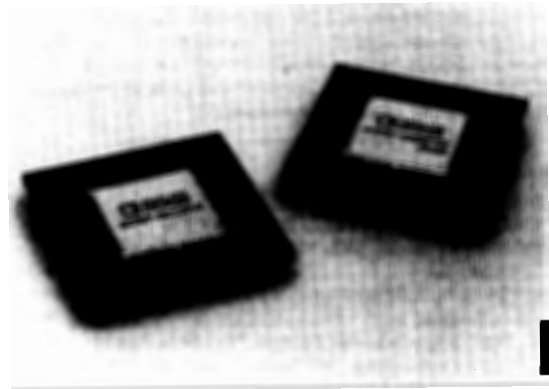
ANALOG DEVICES

64-Bit IEEE Floating-Point Chipset

ADSP-3212/ADSP-3222



FEATURES

- **Complete 40 MFLOPS Floating-Point Chipset**
  - **Multiplier/Divider and ALU**
  - **Fully Compatible with IEEE Standard 754**
- **Arithmetic Operations on Four Data Formats:**
  - **32-Bit Single-Precision Floating-Point**
  - **64-Bit Double-Precision Floating-Point**
  - **32-Bit Twos-Complement Fixed-Point**
  - **32-Bit Unsigned-Magnitude Fixed-Point**
- **Only One Internal Pipeline Stage**
- **20 MFLOPS Pipelined Throughput For Multiplication and Standard ALU Operations**
- **Exact Division: 300ns Single Precision and 600ns Double Precision**
- **Low Latency for Scalar Operations**
  - **130ns for 32-Bit Multiplication or Standard ALU Operations**
  - **155ns for 64-Bit Multiplication or Standard ALU Operations**
- **Exact Square Root ALU Instruction**
- **2.5W Maximum Power Dissipation per Chip with 1.0 μm CMOS Technology**
- **144-Lead Pin Grid Array**
- **Available Specified to MIL-STD-883, Class B**
- **Pin-Compatible Upgrades From ADSP-3211/ADSP-3221**

APPLICATIONS

- **High Performance Digital Signal Processing**
- **Engineering Workstations**
- **Floating-Point Accelerators**
- **Array Processors**
- **Mini-Supercomputers**
- **RISC Processors**

GENERAL DESCRIPTION

The ADSP-3212 Floating-Point Multiplier/Divider and the ADSP-3222 Floating-Point ALU are high speed, low power arithmetic processors conforming to IEEE Standard 754. The multiplier/divider and ALU comprise the basic computational elements for implementing a high speed numeric processor. Operations are supported on four data formats: 32-bit IEEE single-precision floating-point, 64-bit IEEE double-precision floating-point, 32-bit twos-complement fixed-point and 32-bit unsigned-magnitude fixed-point.

The high throughput of the ADSP-3212/ADSP-3222 is achieved with only a single level of internal pipelining, greatly simplifying program development. Theoretical MFLOPS rates are much easier to approach in actual systems with this chip architecture than with alternative, more heavily pipelined chipsets. Also, the minimal internal pipelining in the ADSP-3212/ADSP-3222 results in very low latency, important in scalar processing and in algorithms with data dependencies.

Both chips have internal feedback paths from the output to four of the eight input registers and feedforward paths from all input registers to the output register. Feedback to both banks of input registers facilitates interleaving partial sums and partial products for maximum throughput.

In conforming to IEEE Standard 754, these chips assure complete software portability for computational algorithms adhering to the Standard. All four rounding modes are supported for all floating-point data formats and conversions. Five IEEE exception conditions—overflow, underflow, invalid operation, inexact result and division-by-zero—are available externally on four status pins. The IEEE gradual underflow provisions are also supported, with special instructions for handling denormals. Alternatively, each chip offers a FAST mode which sets results less than the smallest IEEE normalized values to zero, thereby eliminating underflow exception handling when full conformance to the Standard is not essential.

IEEE floating-point division is supported by both the ADSP-3212 and the ADSP-3222. The ADSP-3212 is the faster of the two, performing single-precision division in six cycles and double-precision division in 12 cycles. The division operation is initiated by the assertion of the multiplier/divider's DIVMUL input. On the ADSP-3222 ALU two instructions, SDIV and DDIV, calculate single-precision division (16 cycles) and double-precision division (30 cycles), respectively. ADSP-3222 division instructions are supported for compatibility with the ADSP-3221.

The instruction set of the ADSP-3212/ADSP-3222, a superset of the ADSP-3210/3211/3220/3221 instruction set, is oriented to system-level implementations of function calculations. Specific instructions are included to facilitate such operations as floating-point division and square root, table lookup, quadrant normalization for trigonometric functions, extended-precision integer operations, logical operations and conversions between all data formats.

Both chips have two input ports and eight input registers (two banks of four registers) and are always in a two-input-port configuration; data can always be input on both ports simultaneously. In the 32-bit data loading mode, input can be directed to registers in either bank, although both ports may not input to the same bank at once. If 64-bit parallel data loading is enabled, 64-bit data from both ports may be directed to one of four register pairs.

In addition to double- and single-precision floating-point multiplication and division, the ADSP-3212 Floating-Point Multiplier/Divider supports 32-bit fixed-point multiplications: twos-complement, unsigned-magnitude and mixed-mode. The ADSP-3212 also has a $\overline{\text{HOLD}}$ control that prevents the updating of the output data and status registers.

TABLE OF CONTENTS

The instruction set of the ADSP-3222 Floating-Point ALU includes exact IEEE floating-point division and square-root operations. The ADSP-3222 is pin-compatible with the ADSP-3220/ADSP-3221. It also includes a $\overline{\text{HOLD}}$ control that is enabled through an overhead instruction.

The ADSP-3212/ADSP-3222 chipset is fabricated in double-metal 1.0μm CMOS. Each chip consumes 2.5W maximum, significantly less than comparable bipolar solutions. The differential between the chipset's junction temperature and the ambient temperature stays small because of this low power dissipation. Thus, the ADSP-3212/ADSP-3222 can be safely specified for operation at environmental temperatures over its extended temperature range (−55°C to +125°C ambient).

The ADSP-3212/ADSP-3222 are available for both commercial and extended temperature ranges. Extended temperature range parts are available processed fully to MIL-STD-883, Class B. The ADSP-3212 and ADSP-3222 are packaged in a ceramic 144-lead pin grid array.

FUNCTIONAL DESCRIPTION OVERVIEW

The ADSP-3212/ADSP-3222 share a common architecture (Figure 1) in which all input data is loaded to a set of input registers with both rising and falling clock edges. Two 32-bit operands can be loaded simultaneously; alternatively, the ADSP-3212/ADSP-3222 can operate in a mode in which both halves of a 64-bit operand are loaded in parallel. The input registers can be read to the chip's computational circuitry as they are loaded on a rising edge. At the end of first processing clock cycle, partial results and most controls are clocked into a set of internal pipeline registers. In most cases, only a second clock cycle is required to conclude processing. (The exceptions are division and square root.) At the end of this second processing cycle, results are clocked into an output register. The contents of the output register can then be driven off-chip. An output multiplexer allows driving both halves of a 64-bit double-precision result off-chip through the 32-bit output port in one output cycle.

Because all input and output data is internally registered and because of the single level of internal pipeline registers, operations can be overlapped for high levels of pipelined throughput.

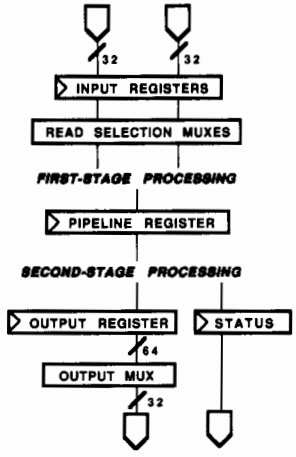


Figure 1. ADSP-32XX Generic Architecture

Figure 2 illustrates a typical sequence of pipelined operations. Note cycle #4 of Figure 2 after the data transfer and internal pipelines are full. While the final A results of the first operation are being driven off-chip, B processing can be concluding at the second stage, C processing beginning at the first stage and D data loading to the input registers.

time (cycles)	Load Input Data	First-Stage Processing	Second-Stage Processing	Output Result
1	Data Set A			
2	Data Set B	Data Set A		
3	Data Set C	Data Set B	Data Set A	
4	Data Set D	Data Set C	Data Set B	Data Set A
5	Data Set E	Data Set D	Data Set C	Data Set B

Figure 2. Typical Pipelining with the ADSP-3212/ADSP-3222

The ADSP-3212/ADSP-3222 can load data on rising edges of the clock and on falling edges of the clock, subject to constraints described in "Method of Operation." The ADSP-3212/ADSP-3222 can also operate in a mode in which all 64 bits of a double-precision word are loaded into an input register pair in parallel. This mode allows direct connection to a 64-bit input bus. All input registers have their own independent load selection controls, allowing the same data to be loaded to multiple registers simultaneously.

A set of read selection multiplexers feeds input data from the input registers to the computational circuitry. These multiplexers can select data that was just loaded at the clock's rising edge, if desired, with no throughput or cycle-time penalty.

All control signals need only be supplied to the chips at their cycle rate. This approach avoids requiring that the sequencing control cycle time be faster than the chipset's major processing cycle rate. Less expensive microcode memory can therefore be used. For this reason, load selection controls for registers to be loaded on the clock's falling edge need only be valid at the previous rising edge. (The designer may choose to supply the asynchronous port configuration, output multiplexer and tristate controls at a higher rate, however.)

The ADSP-3212/ADSP-3222 fully supports the gradual underflow provisions of IEEE Standard 754 for floating-point arithmetic. The Floating-Point ALU can operate directly on both normals and denormals (except division and square root). The Floating-Point Multiplier/Divider operates on normals but cannot operate on denormals directly. Denormals must first be "wrapped" (converted to a format acceptable for multiplication, division or square root) by an ALU. Several flags are available for detecting and handling exceptions caused by loading a denormal into a Floating-Point Multiplier/Divider. Information about rounding and inexact results generated by the multiplier/divider is needed by the ALU to produce results in conformance to Standard 754. Both ADSP-3212/ADSP-3222 chips include a "FAST" control that flushes all denormalized results to zero, avoiding the system delays of IEEE exception processing for gradual underflow.

All status output flags except denormal detection are registered at the output in parallel with their associated results. The asynchronous denormal flag allows an early detection of a denormalized number loaded to a Floating-Point Multiplier/Divider, speeding exception processing.

PIN DEFINITIONS AND FUNCTIONAL BLOCK DIAGRAMS

All control pins are active HI (positive true logic naming convention), except $\overline{\text{RESET}}$ and $\overline{\text{HOLD}}$. Some controls are registered at the clock's rising edge (REG); other controls are latched in clock HI and transparent in clock LO (LAT), and others are asynchronous (ASYN).

ADSP-3212 FLOATING-POINT MULTIPLIER/DIVIDER PIN LIST

Pin Name	Description	Type
DATA PINS		
AIN_{31-0}	32-Bit Data Input	
BIN_{31-0}	32-Bit Data Input	
$DOUT_{31-0}$	32-Bit Data Output	
CONTROL PINS		
$\overline{\text{RESET}}$	Reset	ASYN
$\overline{\text{HOLD}}$	Hold Control	LAT
IPORT	Input Port Configuration Control	ASYN
SELA0	Load Selection for A0	LAT
SELA1	Load Selection for A1	LAT
SELA2	Load Selection for A2	LAT
SELA3	Load Selection for A3	LAT
SELB0	Load Selection for B0	LAT
SELB1	Load Selection for B1	LAT
SELB2	Load Selection for B2	LAT
SELB3	Load Selection for B3	LAT
RDA0	Register Ax Read Selection Control 0	REG
RDA1	Register Ax Read Selection Control 1	REG
RDB0	Register Bx Read Selection Control 0	REG
RDB1	Register Bx Read Selection Control 1	REG
WRAPA	Wrapped Contents in Register Ax	REG
WRAPB	Wrapped Contents in Register Bx	REG
TCA	Twos-Complement Integer in Register Ax	REG
TCB	Twos-Complement Integer in Register Bx	REG
ABSA	Read Absolute Value of Ax	REG
ABSB	Read Absolute Value of Bx	REG
SP	Single-Precision Mode	REG
DP	Double-Precision Mode	REG
RND0	Rounding Mode Control 0	REG
RND1	Rounding Mode Control 1	REG
FAST	Fast Mode	REG
SHLP	Shift Left Fixed-Point Product	REG
FDBK0	Feedback Control 0	REG
FDBK1	Feedback Control 1	REG
LOAD64	Enable 64-Bit Parallel Input	REG
DIVMUL	Divide/Multiply	REG
MSWSEL	Select MSW of Output Register	ASYN
OEN	Output Data Enable	ASYN

