resvd2	65
FpSysOut*	18	FpPresent*	59

Note: An asterisk * indicates an Active-LOW signal

Operating Parameters

Absolute Maximum Ratings[1]

Parameter	Description	Min	Max	Units
VCC	Supply Voltage	−0.5	+7.0	V
VIN	Input Voltage	−0.5[2]	+7.0	V
TST	Storage Temperature	−65	+150	C
TA	Operating Temperature	0	+70	C
CLD	Load Capacitance on Any Pin		100	pF

Notes:
1. Operation beyond the limits set forth in this table may impair the useful life of the device.
2. VIN Min. = −3.0 V for pulse width less than 15 ns.
3. Not more than one output should be shorted at a time. Duration of the short should not exceed 30 seconds.

Operating Range

Range	Ambient Temperature	VCC
Commercial	0°C to 70°C	5V±5%

LR2010
Floating-Point
Accelerator
Preliminary

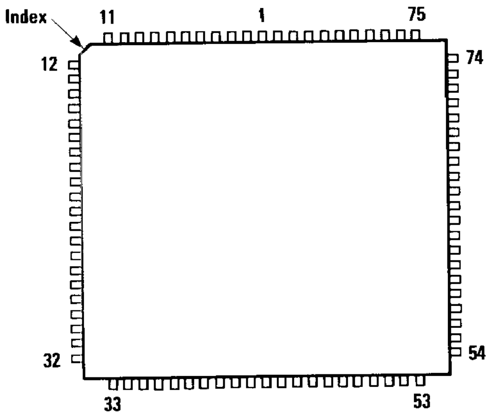
DC Characteristics

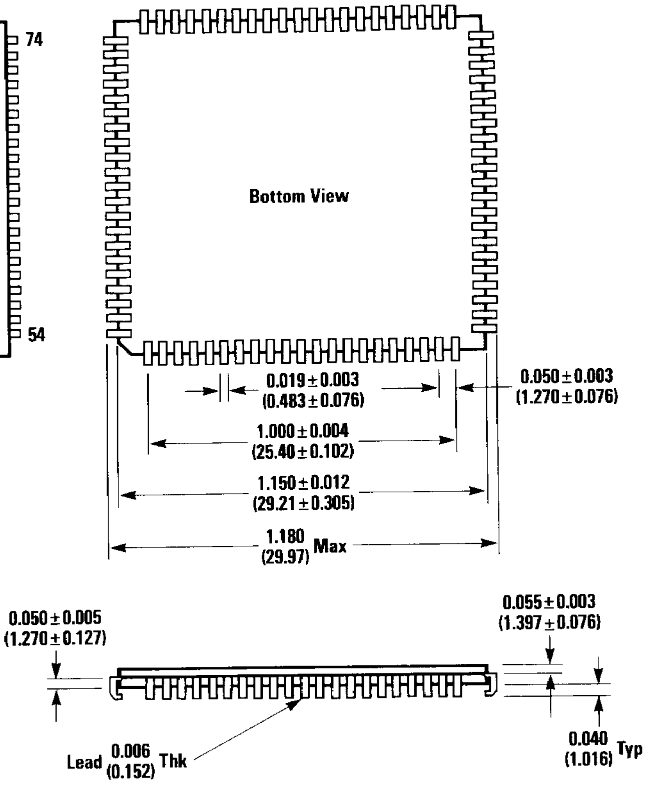
Parameter	Description	Test Conditions	12.5 MHz Min	12.5 MHz Max	16.67 MHz Min	16.67 MHz Max	Units
VOH	Output High Voltage	VCC=Min. IOH=−4 mA	3.5		3.5		V
VOL	Output Low Voltage	VCC=Min. IOL=4 mA		0.4		0.4	V
VIH	Input High Voltage		2.0	VCC+0.5	2.0	VCC+0.5	V
VIL	Input Low Voltage		−0.5[1]	0.8	−0.5[1]	0.8	V
VIHS	Input High Voltage		−2.5[2]	VCC+0.5	3.0[2]	VCC+0.5	V
VILS	Input Low Voltage		−0.5[1]	0.4	−0.5[1]	0.4	V
VIHC	Input High Voltage		4.0[3]	VCC+0.5	4.0[3]	VCC+0.5	V
VILC	Input Low Voltage		−0.5[1]	0.4	−0.5[1]	0.4	V
CIn	Input Capacitance		10		10		pF
COut	Output Capacitance		10		10		pF
ICC	Operating Current			500		550	mA

Notes:
1. VIL Min. = −3.0 V for pulse width less than 15 ns.
2. VIHS and VILS apply to Clk2xSys, Clk2xSmp, Clk2xRd, Clk2xPhi.
3. VIHC and VILC apply to Run* and Exception*

Packaging

84-Pin Quad Type "J" Package





LSI LOGIC

LR2010 Floating-Point Accelerator
Preliminary

Sales Offices and Design Resource Centers

LSI Logic Corporation Headquarters
Milpitas CA
■ **408.433.8000**

Arizona
602.951.4560

California
San Jose
408.248.5100

Irvine
■ 714.553.5600

Sherman Oaks
■ 818.906.0333

Colorado
303.756.8800

Florida
Altamonte Springs
407.339.2242

Boca Raton
■ 407.395.6200

Georgia
404.448.4898

Illinois
■ 312.773.0111

Maryland
■ 301.897.5800

Massachusetts
■ 617.890.0180 (Design Ctr)
617.890.0161 (Sales Ofc)

Michigan
313.930.6975

Minnesota
■ 612.921.8300

New Jersey
■ 201.549.4500

New York
914.226.1620

North Carolina
■ 919.872.8400

Ohio
614.438.2644

Oregon
503.644.6697

Pennsylvania
215.638.3010

Texas
Austin
512.338.2140

Dallas
■ 214.788.2966

Washington
■ 206.822.4384

Austria
LSI Logic/Steiner
■ 43.222.827474.0

LSI Logic Corporation of Canada, Inc.
Headquarters
Calgary
■ 403.262.9292

Edmonton
■ 403.450.4400

Ottawa
■ 613.592.1263

Montreal
■ 514.694.2417

Toronto
■ 416.622.0403

Vancouver
■ 604.433.5705

France
LSI Logic S.A.
■ 33.1.46212525

Israel
LSI Logic Limited
■ 972.3.5403741

Italy
LSI Logic SPA
■ 39.39.651575

Japan
LSI Logic K. K.
Tokyo
■ 81.3.589.2711

Tsukuba-Shi
■ 81.298.52.8371

Osaka
■ 81.6.947.5281

LSI Logic Corporation of Korea Limited
■ 82.2.785.1693

Netherlands
LSI Logic/Arcobel
■ 31.4120.30335

Scotland
LSI Logic Limited
■ 44.506.416767

Sweden
LSI Logic Limited
46.8.703.4680

Switzerland
LSI Logic/Sulzer
■ 41.32.515441

United Kingdom
LSI Logic Limited
■ 44.344.426544

West Germany
LSI Logic GmbH
Headquarters
Munich
■ 49.89.926903.0

Dusseldorf
■ 49.211.5961066

Stuttgart
■ 49.711.2262151

LSI Logic/EKB
Berlin
■ 49.30.311006.0

LSI Logic/Purfürst
Isernhagen
■ 49.511.6104.0

LSI Logic/TEP Elektronik Ingenieurtechnik
Luebeck
■ 49.451.893941

■ Sales Offices with Design Resource Centers

Printed in USA
108.83.DS032.10K.IM.CS

013866 X X