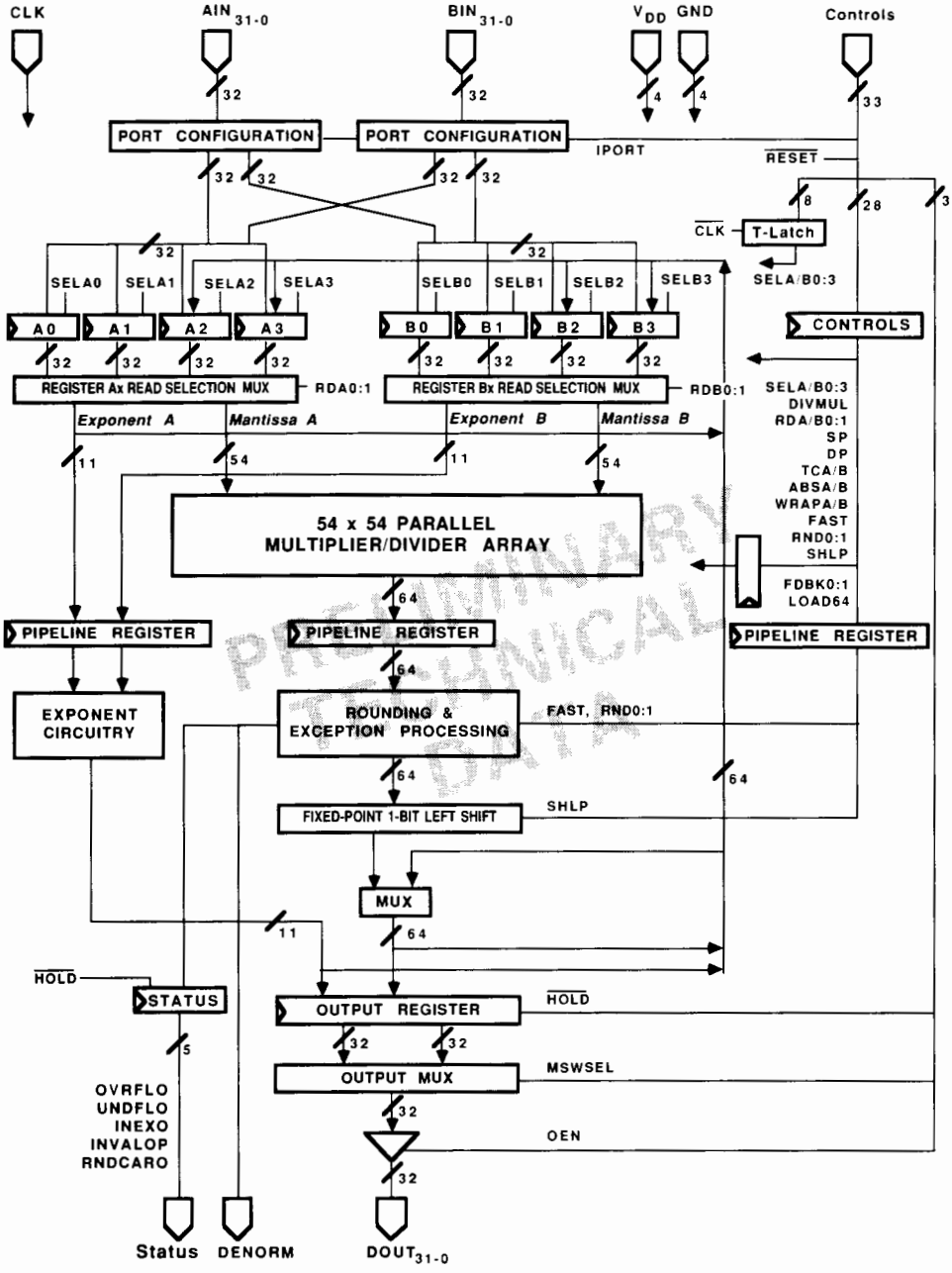


Figure 3. ADSP-3212 Block Diagram

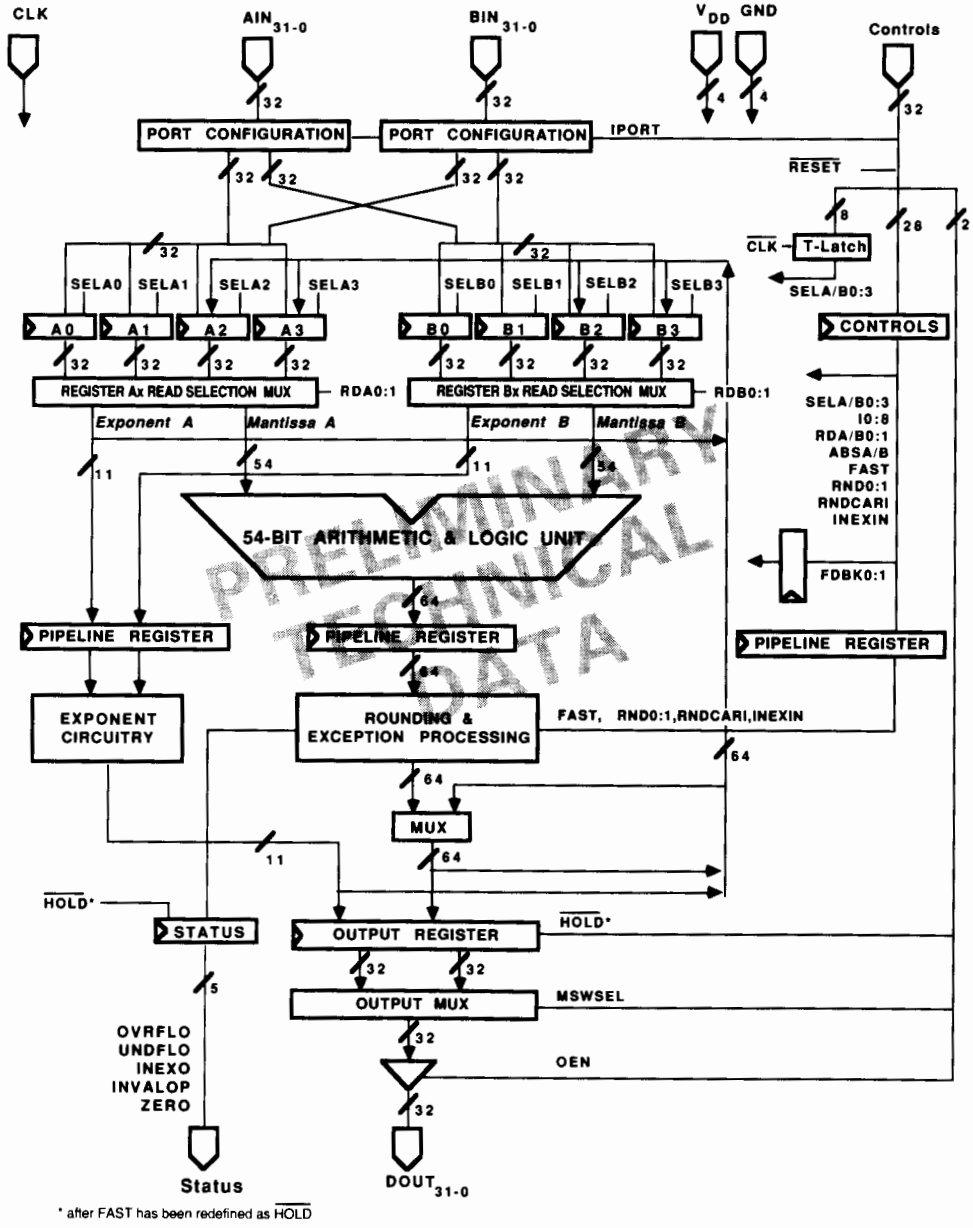


Figure 4. ADSP-3222 Block Diagram

Pin Name	Description	Type
STATUS OUT		
INEXO	Inexact Result	
OVRFLO	Overflowed Result	
UNDFLO	Underflowed Result	
INVALOP	Invalid Operation	
DENORM	Denormal Output	
RNDCARO	Round Carry Propagation Out	
MISCELLANEOUS		
CLK	Clock Input	
V_{DD}	+5V Power Supply (four lines)	
GND	Ground Supply (four lines)	

ADSP-3222 FLOATING-POINT ALU PIN LIST

Pin Name	Description	Type
DATA PINS		
AIN_{31-0}	32-Bit Data Input	
BIN_{31-0}	32-Bit Data Input	
$DOUT_{31-0}$	32-Bit Data Output	
CONTROL PINS		
$\overline{RESET}$	Reset	ASYN
IPORT	Input Port Configuration Control	ASYN
SELA0	Load Selection for A0	LAT
SELA1	Load Selection for A1	LAT
SELA2	Load Selection for A2	LAT
SELA3	Load Selection for A3	LAT
SELB0	Load Selection for B0	LAT
SELB1	Load Selection for B1	LAT
SELB2	Load Selection for B2	LAT
SELB3	Load Selection for B3	LAT
RDA0	Register Ax Read Selection Control 0	REG
RDA1	Register Ax Read Selection Control 1	REG
RDB0	Register Bx Read Selection Control 0	REG
RDB1	Register Bx Read Selection Control 1	REG
ABSA	Read Absolute Value of Ax	REG
ABSB	Read Absolute Value of Bx	REG
I_{8-0}	ALU Instruction	REG
RND0	Rounding Mode Control 0	REG
RND1	Rounding Mode Control 1	REG
FAST	Fast Mode/$\overline{HOLD}$ Control	REG
FDBK0	Feedback Control 0	REG
FDBK1	Feedback Control 1	REG
MSWSEL	Select MSW of Output Register	ASYN
OEN	Output Data Enable	ASYN
STATUS IN		
INEXIN	Inexact Data In	REG
RNDCARI	Round Carry Propagation In	REG
STATUS OUT		
INEXO	Inexact Result	
OVRFLO	Overflowed Result	
UNDFLO	Underflowed Result	
INVALOP	Invalid Operation	
ZERO	Zero Result	
MISCELLANEOUS		
CLK	Clock Input	
V_{DD}	+5V Power Supply (four lines)	
GND	Ground Supply (four lines)	

METHOD OF OPERATION

Data Formats

The ADSP-3212/ADSP-3222 chipset supports both single- and double-precision floating-point data formats and operations as defined in IEEE Standard 754-1985. Both chips support 32-bit twos-complement fixed-point as well as 32-bit unsigned-magnitude data formats and operations (the ADSP-3212 supports fixed-point multiplication but not fixed-point division). Both chips operate directly on 32-bit fixed-point data. No time consuming conversions to and from floating-point formats are required.

IEEE Single-Precision Floating-Point Data Format

IEEE Standard 754 specifies a 32-bit single-precision floating-point format, which consists of a sign bit *s*, a 24-bit significand and an 8-bit unsigned-magnitude exponent *e*. For normalized numbers, this significand consists of a 23-bit fraction *f* and a "hidden" bit of 1 that is implicitly presumed to precede f_{22} in the significand. The binary point is presumed to lie between this hidden bit and f_{22}. The least significant bit of the fraction is f_0; the LSB of the exponent is e_0. The hidden bit effectively increases the precision of the floating-point significand to 24 bits from the 23 bits actually stored in the data format. It also insures that the significand of any number in the IEEE normalized-number format is always greater than or equal to 1 and less than 2.

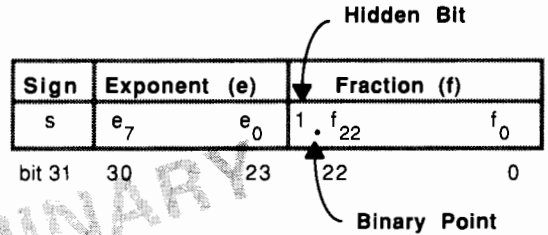


Figure 5. IEEE Single-Precision Floating-Point Format

The unsigned exponent *e* for normals can range between $1 \le e \le 254$ in the single-precision format. This exponent is *biased* by +127 (254÷2) in the single-precision format. This means that to calculate the "true" *unbiased* exponent, 127 must be subtracted from *e*.

The IEEE Standard also provides for several special data types. In the single-precision floating-point format, an exponent value of 255 (all ones) with a non-zero fraction is a not-a-number (NAN). NANs are usually used as flags for data flow control, for the values of uninitialized variables and for the results of invalid operations such as $0 \cdot \infty$. Infinity is represented as an exponent of 255 and a zero fraction. Note that because the fraction is signed, both positive and negative INF can be represented.

The IEEE Standard requires the support of denormalized data formats and operations. A denormalized number, or "denormal," is a number with a magnitude less than the minimum normalized ("normal") number in the IEEE format. Denormals have a zero exponent and a non-zero fraction. Denormals have no hidden "one" bit. (Equivalently, the hidden bit of a denormal is zero.) The unbiased (true) value of a denormal's exponent is −126 in the single-precision format, i.e., one minus the exponent bias. Note that because denormals are not required to have a significant leading one bit, the precision of a denormal's significand can be as little as one bit for the minimum representable denormal.

ZERO is represented by a zero exponent and a zero fraction. As with INF, both positive ZERO and negative ZERO can be represented.

The IEEE single-precision floating-point data types and their interpretations are summarized:

Mnemonic	Exponent	Fraction	Value	Name	IEEE Format?
NAN	255	non-zero	undefined	not-a-number	yes
INF	255	zero	$(-1)^s$(infinity)	infinity	yes
NORM	1 thru 254	any	$(-1)^s(1.f)2^{e-127}$	normal	yes
DNRM	0	non-zero	$(-1)^s(0.f)2^{-126}$	denormal	yes
ZERO	0	zero	$(-1)^s0.0$	zero	yes
WRAP	−22 thru 0	any	$(-1)^s(1.f)2^{e-127}$	wrapped	no
UNRM	−171 thru −23	any	$(-1)^s(1.f)2^{e-127}$	unnormal	no

Table I. IEEE Single-Precision Floating-Point Data Types and Interpretations

The ADSP-3212/ADSP-3222 chipset also supports two data types not included in the IEEE Standard, "wrapped" and "unnormal." These data types are necessitated by the fact that the ADSP-3222 ALU (during division and square root) and the ADSP-3212 Multiplier/Divider do not operate directly on denormals. (To do so, they would need shifting hardware that would slow them significantly.) Denormal operands must first be translated by the ADSP-3222 ALU to wrapped numbers to be readable by the ADSP-3212 Multiplier/Divider. Wrapped and unnormal multiplier/divider results must also be unwrapped by the ADSP-3222 before an ALU can operate on these results in general. (See "Gradual Underflow and IEEE Exceptions.")

The interpretation of wrapped numbers differs from normals only in that the exponent is treated as a twos-complement number. Single-precision wrapped numbers have a hidden bit of one and an exponent bias of +127. All single-precision denormals can be mapped into wrapped numbers where the exponent e ranges between $-22 \le e \le 0$. WRAPA and WRAPB controls on the ADSP-3212 tell the multiplier/divider to interpret a data value as a wrapped number.

The ranges of the various single-precision IEEE floating-point data formats supported by the ADSP-3212/ADSP-3222 are summarized in Table II.

The multiplication of a wrapped number by a normal number or another wrapped number can produce a number smaller than can be represented as a wrapped number. Such numbers are called "unnormals." Unnormals are interpreted exactly as are wrapped numbers. They differ only in the range of their exponents, which is $-171 \le e \le -23$ for single-precision unnormals. The smallest unnormal is the result of multiplying WRAP.MIN by itself. Unnormals, because they are smaller than DRNM.MIN, generally unwrap to ZERO. (Unnormals can unwrap to DRNM.MIN, depending on the rounding mode.)

The underflow flag should be thought of as an implicit most significant ninth bit, the sign bit. For unnormals for which $-171 \le e < -128$, the most significant bit in the eight-bit exponent field (e_7, Bit 30) will be zero, but the underflow flag understood as weighted by −256 allows their representation without ambiguity. This sign bit is implicitly assumed by the ALU to be present when unwrapping unnormals, making this convention for very small unnormals transparent to the user.

Data name (positive)	Exponent	Exp. data type	Exponent bias	Hidden bit	Fraction (binary)	Unbiased absolute value
NORM.MAX	254	unsigned	+127	1	111.....11	$2^{+127}\cdot(2-2^{-23})$
NORM.MIN	1	unsigned	+127	1	000.....00	2^{-126}
DNRM.MAX	0	unsigned	+126	0	111.....11	$2^{-126}\cdot(1-2^{-23})$
DNRM.MIN	0	unsigned	+126	0	000.....01	$2^{-126}\cdot 2^{-23}$
WRAP.MAX	0	2scmplmt	+127	1	111.....11	$2^{-127}\cdot(2-2^{-23})$
WRAP.MIN	−22	2scmplmt	+127	1	000.....00	2^{-149}
UNRM.MAX	−23	2scmplmt	+127	1	111.....11	$2^{-150}\cdot(2-2^{-23})$
UNRM.MIN	−171	2scmplmt	+127	1	000.....00	2^{-298}

Table II. IEEE Single-Precision Floating-Point Range Limits

IEEE Double-Precision Floating-Point Data Format

IEEE Standard 754 specifies a 64-bit double-precision floating-point format:

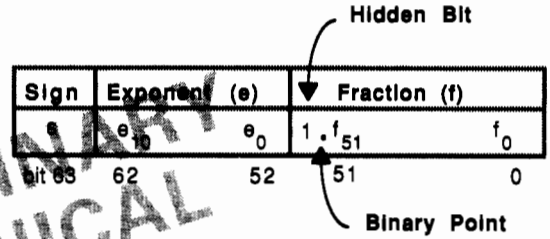


Figure 6. IEEE Double-Precision Floating-Point Format

The key differences with the single-precision format are that the exponent e is now 11 bits in length and the fraction f is now 52 bits in length, yielding a 53-bit significand for double-precision normals. Double-precision, like single-precision, has an implicit hidden bit; in this case the hidden bit precedes f_{51}. The binary point comes between the hidden bit and f_{51}. The exponent bias for double-precision floating-point normals is +1023 (2046÷2).

In other respects, IEEE double-precision floating-point is exactly analogous to single-precision, with the same data types whose values are summarized in Table III.

The unbiased value of a denormal's exponent is −1022 for double-precision denormals, i.e., one minus the bias. Because of the extended width of the double-precision fraction, the exponent of double-precision wrapped numbers can range from $-51 \le e \le 0$. The exponent of unnormals can range from $-1125 \le e \le -52$. Again, the smallest unnormal is the result of multiplying the smallest wrapped number by itself. Note that $e = -1024$ is the smallest double-precision exponent that is directly representable in the eleven-bit IEEE twos-complement exponent field. The underflow flag should be thought of as a most significant twelfth bit, the sign bit, as explained above for single-precision unnormals.

Mnemonic	Exponent	Fraction	Value	Name	IEEE Format?
NAN	2047	non-zero	undefined	not-a-number	yes
INF	2047	zero	$(-1)^s$(infinity)	infinity	yes
NORM	1 thru 2046	any	$(-1)^s(1.f)2^{e-1023}$	normal	yes
DNRM	0	non-zero	$(-1)^s(0.f)2^{-1022}$	denormal	yes
ZERO	0	zero	$(-1)^s0.0$	zero	yes
WRAP	−51 thru 0	any	$(-1)^s(1.f)2^{e-1023}$	wrapped	no
UNRM	−1125 thru −52	any	$(-1)^s(1.f)2^{e-1023}$	unnormal	no

Table III. IEEE Double-Precision Floating-Point Data Types and Interpretations

The ranges for the various double-precision data types are:

Data name (positive)	Exponent	Exp. data type	Exponent bias	Hidden bit	Fraction (binary)	Unbiased absolute value
NORM.MAX	2046	unsigned	+1023	1	111.....11	$2^{+1023}\cdot(2-2^{-52})$
NORM.MIN	1	unsigned	+1023	1	000.....00	2^{-1022}
DNRM.MAX	0	unsigned	+1022	0	111.....11	$2^{-1022}\cdot(1-2^{-52})$
DNRM.MIN	0	unsigned	+1022	0	000.....01	$2^{-1022}\cdot 2^{-52}$
WRAP.MAX	0	2scmplmt	+1023	1	111.....11	$2^{-1023}\cdot(2-2^{-52})$
WRAP.MIN	−51	2scmplmt	+1023	1	000.....00	2^{-1074}
UNRM.MAX	−52	2scmplmt	+1023	1	111.....11	$2^{-1075}\cdot(2-2^{-52})$
UNRM.MIN	−1125	2scmplmt	+1023	1	000.....00	2^{-2148}

Table IV. IEEE Double-Precision Floating-Point Range Limits

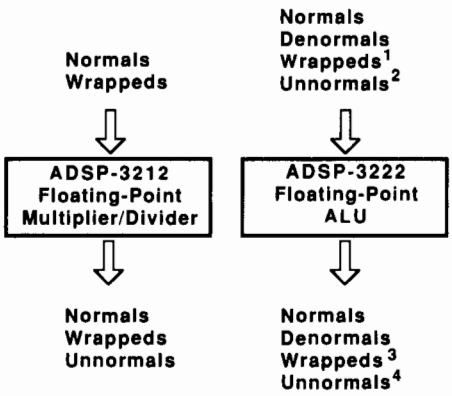
Supported Floating-Point Data Types

The floating-point data types supported directly by the ADSP-3212/ADSP-3222 chipset are summarized in Figure 7.

Not all the data types described above are supported directly. See the section below, "Gradual Underflow and IEEE Exceptions" for a full description of how the chips work together to implement the IEEE Standard. For systems not requiring full conformance to Standard 754, the section below, "FAST/IEEE Control," describes a simplified operation for this chipset that avoids denormals, wrappeds, and unnormals altogether.

32-Bit Fixed-Point Data Formats

The ADSP-3212/ADSP-3222 chipset supports two 32-bit fixed-point formats: twos-complement and unsigned-magnitude. With the ALU, the output data format is identical with the input data format, i.e., 32 bits wide. In contrast, the multiplier/divider produces a 64-bit product from two 32-bit inputs. Fixed-point division and fixed-point square root are not supported directly. However, they can be accomplished using the fixed-point/floating-point conversions.



1. for unwrapping, division, and square root
2. for unwrapping only
3. from wrapping and division
4. from division

Figure 7. Data Types Directly Supported by the ADSP-3212/ADSP-3222

The 32-bit twos-complement data format for ALU inputs and outputs and multiplication inputs is:

WEIGHT	Sign -2^{k+31}	2^{k+30}	2^{k+29}	...	2^{k}
VALUE	i_{31}	i_{30}	i_{29}	...	i_0
POSITION	31	30	29	...	0

Figure 8. 32-Bit Twos-Complement Fixed-Point Data Format

The MSB is i_{31}, which is also the sign bit; the LSB is i_0. Note that the sign bit is negatively weighted in twos-complement format. The position of the binary point for fixed-point data is represented here in full generality by the integer k. Integers (binary point to right of bit position 0) are represented when k=0; signed fractional numbers (binary point between bit positions 31 and 30) are represented when k = −31. The value of k is for user interpretation only and in general does not affect the operation of the chips. The only exceptions are the ALU conversion operations between floating-point and fixed-point. For these operations, the fixed-point format is presumed to be twos-complement integers, i.e., k=0.

The ADSP-3212 Multiplier/Divider can produce a 64-bit product at its output register. The ADSP-3212 will produce results in the format of Figure 9 at the DOUT port if the Shift Left Fixed-Point Product (SHLP) control (described below in "Output Control") is LO:

WEIGHT	Sign -2^{r+63}	2^{r+62}	...	2^{r+32}	2^{r+31}	...	2^{r+1}	2^{r}
VALUE	i_{63}	i_{62}	...	i_{32}	i_{31}	...	i_1	i_0
POSITION	63	62	...	32	31	...	1	0

Most Significant Product — Least Significant Product

Figure 9. 64-Bit Twos-Complement Fixed-Point Data Format at Multiplier/Divider Output Register with SHLP LO

The weighting of the product bits is given by the integer r. When k_A represents the weighting of operand A and k_B the weighting of operand B, then $r = k_A + k_B$.

When HI, the SHLP control shifts all bits left one position as they are loaded to the output register. The results will then be in the format:

WEIGHT	Sign -2^{r+62}	2^{r+61}	...	2^{r+31}	2^{r+30}	...	2^{r}	2^{r-1}
VALUE	i_{62}	i_{61}	...	i_{31}	i_{30}	...	i_0	0
POSITION	63	62	...	32	31	...	1	0

Most Significant Product — Least Significant Product

Figure 10. 64-Bit Twos-Complement Fixed-Point Data Format at Multiplier/Divider Output Register with SHLP HI

The LSB becomes zero and i_{62} moves into the sign bit position. Normally i_{63} and i_{62} will be identical in twos-complement products. (The only exception is full-scale negative multiplied by itself.) Hence, a one-bit left-shift normally removes a redundant sign bit, thereby increasing the precision of the most significant product. Also, if the fixed-point data format is fractional ($k = -31$ in Figure 8), then a single-bit left-shift will renormalize the MSP to a fractional format (because $r = 2 \cdot k = 2 \cdot (-31) = -62$).

For unsigned-magnitude data formats, inputs to the ADSP-3212 Multiplier/Divider and inputs and outputs for the ADSP-3222 ALU will be 32-bits wide. The 32-bit unsigned-magnitude data format is:

WEIGHT	2^{k+31}	2^{k+30}	2^{k+29}	...	2^{k}
VALUE	I_{31}	I_{30}	I_{29}	...	I_{0}
POSITION	31	30	29	...	0

Figure 11. 32-Bit Unsigned-Magnitude Fixed-Point Data Format

Again, the position of the binary point for fixed-point data is represented here in full generality by the integer *k*. Integers (binary point to the right of bit position 0) are represented when $k=0$; unsigned fractional numbers (binary point left of bit position 31) are represented when $k = -32$. The value of k is for user interpretation only and, except for conversions to fixed-point, does not affect the operation of the chips.

The ADSP-3212 Multiplier/Divider discriminates twos-complement from unsigned-magnitude inputs with TCA and TCB controls (see "Controls"). When TCA and TCB are both LO, the ADSP-3212 produces a 64-bit unsigned-magnitude product at its output register. The ADSP-3212 will produce results in this format if SHLP is LO:

WEIGHT	2^{r+63}	2^{r+62}	...	2^{r+32}	2^{r+31}	...	2^{r+1}	2^{r}
VALUE	I_{63}	I_{62}	...	I_{32}	I_{31}	...	I_{1}	I_{0}
POSITION	63	62	...	32	31	...	1	0

Most Significant Product — Least Significant Product

Figure 12. 64-Bit Unsigned-Magnitude Fixed-Point Data Format at Multiplier/Divider Output Register with SHLP LO

Again, the weighting of the product bits is given by the integer *r*. When k_A represents the weighting of operand A and k_B the weighting of operand B, then $r = k_A + k_B$.

If SHLP is HI, the data at the output register will have been shifted left one position and zero-filled in the format shown in Figure 13.

WEIGHT	2^{r+62}	2^{r+61}	...	2^{r+31}	2^{r+30}	...	2^{r}	2^{r-1}
VALUE	I_{62}	I_{61}	...	I_{31}	I_{30}	...	I_{0}	0
POSITION	63	62	...	32	31	...	1	0

Most Significant Product — Least Significant Product

Figure 13. 64-Bit Unsigned-Magnitude Fixed-Point Data Format at Multiplier/Divider Output Register with SHLP HI

The ADSP-3212 also supports mixed-mode multiplications, i.e., twos-complement by unsigned-magnitude. These are valuable in extended-precision fixed-point multiplications, e.g., 64×64 and 128×128. The result of a mixed-mode multiplication will be in a twos-complement format. Unlike twos-complement multiplications, however, mixed-mode results do not in general have a redundant sign bit in i_{62}. Hence, mixed-mode results should be read out with SHLP LO as in Figure 9.

Controls

The controls for the ADSP-3212/ADSP-3222 (see Pin Definitions above) are all active HI, with the exceptions of $\overline{\text{RESET}}$ and $\overline{\text{HOLD}}$. The controls are either *registered* into the input control register at the clock's rising edge, *latched* into the input control register with clock HI and transparent in clock LO, or *asynchronous*. The controls are discussed below in the order in which they affect data flowing through the chipset.

Registered controls, in general, are pipelined to match the flow of data. All data and control pipelines advance with the rising edge of each clock cycle. For example, to perform an optional fixed-point one-bit left-shift on output with the product of X and Y, you would assert the registered, pipelined control SHLP on the rising edge that causes X and Y inputs to be read into the multiplier array. Just before the result was ready to be loaded to the output register, the pipelined SHLP control would perform the proper shift. After the initiation of a multi-cycle operation, registered control inputs are ignored until the end of the operation time. (See "Timing" below for a precise definition of "operation time.")

Because this chipset uses CMOS static logic throughout and controls are pipelined, the clock can be stopped as long as desired for generating wait-states, diagnostic analysis, or whatever. These chips can also be easily adapted to "state-push" implementations. The machine's state can be pushed forward one stage by simply providing a rising edge to the clock input when desired.

The only controls that are latched (as opposed to registered) are the Load Selection Controls. They are transparent in clock LO and latched with clock HI. Load selection controls are set up to the chips exactly as if they were registered, with the same setup time. The fact that they are transparent in clock LO allows them to select input registers in parallel with the setup of data to be loaded on the rising edge. Because they are latched with clock HI, microcode need only be presented at the clock rate, though data is loaded on both clock rising and falling edges.

A few controls are asynchronous. These controls take effect immediately and are thus neither registered nor pipelined. Each has an independently specified setup time.

FAST/IEEE Control (REG)
FAST is a pipelined, registered control. It affects the interpretation of data read into processing circuitry immediately after having been loaded to the input control register. FAST affects the format of results in the rounding and exception processing pipeline stage. FAST also affects the definition of some exception flags (see "Status Flags").

IEEE Standard 754 requires a system to perform operations on denormal operands (which are smaller in magnitude than the minimum representable normalized number). This capability to accommodate these numbers is known as "gradual underflow." For floating-point systems not requiring strict adherence to the IEEE Standard, the ADSP-3212/ADSP-3222 provides a FAST mode (FAST control pin HI) which consistently flushes post-rounded results less than NORM.MIN to ZERO. This approach greatly simplifies exception processing and avoids generating the denormal, wrapped, and unnormal data types described above. When in FAST mode, the multiplier/divider will treat denormal inputs as ZERO. The ALU will treat denormal inputs exactly as it does in IEEE mode but still flush post-rounded results less than NORM.MIN to ZERO.

Systems implementing gradual underflow with the ADSP-3212/ADSP-3222 must treat the multiplication of operands that include a denormal as an exception to normal process flow. FAST should be LO on all chips. See the section below, "Gradual Underflow and IEEE Exceptions," for a fuller discussion of the details of implementing an IEEE system with this chipset.

On the ADSP-3222, the FAST input can be redefined after reset as a $\overline{\text{HOLD}}$ input through the HOLDEN instruction (see Table XIII). The mode (IEEE or FAST) that is active when the instruction is executed remains in effect. To restore the FAST function, you must reset the ADSP-3222.

$\overline{\text{RESET}}$ Control (ASYN)
The asynchronous, active LO $\overline{\text{RESET}}$ control clears all control functions in the ADSP-3212/ADSP-3222. $\overline{\text{RESET}}$ should be asserted on power up to insure proper initialization. ($\overline{\text{RESET}}$ will abort any multicycle operation in progress.) Status flags are cleared by $\overline{\text{RESET}}$. No input register contents are affected by $\overline{\text{RESET}}$; however, the output register can be invalidated if $\overline{\text{RESET}}$ is asserted LO during a multicycle operation. All load selection controls (SELA/B) must be LO at $\overline{\text{RESET}}$.

On reset, the ADSP-3222 is set for 32-bit input data loads and for the IEEE/FAST function on its FAST input.

Port Configuration – IPORT Control (ASYN)
This chipset offers two options on input port configuration. Both configurations in Figure 14 are "two-port" configurations. The options are controlled by the asynchronous input IPORT, which allows you to switch the port configuration dynamically—as often as every half cycle. Take care that IPORT is at valid logic levels at all clock edges at which data is loaded. IPORT must meet setup and hold times at every point at which data is loaded—a rising edge or falling edge of the clock. Proper data loading cannot be guaranteed unless this requirement is observed.

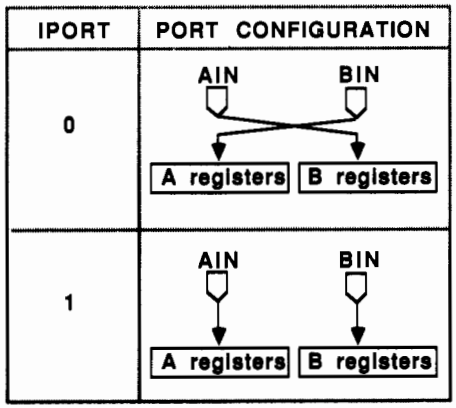


Figure 14. ADSP-3212/ADSP-3222 Input Port Configurations

Input Register Loading and Operand Storage – SELA/B Controls (LAT)
The chipset's 32-bit input registers are selected for data loading with the latched load selection controls, SELA0:3 and SELB0:3. Since each input register has its own control, the load selection controls are independent of one another. Multiple registers can be selected for parallel loads of the same input data, if desired. The load selection controls' effects on data loading are summarized in Figure 15.

SEL control	register loaded
SELA0	A0
SELA1	A1
SELA2	A2
SELA3	A3
SELB0	B0
SELB1	B1
SELB2	B2
SELB3	B3

Figure 15. ADSP-3212/ADSP-3222 Load Selection Controls

For 32-bit data loading, even-numbered registers load data on rising edges and odd-numbered registers on falling edges, as shown in Figure 16. However, you should not load a 32-bit register on the clock's falling edge in the same cycle that the register is read into the chip's processing circuits (see "Restrictions on Register Storage," below).

In the 64-bit parallel loading mode, inputs from both ports can be directed to appropriate register pairs (see "Restrictions on Register Storage," below). This mode is enabled by the LOAD64 pin on the ADSP-3212; on the ADSP-3222, the LOAD64 instruction sets 64-bit loading until the next reset. If

A registers are selected, they are loaded on the rising edge of the clock; B registers are loaded on the falling clock edge. LOAD64 is pipelined for one cycle on both parts; that is, the 64-bit load option will become effective at the *next* rising edge after the pin is asserted or the instruction is presented.

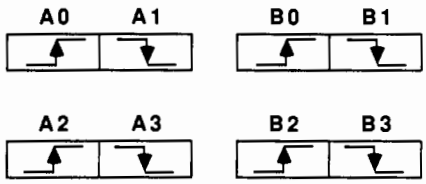


Figure 16. ADSP-3212/ADSP-3222 Clock Edge for 32-Bit Data Loading

Restrictions on Register Storage

For single-precision and fixed-point data, any convenient register can be used. The only restriction is that the register being loaded is not currently in use by the chip's processing elements. For all multiplications and most ALU operations, input registers are only read into the computational circuits for one cycle. Do not load a register for 32-bit operations on the clock's falling edge when that register has been selected to feed the chip's processing circuits in that same cycle (with the RDA/B controls described in "Input Data Register Read Selection"). Pick a register not in use.

The ADSP-3222 ALU is capable of two multicycle operations: IEEE floating-point division (16 cycles for single precision and 30 cycles for double precision) and square root (29 cycles for single precision and 58 cycles for double precision). The ADSP-3212 Multiplier/Divider performs multicycle (6 and 12 cycles for single precision and double precision, respectively) floating-point division. For single-precision floating-point division, the dividend can be stored in any A register and the divisor can be stored in any B register. Single-precision operands for IEEE square root can be stored in any B register. The registers selected to the computational circuits for these operations must be stable until the end of the operation time, whether single-precision or double-precision. (See "Timing" and the timing diagrams below for a precise definition of "operation time.")

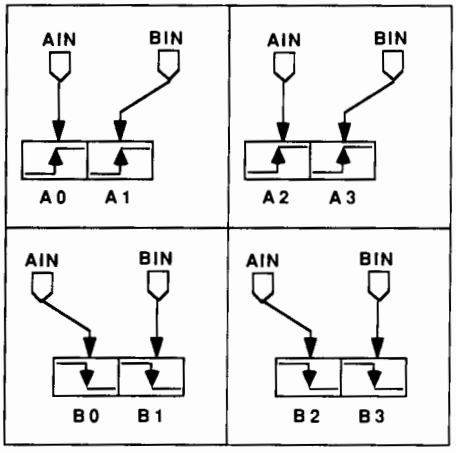


Figure 17. ADSP-3212/ADSP-3222 64-Bit Parallel Load

With 64-bit double-precision data, there are constraints on which registers hold which 32-bit halves of operands. You must load 64-bit data in adjacent pairs of 32-bit registers as shown in Figure 18. The 32-bit most significant word (MSW) will be in one even register and the 32-bit least significant word (LSW) in its odd neighbor.

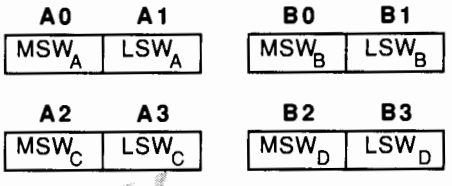


Figure 18. ADSP-3212/ADSP-3222 Operand Storage for Double-Precision Operations

Restrictions on Register Stability

With 64-bit data—as with 32-bit data—registers should not be loaded that are currently in use by the processing elements (i.e., selected by the RDA/B controls). Half the 32-bit registers in any pair of 64-bit operands will loaded on the falling edge with both members of this chipset.

To operate the ALU at full throughput in single-cycle double-precision operations, 64-bit register sets should be alternated every cycle. For example, A_0/A_1 and B_2/B_3 could be loaded with new operands while A_2/A_3 and B_0/B_1 were feeding the computational circuits (and were not changing). In this way, data loading will not disturb the contents of registers in use.

The ADSP-3222 ALU includes two double-precision multicycle operations in its instruction set: IEEE division (30 cycles) and square root (58 cycles). The ADSP-3212 performs a 12-cycle double-precision floating-point division. For double-precision floating-point division, the 64-bit dividend can be stored in either pair of A registers consistent with Figure 18. The divisor can be stored in either pair of B registers, also consistent with Figure 18. Double-precision operands for IEEE square root can be stored in either pair of B registers consistent with Figure 18. Registers containing operands in use must remain unchanged until the end of the operation time.

Data Format Selection – SP and DP Controls (REG)

The three data formats processed by the ADSP-3212/ADSP-3222 chipset are *single-precision* floating-point, *double-precision* floating-point, and *fixed-point*. With the ADSP-3212 Multiplier/Divider, the data format is indicated explicitly by the states of the DP and the SP registered controls:

SP	DP	Data Format Selection
0	0	fixed
0	1	double-precision
1	0	single-precision
1	1	illegal mode

Figure 19. ADSP-3212 Multiplier/Divider Data Format Selection

The state of the SP and DP controls at the rising edge when data is read into the multiplier array determines whether the data is interpreted as single-precision floating-point, double-precision floating-point, or fixed-point. Fixed-point division is not supported; fixed-point numbers should be converted to floating-point format for division and the result converted back to fixed-point format. Division is a multicycle operation (6 cycles for single precision and 12 cycles for double-precision); once initiated, the states of SP and DP don't matter until the next data is read to the processing circuitry.

For the ADSP-3222 ALU, data format selection is implicit in the ALU instruction, I_{8-0}. (See "Instructions and Operations" section below.)

Input Data Register Read Selection – RDA/B Controls (REG)
The register read selection controls, RDA0:1 and RDB0:1, are registered controls which select the input registers that are read into the chipset's processing circuitry. Any pair of input registers can be read into the processing circuitry. (For single-operand operations, the state of the selection controls for the unused register bank doesn't matter.) Data loaded to an input register on a rising edge can be read into the processing circuitry on that same edge.

The data format selected affects the interpretation of the RDA/B controls as follows:

RDA1	RDA0	SP & Fixed: A register selected	DP: A registers selected
0	0	A2	illegal state
0	1	A3	A2, A3
1	0	A0	illegal state
1	1	A1	A0, A1

RDB1	RDB0	SP & Fixed: B register selected	DP: B registers selected
0	0	B2	illegal state
0	1	B3	B2, B3
1	0	B0	illegal state
1	1	B1	B0, B1

Figure 20. ADSP-3212/ADSP-3222 Input Register Read Selection

Note that when feedforward is activated, the definitions of the RDA/B controls change. See "Feedback and Feedforward," below.

After the initiation of multicycle operations, the RDA/B controls are ignored.

Feedback and Feedforward – FDBK Controls (REG)
The ADSP-3212/ADSP-3222 have feedback paths to A_2, A_3, B_2, and B_3 and feedforward paths from all registers to the output register. The feedback controls FDBK0:1 determine whether the device is in normal operation, whether the feedback data goes to the A registers or the B registers, and whether feedforward is activated, as shown below:

FDBK1	FDBK0	Interpretation
0	0	normal operation
0	1	feedforward
1	0	feedback to A reg
1	1	feedback to B reg

Figure 21. Feedback and Feedforward Controls

These controls are pipelined for one cycle; that is, they take effect at the *next* rising clock edge from the one at which they are presented. For feedback operations from the 64-bit result (before the output register), each register to receive data must also be selected for loading in the usual way, by asserting the corresponding SELA or SELB; thus, you would assert the FDBK controls in one cycle and the SELA or SELB controls in the next cycle. For feedback, all input registers are loaded in parallel on the rising clock edge.

Both SP and DP feedback transfers are supported. In DP feedback transfers, the MSW is written to A_2 or B_2, and the LSW is written to A_3 or B_3. In SP feedback transfers, the 32-bit result is written to A_2 or B_2; A_3 or B_3 should not be selected in this case. The registers in the bank *not* selected by FDBK0:1 can be loaded concurrently in the normal manner. Also, the low-order (0 and 1) registers in the selected bank can be loaded concurrently in the normal manner. You should not, however, load registers intended to receive feedback data in the cycle before the feedback data is written (the same cycle in which you assert the FDBK controls). In theory, such an action would serve no purpose, because the newly loaded data would be overwritten by the fed-back data before it could be passed to the processing circuits; in practice, the data load will actually inhibit the feedback.

When feedforward is selected, a pair of input registers will be fed directly to the output port in the same cycle. RDA0 determines whether A registers (HI) or B registers (LO) are fed forward. The pair is selected by the register read selection controls, RDA1/B1, as shown in Figure 22.

RDA0	RDA1	RDB1	Feedforward Registers
0	X	0	B2/B3
0	X	1	B0/B1
1	0	X	A2/A3
1	1	X	A0/A1

Figure 22. Feedforward Register Selection

The pair fedforward must have been in input registers from a previous cycle but will reach the output register on the rising edge following the cycle in which the FDBK controls are set up for a feedforward operation (and the read selection controls are also set up). There is no cycle time or output delay penalty for feedforward operations.

For normal operation, FDBK0:1 must both be LO. Feedback and feedforward timing is shown in Figures T5 and T6 in the Timing section.

Absolute Value – ABSA/B Controls (REG)
The registered absolute value controls convert an operand selected by the read selection controls to its absolute value before processing. Asserting ABSA (HI) causes the A operand to be converted to its absolute value; asserting ABSB (HI) causes the B operand to be converted to its absolute value. The contents of the input registers remain unaffected.

With the ADSP-3222 ALU, the ABSA/B controls are effective with most fixed-point and all single-precision and double-precision operations. If the ABSA/B controls are asserted in logical operations, the results will be undefined.

For the ADSP-3212 Multiplier/Divider, the absolute value operation is available on single-precision and double-precision

floating-point operands only. If the ABSA/B controls are asserted with a multiplier/divider for a fixed-point operation, the results will be undefined.

Wrapped Input – WRAPA/B Controls (REG) (and INEXIN and RNDCARI on the ADSP-3222)
The ADSP-3212 cannot operate directly on denormals; denormals to be multiplied must first be converted by an ALU to the "wrapped" format. (See "Gradual Underflow and IEEE Exceptions" below.) The multiplier/divider must be told that an input is in the wrapped format so that its exponent can be interpreted properly as a twos-complement number.

The registered WRAPA/B controls inform the multiplier/divider that a wrapped number has been selected as an operand (RDA/B controls) to the multiplier/divider array. WRAPA indicates (HI) that the selected A register contains a wrapped number; WRAPB, that the selected B register contains a wrapped number.

The ALU in general operates directly on denormals and hence doesn't need a similar set of controls. However, for IEEE division and square root operations, the ALU cannot operate directly on denormals. Like the multiplier/divider, it needs denormals to be converted to wraps before processing. To indicate that the dividend in the A register is a wrapped, INEXIN should be asserted (HI) exactly as WRAPA would be asserted on a multiplier/divider. To indicated that either the divisor in a B register or a square root operand in a B register is wrapped, RNDCARI should be asserted (HI). Except for unwrap, division, and square root operations, both INEXIN and RNDCARI should be held LO.

Twos-Complement Input – TCA/B Controls (REG)
The registered ADSP-3212's Twos-Complement Input Controls inform the multiplier/divider to interpret the selected fixed-point inputs in the twos-complement data format. (See "32-Bit Fixed-Point Data Formats" above.) TCA HI indicates that the selected A register is twos-complement; TCB HI indicates a twos-complement B register. A LO value on either control for fixed-point multiplication indicates that the selected input is in unsigned-magnitude format. Mixed-mode (twos-complement times unsigned-magnitude) multiplications are permitted. The TCA/B controls are operative in fixed-point mode only; in floating-point mode, they are ignored.

Rounding – RND Controls (REG)
For floating-point operations, the ADSP-3212/ADSP-3222 chipset supports all four rounding modes of IEEE Standard 754. These are: Round-to-Nearest, Round-toward-Zero, Round-toward-Plus-Infinity and Round-toward-Minus-Infinity. For fixed-point operations, two rounding modes are available: Round-to-Nearest and Unrounded.

Rounding is involved in all operations in which the precision of the destination format is less than the precision of the intermediate results from the operation. Multiplications internally generate twice as many bits in the intermediate result significand as can be stored in the destination format. Data conversions to a destination format of lesser precision than the source also always force rounding unless the source value fits exactly.

Rounding with the ADSP-3212/ADSP-3222 chipset is controlled by a pair of pipelined, registered round controls, RND0:1. They should be set up with the input data whose result is to be rounded. Rounding is performed in the last stage of processing; the output register always contains rounded results. The effects of the round controls are defined as:

Mnemonic	RND1	RND0	Floating-Point	Fixed-Point
RN	0	0	Round-to-Nearest	Round-to-Nearest
RZ	0	1	Round-toward-Zero	Unrounded
RP	1	0	Round-toward-Plus-Infinity	illegal state
RM	1	1	Round-toward-Minus-Infinity	illegal state

Figure 23. Round Controls

The four floating-point modes of the IEEE Standard can be summarized as follows. In all cases, if the result before rounding can be expressed exactly in the destination format without loss of accuracy, then that will be the destination format result, regardless of specified rounding mode.

Round-toward-Plus-Infinity (RP): "When rounding toward $+\infty$, the result shall be the format's value (possibly $+\infty$) closest to and no less than the infinitely precise result." (Standard 754-1985, Sec. 4.2.) If the result before rounding (the "infinitely precise result") is not exactly representable in the destination format, then the result will be that number which is nearer to positive infinity. Round-toward-Plus-Infinity is available in floating-point operations only. If the result before rounding is greater than NORM.MAX but not equal to Plus Infinity, the result will be Plus Infinity. If the result before rounding is less than $-$NORM.MAX but not equal to Minus Infinity, the result will be $-$NORM.MAX. For fixed-point destination formats, the results of RP are undefined.

Round-toward-Minus-Infinity (RM): "When rounding toward $-\infty$, the result shall be the format's value (possibly $-\infty$) closest to and no greater than the infinitely precise result." (Standard 754-1985, Sec. 4.2.) If the result before rounding is not exactly representable in the destination format, the result will be that number which is nearer to Minus Infinity. Round-toward-Minus-Infinity is available in floating-point operations only. If the result before rounding is greater than NORM.MAX but not equal to Plus Infinity, the result will be NORM.MAX. If the result before rounding is less than $-$NORM.MAX but not equal to Minus Infinity, the result will be Minus Infinity. For fixed-point destination formats, the results of RM are undefined.

Round-toward-Zero and Unrounded (RZ): "When rounding toward 0, the result shall be the format's value closest to and no greater in magnitude than the infinitely precise result." (Standard 754-1985, Sec. 4.2.) If the result before rounding is not exactly representable in the destination format, the result will be that number which is nearer to zero. The Round-toward-Zero operation is available in floating-point operations only. It is equivalent to truncation of the (unsigned-magnitude) significand. If the result before rounding has a magnitude greater than NORM.MAX but not equal to Infinity, the result will be NORM.MAX of the same sign.

For fixed-point destination formats, the RZ mode is *Unrounded.* For fixed-point operations, RZ has no effect on the result at the output register and should be specified whenever unmodified fixed-point results are desired. (Treating the unrounded most significant product as the final result and throwing away the LSP is logically equivalent to Round-toward-Minus-Infinity for

twos-complement numbers and equivalent to Round-toward-Zero [truncation] for unsigned-magnitude numbers.)

Round-to-Nearest (RN): When rounding to nearest, "... the representable value nearest to the infinitely precise result shall be delivered; if the two nearest representable values are equally near, the one with its least significant bit zero shall be delivered." (Standard 754-1985, Sec. 4.1). If the result before rounding is not exactly representable in the destination format, the result will be that number which is nearer to the result before rounding. In the case that the result before rounding is exactly half way between two numbers in the destination format differing by an LSB, the result will be that number which has an LSB equal to zero. If the result before rounding overflows, i.e., has a magnitude greater than or equal to NORM.MAX + 1/2 LSB in the destination format, the result will be the Infinity of the same sign.

Round-to-Nearest is available in both floating-point and fixed-point operations. In fixed-point, Round-to-Nearest treats the most significant product *after* having been shifted in accordance with SHLP (see Figures 9, 10, 12, and 13) as the destination format.

The four rounding modes are illustrated by number lines in Figure 24. The direction of rounding is indicated by an arrow. Numbers exactly representable in the destination format are indicated by "•"s. In subdividing the number lines, square brackets are inclusive of the points on the line they intersect. Note that brackets intersect points representable in the destination format except for Round-to-Nearest, where they intersect the line midway between representable points. Slashes are used to indicate a break in the number line of arbitrary size.

Note that Round-to-Nearest is unique among the rounding modes in that it is *unbiased*. The large-sample statistical mean from a set of numbers rounded in the other modes will be displaced from the true mean. The other three modes will exhibit a large-sample statistical *bias* in the direction of the rounding operation performed.

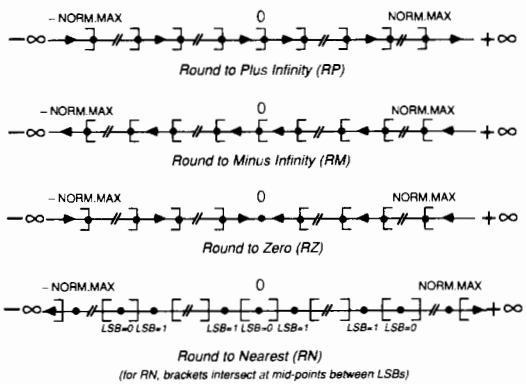


Figure 24. IEEE Rounding Modes

Status Flags

The ADSP-3212/ADSP-3222 chipset generates on dedicated pins the following exception flags specified in the IEEE Standard: Overflow (OVRFLO), Underflow (UNDFLO), Inexact Result (INEXO), and Invalid Operation (INVALOP). The IEEE exception condition Division-by-Zero is flagged by the simultaneous assertion of both OVRFLO and INVALOP pins. The five IEEE exceptions are defined in accordance to the default assumption of Standard 754 of non-trapping exceptions. The ADSP-3222 also generates a ZERO flag to indicate that the result of the ALU operation is ZERO.

Flag results are registered in the status output register when the results they reflect are clocked to the output register. They are held valid until the next rising clock edge. The IEEE Standard specifies that exception flags when set remain set until reset by the user. For full conformance to the Standard, the status outputs from this chipset should be individually latched externally.

Denormal

In addition to the IEEE status flags, the ADSP-3212 Multiplier/Divider has a DENORM output flag that signals the presence of a denormalized number at one of the input registers being read into the multiplier array. This denormal must be wrapped by the ALU before the multiplier/divider can read it. To minimize the system response time to a denormal input exception, the DENORM flag comes out earlier than the associated IEEE status flags. For both multiplication and division, DENORM goes HI during the cycle after a denormal was read into the array (with the RDA/B controls). See Figures T7 through T10. In the cycle following the assertion of DENORM, the INEXO status flag (see "Inexact," below) indicates which operand was denormal; INEXO is HI if the B operand or both operands were denormal and LO if only the A operand was denormal. The DENORM flag is asserted in both IEEE and FAST modes.

Some multiplications with denormal operands do not require wrapping and therefore do not cause the assertion of the DENORM flag. These are DNRM•ZERO, DNRM•INF, and DNRM•NAN. Multiplication of a finite number by zero always yields zero—the result the multiplier/divider will produce anyway—so there is no need to signal an exception. Any finite nonzero number multiplied by INF should yield INF, and in the IEEE mode, the ADSP-3212 Multiplier/Divider will produce this result with a DNRM operand, hence no wrapping is required. In FAST mode, DNRM is treated as ZERO, so a NAN results, and no wrapping is needed. And multiplication of any number by a NAN produces a NAN (and the INVALOP flag); no wrapping is necessary for the multiplier/divider to produce this correct IEEE result.

Similarly, divisions that have a denormal operand and ZERO, INF, or NAN as the other operand do not require wrapping and do not cause the assertion of the DENORM flag on the multiplier/divider (or the INVALOP and UNDFLO combination on the ALU, which flags a denormal operand for division or square root). Zero divided by a finite number always yields zero, so in IEEE mode, ZERO ÷ DNRM yields ZERO without signalling an exception. DNRM ÷ ZERO results in INF, because any finite nonzero number divided by zero should yield INF. In FAST mode, DNRM is treated as ZERO, so ZERO ÷ DNRM and DNRM ÷ ZERO both yield a NAN (and INVALOP). Any finite number divided by INF should yield ZERO, and INF divided by any finite number should yield INF. In both IEEE and FAST modes, INF ÷ DNRM results in INF and DNRM ÷ INF results in ZERO, without generating any flags. Division of any number by a NAN or division of a NAN by any number produces a NAN (and the INVALOP flag); therefore, the multiplier/divider and the ALU generate this result without flagging a denormal and without wrapping.

Note that the ALU in general operates directly on denormals and therefore does not flag any exception. However, it cannot operate directly on denormals in its division and square root operations. For these operations, denormal inputs will cause the

simultaneous assertion of UNDFLO and INVALOP in IEEE mode. For divisions, INEXO LO indicates that the dividend is a DNRM; INEXO HI indicates that the divisor or both operands are DNRMs. In FAST mode, only INVALOP will be asserted. This denormal exception information becomes available with the status outputs, i.e., at the end of an attempted multicycle division or square root.

Invalid Operation and NAN results

INVALOP is generated whenever attempting to execute an invalid operation, as defined in Standard 754, Section 7.1. The INVALOP output is also used in conjunction with other pins to indicate the Division-by-Zero exception and denormal divisor or dividend (on the ADSP-3222). The default non-trapping result is required to be a quiet NAN. Except when passing a NAN with PASS or copying a sign bit to a NAN in the ALU, a NAN output will always have an exponent and fraction of all ones.

Conditions that cause the assertion of INVALOP are:

- NAN input read to computational circuitry (except for logical PASS); INEXO indicates the B operand is a NAN
- Multiplication of either ±INF by either ±ZERO
- In FAST mode, multiplication of either ±INF by either ±DNRM or ±ZERO
- Subtraction of liked-signed INFs or addition of opposite-signed INFs
- Conversion of a NAN or INF to fixed-point
- Wrapping an operand that is neither a denormal nor ZERO
- Division of either ± ZERO by either ± ZERO or of either ±INF by either ±INF. Division by ±ZERO yields INF rather than a NAN (see "Division-by-Zero," below).
- Attempting the square root of a negative number
- In conjunction with OVRFLO, the Division-by-Zero exception
- On the ADSP-3222 in FAST mode, a denormal divisor or dividend; in IEEE mode, in conjunction with UNDFLO, a denormal divisor or dividend
- In conjunction with UNDFLO, a denormal input operand to square root.

Division-by-Zero

The Division-by-Zero exception is generated whenever attempting to divide a finite nonzero dividend by a divisor of zero (Standard 754, Section 7.2). The Division-by-Zero exception is indicated by the simultaneous assertion of both OVRFLO and INVALOP. The result is a correctly signed INF.

Overflow

OVRFLO is generated whenever the unbounded (i.e., supposing hypothetically no bounds on the exponent range of the result), post-rounded result exceeds in magnitude NORM.MAX in the destination format, as defined in Standard 754, Section 7.3. Note that the overflow condition can occur both during computations and during data format conversions. The result in IEEE or FAST mode will be either ±INF or ±NORM.MAX, depending on the sign of the result and the operative rounding mode. (See "Rounding – RND Controls" above.) The OVRFLO pin is also used to signal additional exception conditions.

Conditions that cause the assertion of OVRFLO are:

- Unbounded, post-rounded result exceeds destination format in computation or conversion
- In conjunction with INVALOP, the Division-by-Zero exception
- Comparison when operand A is greater than operand B
- Exponent subtraction when the resultant exponent is more positive than can be represented in the destination format.

Underflow

Underflow is defined in four ways in Standard 754, Section 7.4. The IEEE Standard allows the implementer to choose which definition of underflow to use and provides no guidance. The first option is whether to flag underflow based on results before or after rounding. Consistent with the definition of overflow, underflow is always flagged with this chipset based on results *after* rounding (except for the operations of conversion from floating-point to fixed-point and logical downshifts). Thus, a result whose infinitely precise value is less than NORM.MIN yet which rounds to NORM.MIN will *not* be considered to have underflowed.

The second option is how to interpret what the Standard calls an "extraordinary loss of accuracy." The first way is in terms of the creation of nonzero, post-rounded numbers smaller in magnitude than NORM.MIN. The second way is in terms of loss of accuracy when representing numbers as denormals. With the ADSP-3212/ADSP-3222 chipset, the conditions under which UNDFLO is asserted depend on whether the chip in question can generate denormals in its current operating mode. If the chip cannot generate denormals, the definition in terms of numbers smaller in magnitude than NORM.MIN will apply; if it can generate denormals, the definition in terms of inexact denormals will apply. Thus, which definition applies will depend on whether the chipset is operating in IEEE or FAST mode, whether the result is generated by a multiplier/divider or an ALU and whether the operation is division.

With the ADSP-3212 Multiplier/Divider, UNDFLO is generated whenever the unbounded, post-rounded, nonzero result is of lesser magnitude than NORM.MIN in the destination format. In FAST mode, the data result will be ZERO; in IEEE mode the data result will be in the wrapped format. An exact ZERO result will never cause the assertion of UNDFLO.

With the ADSP-3222 ALU in the FAST mode, UNDFLO is also generated whenever the unbounded, post-rounded, nonzero result is of lesser magnitude than NORM.MIN in the destination format for standard ALU operations as well as for division and square root. For FAST mode underflows, the ALU result will always be ZERO. The only exception to this rule is for sums of and differences between DNRMs; if the unbounded, post-rounded, nonzero result of (DNRM ± DNRM) is of lesser magnitude than NORM.MIN in FAST, then UNDFLO will *not* be set. The ALU result will still be ZERO.

With the ADSP-3222 ALU in IEEE mode, UNDFLO is generated (except for divisions) whenever the unbounded, infinitely precise (i.e., supposing hypothetically no bounds on the precision of the result), post-rounded result is a denormal and does not fit into the denormal destination format *without a loss of accuracy*. In other words, UNDFLO will be generated whenever an inexact denormal result is produced. (See "Inexact" below.) If the result is a denormal and does fit exactly, neither UNDFLO nor INEXO will be asserted. Note that additions, subtractions and comparisons cannot generate this underflow condition

(since no operand contains significant bits of lesser magnitude than DNRM.MIN). IEEE-mode ALU underflow exceptions occur only during conversions and divisions.

The division operation is treated like a multiplication operation in IEEE mode rather than an ALU operation in the definition of underflow. A quotient from division smaller in magnitude than NORM.MIN will always be flagged as underflowed. The data result will be in the wrapped format. Note that $\sqrt{}$(DNRM.MIN) ≥ NORM.MIN. Therefore, square root will never underflow with operands greater than or equal to DNRM.MIN.

Conditions that cause the assertion of UNDFLO are:

- With the ADSP-3212 Multiplier/Divider, whenever the unbounded, post-rounded, nonzero result is of lesser magnitude than NORM.MIN in the destination format
- With the ADSP-3222 ALU in the FAST mode, whenever the unbounded, post-rounded, nonzero result is of lesser magnitude than NORM.MIN in the destination format
- With the ADSP-3222 ALU in IEEE mode, whenever an inexact denormal is produced or whenever the unbounded, post-rounded, nonzero quotient from division is of lesser magnitude than NORM.MIN in the destination format
- Conversions to integer if the magnitude of the floating-point source *before* rounding is less than one
- Conversions from DP floating-point to SP floating-point whenever the unbounded, post-rounded, nonzero result is less than SP DNRM.MIN or whenever an inexact denormal is produced
- Comparison when operand A is less than operand B
- Attempting to wrap a ZERO
- Unwrapping if there is a loss of accuracy
- Exponent subtraction when the resultant exponent is more negative than can be represented in the destination format
- Logical downshift that *before* rounding would have shifted all bits out of the destination format
- With the ADSP-3222 ALU in IEEE mode, in conjunction with INVALOP, a denormal divisor or dividend
- A quotient from division less than NORM.MIN
- In IEEE mode, in conjunction with INVALOP, a denormal input operand for square root.

Inexact

The inexact exception is defined in Standard 754, Section 7.5, as the loss of accuracy of the unbounded, infinitely precise result when fitted to the destination format. It is signalled on the ADSP-3212/ADSP-3222 chipset by INEXO.

For fixed-point multiplications, the ADSP-3212 Multiplier/Divider will assert INEXO HI if and only if any of the least significant 32 bits of the pre-rounded 64-bit product are ones.

INEXO is also used for signaling various other conditions. If the ADSP-3212 or the ADSP-3222 detects a NAN input to a floating-point operation, it asserts INVALOP. At the same time, it asserts INEXO if the B operand or both operands are NANs. If the ADSP-3222 detects a denormal input to division, it asserts UNDFLO and INVALOP to flag the denormal and asserts INEXO if the B operand or both operands are denormals. Similarly, the ADSP-3212 asserts DENORM to flag a denormal input to floating-point multiplication or division. INEXO, in that context, signals which of the two was the denormal: INEXO HI indicates that the B operand or both operands are denormals; INEXO LO indicates that only the A operand is a denormal.

If you convert a fixed-point number in the B input to floating-point format and the fixed-point number has the same bit pattern as a floating-point NAN (i.e., bits 23-30 are all ones and at least one of bits 0-22 is a one), INEXO is asserted. This occurs only in the ALU's DFLOATB and SFLOATB instructions, not DFLOATA or SFLOATA. These instructions are described in *Floating-Point ALU Operations*.

Conditions that cause the assertion of INEXO are:

- Loss of accuracy when fitting result to destination format
- For fixed-point multiplications, the pre-rounded 64-bit product contains ones in the least-significant 32-bits
- For floating-point operations, in conjunction with INVALOP, the B operand is a NAN
- In IEEE mode on the ADSP-3222, in conjunction with UNDFLO and INVALOP, dividend is a denormal (LO) or divisor is a denormal or both are denormals (HI)
- In IEEE mode on the ADSP-3212, in conjunction with DENORM, A operand is a denormal (LO) or B operand is a denormal or both are denormals (HI).

Zero

The ADSP-3222 has a dedicated ZERO flag. ZERO goes HI whenever the post-rounded result in the output register is ±ZERO, except for a few floating-point instructions. Thus, inexact results that round to ±ZERO are flagged, as well as ±ZERO results from format conversions and logical operations. The instructions for which ZERO is not valid are: SXSUB, DXSUB (exponent subtract), SITRN, DITRN (logical downshift), SFIXA, DFIXA, SFIXB and DFIXB (floating-point to fixed-point conversion). For these instructions, the state of ZERO is undefined.

Less Than, Equal, Greater Than and Unordered

For comparison operations in the ALU, the OVRFLO, UNDFLO and INVALOP status outputs are used to indicate the four comparison conditions of IEEE Standard 754, Section 5.7. They are defined as follows:

- "Less than" is signalled by the assertion of UNDFLO (while OVRFLO is LO)
- "Equal" is signalled by *not* asserting either OVRFLO or UNDFLO (i.e., both LO)
- "Greater than" is signalled by the assertion of OVRFLO (while UNDFLO is LO)
- "Unordered" is signalled by the assertion of INVALOP, caused by attempting a comparison with at least one NAN operand.

The data result from a comparison operation is identical to subtracting operand B from operand A. See Tables XIV and XV.

In IEEE comparisons, the data types are always ordered in ascending sequence: −INF, −NORM, −DNRM, ZERO, DNRM, NORM and INF. Comparisons between like signed INFs will generate the "Equal" status condition. Comparisons between signed ZEROs will also generate the "Equal" status. Any comparison to a NAN will also cause INVALOP and produce an all-ones NAN. Even in FAST mode, DNRMs will be compared based on their true value (rather than all being treated as ZEROs).

Special Flags for Unwrapping

The ADSP-3212 generates a Round Carry Propagation Out flag, RNDCARO, that indicates whether or not a carry bit propagated into the destination format's fraction during the multiplier/divider's floating-point rounding operation. The rounding

that the multiplier/divider does in creating the wrapped or unnormal result may cause a carry bit into the LSB in the destination's format's fraction. This rounding position will not in general be correct for a properly rounded denormal. Thus, when the underflowed multiplier/divider result is unwrapped to a denormal, the ALU has to undo the multiplier/divider's rounding and re-round to achieve the properly rounded denormal.

To do this, the ALU has to know if any carry bits in the multiplier/divider's rounding operation propagated into the fraction of the result. This information is provided in the multiplier/divider's RNDCARO flag. The ALU also needs to know if the multiplier/divider's rounded result caused a loss of accuracy when expressed in its destination wrapped format, indicated by the multiplier/divider's Inexact Result (INEXO) flag.

The ADSP-3222 ALU has a corresponding pair of flag status input pins: Round Carry Propagation In (RNDCARI) and Inexact Data In (INEXIN). In an unwrap operation, these flags are used by the ALU when converting from a WRAP to a DNRM to obtain the properly rounded result. RNDCARI and INEXIN should be setup to the ALU with the instruction for the unwrap operation. Both multiplier/divider and ALU must be using the same rounding mode.

The ADSP-3222 ALU itself generates WRAPs in underflowed division operations. These WRAPs must be fed back to the ALU to be unwrapped to DNRMs. The ADSP-3222, unlike the multiplier/divider, does not have a RNDCARO pin to signal whether or not a carry bit propagated into the destination format on rounding. For this reason, WRAPs produced by the ADSP-3222 ALU in division are rounded differently than they are on the multiplier/divider; underflowed (only) quotients are always truncated (Round-toward-Zero) to the destination wrapped format. Hence there *is* no carry bit propagation. When unwrapping a WRAP quotient produced by the ALU, RNDCARI should always be held LO. INEXIN should reflect the status of INEXO when the ALU produced the underflowed wrapped quotient.

The ADSP-3222 ALU also uses the RNDCARI and INEXIN pins to indicated wrapped A and B operands, respectively, to ALU division and square root operations. Both RNDCARI and INEXIN should be held LO except for unwrap, ALU division, and square root operations.

Instructions and Operations

The ADSP-3212 multiplier/divider executes one of two instructions: multiply or divide. If the DIVMUL input is LO, the ADSP-3212 multiplies; if DIVMUL is HI, the ADSP-3212 divides. The instruction need not be specified explicitly in microcode. The data format of results and status flags from multiplication are shown in Tables V and VI. Format and status for division are shown in Tables VII and VIII.

A denormal input operand will generally cause the DENORM exception (see "Status Flags" above) unless the other operand is ZERO, INF or a NAN. (See Tables V through VIII.) The sign bit of the NAN generated from any invalid operation will depend on the operands. (The IEEE Standard does not specify conditions for the sign bit of a NAN.) On the ADSP-3212 Multiplier/Divider, the sign of a NAN result will be the exclusive OR of the signs of the input operands.

4

A operand \ B operand	ZERO		DNRM		WRAP		NORM		INF		NAN	
	result	*status*	*result*	*status*	*result*	*status*	*result*	*status*	*result*	*status*	*result*	*status*
ZERO	ZERO		ZERO		ZERO		ZERO		NAN	INVALOP	NAN	INVALOP INEXO
DNRM	ZERO		ZERO	DENORM	ZERO	DENORM	ZERO	DENORM	INF		NAN	INVALOP INEXO
WRAP	ZERO		ZERO	DENORM	UNRM	UNDFLO	NORM WRAP UNRM	UNDFLO UNDFLO	INF		NAN	INVALOP INEXO
NORM	ZERO		ZERO	DENORM	NORM WRAP UNRM	UNDFLO UNDFLO	INF,NORM,MAX[1] NORM WRAP	OVRFLO UNDFLO	INF		NAN	INVALOP INEXO
INF	NAN	INVALOP	INF		INF		INF		INF		NAN	INVALOP INEXO
NAN	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP INEXO

1. Either INF or NORM.MAX, depending on rounding mode. See "Round Controls."

Table V. ADSP-3212 Floating-Point Multiplication (IEEE Mode)

A operand \ B operand	ZERO		DNRM		NORM		INF		NAN	
	result	*status*	*result*	*status*	*result*	*status*	*result*	*status*	*result*	*status*
ZERO	ZERO		ZERO		ZERO		NAN	INVALOP	NAN	INVALOP INEXO
DNRM	ZERO		ZERO	DENORM	ZERO	DENORM	NAN	INVALOP	NAN	INVALOP INEXO
NORM	ZERO		ZERO	DENORM	INF,NORM,MAX[1] NORM ZERO	OVRFLO UNDFLO	INF		NAN	INVALOP INEXO
INF	NAN	INVALOP	NAN	INVALOP	INF		INF		NAN	INVALOP INEXO
NAN	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP INEXO

In FAST mode, WRAP inputs are illegal.

1. Either INF or NORM.MAX, depending on rounding mode. See "Round Controls."

Table VI. ADSP-3212 Floating-Point Multiplication (FAST Mode)

The product of INF with anything except ZERO or NAN is a correctly signed INF. INF•ZERO will cause INVALOP and yield a NAN. A NAN operand in either multiplication or division will also cause INVALOP and yield a NAN. If the NAN is the B operand, INEXO is also asserted.

A operand \ B operand	ZERO		DNRM		WRAP		NORM		INF		NAN	
	result	status	result	status	result	status	result	status	result	status	result	status
ZERO	NAN	INVALOP	ZERO		ZERO		ZERO		ZERO		NAN	INVALOP INEXO
DNRM	INF	INVALOP OVRFLO	NAN	DENORM	NAN	DENORM	NAN	DENORM	ZERO		NAN	INVALOP INEXO
WRAP	INF	INVALOP OVRFLO	NAN	DENORM	NORM		NORM WRAP	UNDFLO	ZERO		NAN	INVALOP INEXO
NORM	INF	INVALOP OVRFLO	NAN	DENORM	INF, NORM MAX[1] NORM	OVRFLO	INF, NORM MAX[1] NORM WRAP	OVRFLO UNDFLO	ZERO		NAN	INVALOP INEXO
INF	INF		INF		INF		INF		NAN	INVALOP	NAN	INVALOP INEXO
NAN	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP INEXO

1. Either INF or NORM MAX, depending on rounding mode. See "Round Controls."

Table VII. ADSP-3212 Floating-Point Division (A÷B) (IEEE Mode)

A operand \ B operand	ZERO		DNRM		NORM		INF		NAN	
	result	status	result	status	result	status	result	status	result	status
ZERO	NAN	INVALOP	NAN	INVALOP	ZERO		ZERO		NAN	INVALOP INEXO
DNRM	NAN	INVALOP	NAN	INVALOP DENORM	ZERO	DENORM	ZERO		NAN	INVALOP INEXO
NORM	INF	INVALOP OVRFLO	INF	OVRFLO INVALOP DENORM	INF, NORM MAX[1] NORM ZERO	OVRFLO UNDFLO	ZERO		NAN	INVALOP INEXO
INF	INF		INF		INF		NAN	INVALOP	NAN	INVALOP INEXO
NAN	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP INEXO

In FAST mode, WRAP inputs are illegal.

1. Either INF or NORM MAX, depending on rounding mode. See "Round Controls."

Table VIII. ADSP-3212 Floating-Point Division (A÷B) (FAST Mode)

The ADSP-3222 ALU, in contrast to the multiplier/divider, is instruction driven with the operation specified by I_{8-0}. The ALU instructions fall into five categories: Fixed-Point, Logical, Single-Precision Floating-Point, Double-Precision Floating-Point and Miscellaneous. Instructions are summarized in Tables IX through XIII and described in this section below. The data format of results and status flags from the various ALU operations are shown in Tables XIV and XV. Division is shown in Tables XIX and XX; square root in Table XXI. Conversions are illustrated in Tables XVI, XVII, and XVIII.

Mnemonic	Instruction (I_{8-0})			Description
	I_{8-6}	I_{5-3}	I_{2-0}	
IADD	001	000	011	Fixed-point A + B
ISUBB	001	001	011	Fixed-point A – B
ISUBA	001	000	111	Fixed-point B – A
IADDWC	001	010	011	Fixed-point A + B with carry
ISUBWBB	001	011	011	Fixed-point A – B with borrow
ISUBWBA	001	010	111	Fixed-point B – A with borrow
INEGA	001	000	101	Fixed-point –A. ABSA/B must be LO.
INEGB	001	001	010	Fixed-point –B. ABSA/B must be LO.
IADDAS	001	100	011	Fixed-point |A + B|
ISUBBAS	001	101	011	Fixed-point |A – B|
ISUBAAS	001	100	111	Fixed-point |B – A|

Table IX. ADSP-3222 Fixed-Point ALU Operations

Mnemonic	Instruction (I_{8-0})			Description
	I_{8-6}	I_{5-3}	I_{2-0}	
COMPLA	000	000	101	Ones-complement A
COMPLB	000	001	010	Ones-complement B
PASSA	000	000	001	Pass A unmodified. Set no flags.
PASSB	000	000	010	Pass B unmodified. Set no flags.
AANDB	000	010	010	Bitwise logical AND
AORB	000	100	010	Bitwise logical OR
AXORB	000	110	010	Bitwise logical XOR
NOP	000	000	000	No operation. Preserve status flags and Output Register contents.
CLR	100	000	000	Clear all status flags. Data register contents are unaffected.

Table X. ADSP-3222 ALU Logical Operations

Mnemonic	Instruction (I_{8-0})			Description
	I_{8-6}	I_{5-3}	I_{2-0}	
SADD	111	000	011	SP FltgPt (A + B)
SSUBB	111	000	111	SP FltgPt (A – B)
SSUBA	111	001	011	SP FltgPt (B – A)
SCOMP	111	001	111	SP FltgPt comparison of A to B. Result is (A – B). Greater Than: OVRFLO HI Equal: OVRFLO LO, UNDFLO LO Less Than: UNDFLO HI Unordered: INVALOP HI
SADDAS	011	000	011	SP FltgPt |A + B|
SSUBBAS	011	000	111	SP FltgPt |A – B|
SSUBAAS	011	001	011	SP FltgPt |B – A|
SFIXA	011	001	101	Convert SP FltgPt A to twos-complement Integer
SFIXB	011	001	110	Convert SP FltgPt B to twos-complement Integer
SFLOATA	011	100	101	Convert twos-complement integer A to SP FltgPt
SFLOATB	011	100	110	Convert twos-complement integer B to SP FltgPt
DOUBLEA	011	101	101	Convert SP FltgPt A to DP FltgPt
DOUBLEB	011	101	110	Convert SP FltgPt B to DP FltgPt
SPASSA	011	110	001	Pass SP FltgPt A. NANs cause INVALOP.
SPASSB	011	110	010	Pass SP FltgPt B. NANs cause INVALOP.
SWRAPA	011	100	001	Wrap SP DNRM A to SP WRAP
SWRAPB	011	100	010	Wrap SP DNRM B to SP WRAP
SUNWRAPA	011	010	001	Unwrap SP WRAP A to SP DNRM
SUNWRAPB	011	010	010	Unwrap SP WRAP B to SP DNRM
SSIGN	011	111	101	Copy sign from SP FltgPt B to SP FltgPt A. Result is [sign B, exponent A, fraction A].
SXSUB	011	111	001	Subtract B exponent from A exponent. Result is [sign A, (expt A – expt B), fraction A] for all data types. If the unbiased exponent is ≥ +128, INF results. If the unbiased exponent is ≤ −127, ZERO results.
SITRN	011	010	101	Downshift SP FltgPt A mantissa (with hidden bit) logically by the unbiased SP FltgPt B exponent to a 32-bit unsigned-magnitude integer. Use RZ only.
SDIV	011	110	111	SP FltgPt (A÷B)
SSQR	111	110	110	SP FltgPt $\sqrt{B}$

Table XI. ADSP-3222 ALU Single-Precision Floating-Point Operations

Mnemonic	Instruction (I_{8-0})			Description
	I_{8-6}	I_{5-3}	I_{2-0}	
DADD	110	000	011	DP FltgPt (A + B)
DSUBB	110	000	111	DP FltgPt (A – B)
DSUBA	110	001	011	DP FltgPt (B – A)
DCOMP	110	001	111	DP FltgPt comparison of A to B. Result is (A – B). Greater Than: OVRFLO HI Equal: OVRFLO LO, UNDFLO LO Less Than: UNDFLO HI Unordered: INVALOP HI
DADDAS	010	000	011	DP FltgPt |A + B|
DSUBBAS	010	000	111	DP FltgPt |A – B|
DSUBAAS	010	001	011	DP FltgPt |B – A|
DFIXA	010	011	101	Convert DP FltgPt A to twos-complement integer DFIXB
	010	011	110	Convert DP FltgPt B to twos-complement integer
DFLOATA	010	100	101	Convert twos-complement integer A (even A register sources only) to DP FltgPt
DFLOATB	010	100	110	Convert twos-complement integer B (even B register sources only) to DP FltgPt
SINGLEA	110	011	101	Convert DP FltgPt A to SP FltgPt
SINGLEB	110	011	110	Convert DP FltgPt B to SP FltgPt
DPASSA	010	110	001	Pass DP FltgPt A. NANs cause INVALOP.
DPASSB	010	110	010	Pass DP FltgPt B. NANs cause INVALOP.
DWRAPA	010	100	001	Wrap DP DNRM A to DP WRAP
DWRAPB	010	100	010	Wrap DP DNRM B to DP WRAP
DUNWRAPA	010	010	001	Unwrap DP WRAP A to DP DNRM
DUNWRAPB	010	010	010	Unwrap DP WRAP B to DP DNRM
DSIGN	010	111	101	Copy sign from DP FltgPt B to DP FltgPt A. Result is [sign B, exponent A, fraction A].
DXSUB	010	111	001	Subtract B exponent from A exponent. Result is [sign A, (expt A – expt B), fraction A] for all data types. If the unbiased exponent is ≥ +1024, INF results. If the unbiased exponent is ≤ −1023, ZERO results.
DITRN	010	010	101	Downshift DP FltgPt A mantissa (with hidden bit) logically by the unbiased DP FltgPt B exponent to a 32-bit unsigned-magnitude integer. Use RZ only.
DDIV	010	110	111	DP FltgPt (A÷B)
DSQR	110	110	110	DP FltgPt $\sqrt{B}$

Table XII. ADSP-3222 ALU Double-Precision Floating-Point Operations

Mnemonic	Instruction (I_{8-0})			Description
	I_{8-6}	I_{5-3}	I_{2-0}	
HOLDEN	100	000	100	Redefine FAST to $\overline{\text{HOLD}}$ control
LOAD64	100	001	100	Enable 64-Bit parallel data loading

Table XIII. ADSP-3222 ALU Miscellaneous Operations

Fixed-Point Arithmetic ALU Operations

The negation operation is a twos-complementing of the input operand.

The OVRFLO flags can be set by fixed-point ALU operations. The twos-complement data format is presumed in the definition of fixed-point overflow.

Absolute Value Controls

Absolute value controls (ABSA/B) cannot be used with all operands input to all fixed-point ALU operations. ABSA/B must be LO for negation (INEGA/B) operations or results will be undefined. Absolute value controls can be used with all other fixed-point operations.

Extended-Precision Fixed-Point Arithmetic

The ADSP-3222's fixed-point ALU operations include three operations for extended fixed-point precision: addition with carry and two subtractions with borrow. The carry bit generated by an addition or subtraction is latched internally for one cycle only.

To illustrate, these instructions can be used to add two 64-bit fixed-point numbers. The two least-significant 32-bit halves can be added with IADD. Any carry bit generated would be latched internally in the ADSP-3222. On the next cycle, the most significant 32-bit halves can be added with IADDWC which would also add in the carry bit from the previous operation if any. The two fixed-point results will be latched in the output register in consecutive cycles. As with all fixed-point results,

they will appear in consecutive cycles in the most significant 32-bits of the output register (bit positions 63 through 32).

Extended precision fixed-point subtraction is exactly analogous. The least significant 32-bit halves can be subtracted with either ISUBA or ISUBB. On the next cycle, the most significant 32-bit halves can be subtracted with either ISUBWBA or ISUBWBB.

Logical ALU Operations

The ones-complement instructions (COMPLA/B) change every one bit in the operand to a zero bit and every zero bit in the operand to a one bit. Ones-complementing is equivalent to a bitwise logical NOT operation on the 32-bit operand. The pass instructions (PASSA/B) pass all operands unmodified, including NANs, without signaling an INVALOP exception. PASSA/B set no flags.

The logical AND, OR and XOR (AANDB, AORB, AXORB) operate bitwise on all 32-bits in their pair of operand fields to produce a 32-bit result.

NOP will advance the ALU pipeline one cycle. Both the status flags and the output register will be preserved during NOP. CLR simply resets all status flags. Note that CLR is pipelined and takes effect one cycle after it is presented. All data register contents, including the output register, remain unaffected.

Do not assert the absolute value controls (ABSA/B) with logical operations. The results will be undefined.

Floating-Point ALU Operations

The single-precision and double-precision floating-point operations are exactly analogous and both will be discussed here. The data types and flags resulting from additions, subtractions, comparisons, absolute sums, and absolute differences are shown in Tables XIV and XV. The INEXO flag is not shown explicitly in these tables (or any other) since it may or may not be set, depending on whether the result is inexact.

A operand \ B operand	ZERO		DNRM		NORM		INF		NAN	
	result	*status*	*result*	*status*	*result*	*status*	*result*	*status*	*result*	*status*
ZERO	ZERO[2]		DNRM		NORM		INF		NAN	INVALOP INEXO
DNRM	DNRM		NORM DNRM ZERO		INF,NORM,MAX[1] NORM DNRM	OVRFLO	INF		NAN	INVALOP INEXO
NORM	NORM		INF,NORM,MAX[1] NORM DNRM	OVRFLO	INF,NORM,MAX[1] NORM DNRM ZERO	OVRFLO	INF		NAN	INVALOP INEXO
INF	INF		INF		INF		INF[3] NAN[3]	INVALOP	NAN	INVALOP INEXO
NAN	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP INEXO

1. Either INF or NORM,MAX, depending on rounding mode. See "Round Controls."
2. (± ZERO) + (± ZERO) ⇒ ± ZERO
   (± ZERO) + (∓ ZERO) ⇒ + ZERO (RN, RZ, RP rounding modes)
   (± ZERO) + (∓ ZERO) ⇒ – ZERO (RM rounding mode)
3. (± INF) + (± INF) ⇒ ± INF
   (± INF) + (∓ INF) ⇒ +NAN (RN, RZ, RP rounding modes)
   (± INF) + (∓ INF) ⇒ –NAN (RM rounding mode)
4. If DNRM result is inexact, UNDFLO will be set.
5. The ADSP-3222 does not accept wrapped numbers as inputs for standard arithmetic operations.

Table XIV. ADSP-3222 Floating-Point Addition/Subtraction (IEEE Mode)

A operand \ B operand	ZERO		DNRM		NORM		INF		NAN	
	result	*status*	*result*	*status*	*result*	*status*	*result*	*status*	*result*	*status*
ZERO	ZERO[2]		ZERO	UNDFLO	NORM		INF		NAN	INVALOP INEXO
DNRM	ZERO	UNDFLO	NORM ZERO		INF,NORM,MAX[1] NORM ZERO	OVRFLO UNDFLO	INF		NAN	INVALOP INEXO
NORM	NORM		INF,NORM,MAX[1] NORM ZERO	OVRFLO UNDFLO	INF,NORM,MAX[1] NORM ZERO ZERO[4]	OVRFLO UNDFLO	INF		NAN	INVALOP INEXO
INF	INF		INF		INF		INF[3] NAN[3]	INVALOP	NAN	INVALOP INEXO
NAN	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP INEXO

In FAST mode, WRAP inputs are illegal.

1. Either INF or NORM,MAX, depending on rounding mode. See "Round Controls."
2. (± ZERO) + (± ZERO) ⇒ ± ZERO
   (± ZERO) + (∓ ZERO) ⇒ + ZERO (RN, RZ, RP rounding modes)
   (± ZERO) + (∓ ZERO) ⇒ – ZERO (RM rounding mode)
3. (± INF) + (± INF) ⇒ ± INF
   (± INF) + (∓ INF) ⇒ +NAN (RN, RZ, RP rounding modes)
   (± INF) + (∓ INF) ⇒ –NAN (RM rounding mode)
4. Exact result

Table XV. ADSP-3222 Floating-Point Addition/Subtraction (FAST Mode)

Absolute Value Controls
Absolute value controls (ABSA/B) can be used with all operands input to all floating-point ALU operations.

Sign of NAN Results
On the ADSP-3222, the sign of a NAN resulting from any operation (except division) involving at least one NAN operand will be the sign which would be produced if the magnitude portion (sign plus fraction) of the NAN operand(s) were treated as normal numbers.

Some ALU operations with two INF inputs can cause INVALOP and generate NANs. The assignment of sign to the NAN is analogous to additions with signed zeros:

$(\pm INF)+(\pm INF)=(\pm INF)-(\mp INF)\rightarrow \pm INF$
$(\pm INF)+(\mp INF)=(\pm INF)-(\pm INF)\rightarrow +NAN$
(RN, RZ, RP rounding modes)
$(\pm INF)+(\mp INF)=(\pm INF)-(\pm INF)\rightarrow -NAN$
(RM rounding mode)

In this notation, the expression to the left of the equals sign in the first line refers to (+INF) + (+INF) or to (−INF) + (−INF). The leading expressions on the second and third lines refer to (+INF) + (−INF) or to (−INF) + (+INF).

Comparisons
Comparison generates the data result, [operand A minus operand B]. The flags, however, are defined to indicate the comparison conditions rather than the flag conditions for subtraction. Signed INFs will be compared as expected. A NAN input to the comparison operation will cause the unordered flag result (INVALOP) and the production of an all-ones NAN. Even in FAST mode, the ALUs will accept denormals as inputs to the comparison operation. See "Less Than, Equal, Greater Than and Unordered" in the "Status Flag" section above for a complete discussion of these flags in comparison operations.

Conversions: Floating to Fixed
Conversions from floating-point to twos-complement integer (SFIXA/B and DFIXA/B) are considered "floating-point" operations, and all four rounding modes are available. If the operand *after rounding* overflows the destination format, OVRFLO will be set, and the results will be undefined. Thus, OVRFLO for fixed-point operations is treated exactly as it is for floating-point operations.

If the nonzero operand *before rounding* is of magnitude less than one, UNDFLO will be set in a conversion to integer. The magnitude of the result may be either one or zero, depending on the rounding mode. Conversion to integer is the only operation where UNDFLO depends on the *pre*-rounded result. The reason for this is that the infinitely precise result could be almost one integer unit away from the post-rounded result, potentially a large difference. We have chosen to flag underflow whenever the magnitude of the source operand is less than one, thereby alerting the user to a potentially significant loss of accuracy.

INEXO will be asserted if the conversion is inexact. NANs and INFs will both convert to an all-ones twos-complement integer with the sign bit preserved, either full-scale positive (for +NAN or +INF) or −1 (for −NAN or −INF). INVALOP will be asserted. See Tables XVI and XVII for illustrations of fixing single- and double-precision floating-point numbers.

Conversions: Fixed to Floating
All four rounding modes are also available for conversions from twos-complement integer to floating-point. For conversion to single-precision floating-point (SFLOATA/B), the numerical result will always be IEEE normals. The only flag ever set is INEXO. INEXO will be set if the source integer contains more than 24 bits of significance. "Significance" is defined as follows: For positive twos-complement integers, the number of significant bits is [(32 minus the number of leading zeros) minus the number of trailing zeros]. "Leading zeros" are the contiguous string of zeros starting from the most significant bit. "Trailing zeros" are the contiguous string of zeros starting from the least significant bit. For negative twos-complement integers, the number of significant bits is [(33 minus the number of leading ones) minus the number of trailing zeros].

For conversion from twos-complement integer to double-precision floating-point (DFLOATA/B), the numerical result will always be an IEEE normal. No flags will be set. Only even-numbered registers (A_0, A_2, B_0 or B_2) can be sources for the DFLOAT operation.

If you convert a fixed-point number in the B input to floating-point format and the fixed-point number has the same bit pattern as a floating-point NAN (i.e., Bits 23-30 are all ones and at least one of Bits 0-22 is a one), INEXO will be asserted. This occurs only in the ALU's DFLOATB and SFLOATB instructions, not DFLOATA or SFLOATA.

Conversions: Floating to Floating
For conversion from single-precision to double-precision (DOUBLEA/B), all single-precision normals and denormals will convert without exceptions. A single-precision NAN will convert to a double-precision all-ones NAN; the INVALOP flag will be set. Single-precision INF converts to double-precision INF; no flags are set. Single-precision ZERO converts to double-precision ZERO; no flags are set.

Conversions from double-precision to single-precision floating-point (SINGLEA/B) can cause exceptions because overflow, underflow and inexact status can result in mapping to the smaller destination format. See Table XVIII for illustrations. A double-precision NAN will convert to a single-precision all-ones NAN; the INVALOP flag will be set. DP INFs convert to SP INFs; no flags are set. Finite numbers greater in magnitude than single-precision NORM.MAX will result in an OVRFLO flag and SP INF or SP NORM.MAX , depending on the rounding mode, (see "Rounding − RND Controls" above). Nonzero, post-rounded operands whose magnitudes are between SP NORM.MAX and SP NORM.MIN inclusive will be SP NORMs. In IEEE mode, operands between SP DNRM.MAX and SP DNRM.MIN inclusive will be SP DNRMs, whereas in FAST mode, the result is ZERO with UNDFLO and INEXO flags set.

For both normals and denormals, INEXO will be asserted if the conversion from double-precision to single-precision floating-point is inexact. If the conversion to denormals is inexact, both INEXO and UNDFLO will be set, in accordance with the IEEE definition in terms of loss of accuracy when representing a denormal (see "Underflow" in "Status Flags" above). Post-rounded, nonzero numbers less than SP DNRM.MIN will convert to ZERO; UNDFLO and INEXO will be set. DP ZERO converts to SP ZERO without exception.

Pass
Pass instructions (SPASSA/B and DPASSA/B) pass all operands unmodified. Unlike the PASSA/B instructions, the floating-point pass instructions will cause INVALOP if a NAN is passed. The NAN will pass unmodified. INFs are passed without setting any flags. The absolute value controls can be used with the floating-

Sign	HB	f22 . . . f1	f0		Unbiased Expnt	Source Name	Sign	i30	i29	i28	i27	i26	i25	i24	i23	i22 . . .	i7	i6	i5	i4	i3	i2	i1	i0	Rounding Modes	Status Flags
0	1	X . . . X	X	2**	128	+NAN	0	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	all	INVALOP
0	1	0 . . . 0	0	2**	128	+INF	0	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	all	INVALOP
0	1	0 . . . 0	0	2**	31		U*	U	U	U	U	U	U	U	U	U . . .	U	U	U	U	U	U	U	U	all	OVRFLO
0	1	1 . . . 1	1	2**	30		0	1	1	1	1	1	1	1	1	1 . . .	1	0	0	0	0	0	0	0	all	
0	1	1 . . . 1	1	2**	23		0	0	0	0	0	0	0	0	1	1 . . .	1	1	1	1	1	1	1	1	all	
0	1	0 . . . 0	0	2**	23		0	0	0	0	0	0	0	0	1	0 . . .	0	0	0	0	0	0	0	0	all	
0	1	1 . . . 1	1	2**	22		0	0	0	0	0	0	0	0	1	0 . . .	0	0	0	0	0	0	0	0	RN,RP	INEXO
0	1	1 . . . 1	1	2**	22		0	0	0	0	0	0	0	0	0	1 . . .	1	1	1	1	1	1	1	1	RZ,RM	INEXO
0	1	0 . . . 0	0	2**	0	one	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	1	all	
0	1	1 . . . 1	1	2**	−1	one − 1LSB	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	1	RN, RP	UNDFLO,INEXO
0	1	1 . . . 1	1	2**	−1	one − 1LSB	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RZ,RM	UNDFLO,INEXO
0	1	0 . . . 0	1	2**	−1	1/2 + 1LSB	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	1	RN,RP	UNDFLO,INEXO
0	1	0 . . . 0	1	2**	−1	1/2 + 1LSB	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RZ,RM	UNDFLO,INEXO
0	1	0 . . . 0	0	2**	−1	1/2	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	1	RP	UNDFLO,INEXO
0	1	0 . . . 0	0	2**	−1	1/2	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RM,RN,RZ	UNDFLO,INEXO
0	1	0 . . . 0	0	2**	−126	+NORM.MIN	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	1	RP	UNDFLO,INEXO
0	1	0 . . . 0	0	2**	−126	+NORM.MIN	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RM,RN,RZ	UNDFLO,INEXO
0	0	0 . . . 0	1	2**	−126	+DENORM.MIN	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	1	RP	UNDFLO,INEXO
0	0	0 . . . 0	1	2**	−126	+DENORM.MIN	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RM,RN,RZ	UNDFLO,INEXO
0	0	0 . . . 0	0		0	+ZERO	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	all	
1	0	0 . . . 0	1	2**	−126	−DENORM.MIN	1	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	RM	UNDFLO,INEXO
1	0	0 . . . 0	1	2**	−126	−DENORM.MIN	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RP,RN,RZ	UNDFLO,INEXO
1	1	0 . . . 0	0	2**	−126	−NORM.MIN	1	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	RM	UNDFLO,INEXO
1	1	0 . . . 0	0	2**	−126	−NORM.MIN	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RP,RN,RZ	UNDFLO,INEXO
1	1	0 . . . 0	0	2**	−1	−1/2	1	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	RM	UNDFLO,INEXO
1	1	0 . . . 0	0	2**	−1	−1/2	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RP,RN,RZ	UNDFLO,INEXO
1	1	0 . . . 0	1	2**	−1	−1/2 − 1LSB	1	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	RM,RN	UNDFLO,INEXO
1	1	0 . . . 0	1	2**	−1	−1/2 − 1LSB	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RP,RZ	UNDFLO,INEXO
1	1	1 . . . 1	1	2**	−1	−one + 1LSB	1	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	RM,RN	UNDFLO,INEXO
1	1	1 . . . 1	1	2**	−1	−one + 1LSB	0	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	RP,RZ	UNDFLO,INEXO
1	1	0 . . . 0	0	2**	0	−one	1	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	all	
1	1	1 . . . 1	1	2**	22		1	1	1	1	1	1	1	1	1	0 . . .	0	0	0	0	0	0	0	0	RM,RN	INEXO
1	1	1 . . . 1	1	2**	22		1	1	1	1	1	1	1	1	1	0 . . .	0	0	0	0	0	0	0	1	RP,RZ	INEXO
1	1	0 . . . 0	0	2**	23		1	1	1	1	1	1	1	1	1	0 . . .	0	0	0	0	0	0	0	0	all	
1	1	1 . . . 1	1	2**	23		1	1	1	1	1	1	1	1	1	0 . . .	0	0	0	0	0	0	0	1	all	
1	1	1 . . . 1	1	2**	30		1	0	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	1	all	
1	1	0 . . . 0	0	2**	31		1	1	0	0	0	0	0	0	0	0 . . .	0	0	0	0	0	0	0	0	all	
1	1	0 . . . 0	1	2**	31		U	U	U	U	U	U	U	U	U	U . . .	U	U	U	U	U	U	U	U	all	
1	1	0 . . . 0	0	2**	128	−INF	1	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	all	INVALOP
1	1	X . . . X	X	2**	128	−NAN	1	1	1	1	1	1	1	1	1	1 . . .	1	1	1	1	1	1	1	1	all	INVALOP

*"U" denotes an undefined result.

Table XVI. Conversion of 32-Bit Single-Precision Floating-Point to 32-Bit Twos-Complement Integer

Sign	HB	f51 . . . f22	f21	f20	f19 . . . f1	f0		Unbiased Expnt	Source Name	Sign	i30 . . . i1	i0	Rounding Modes	Status Flags
0	1	X . . . X	X	X	X . . . X	X	2**	1024	+NAN	0	1 . . . 1	1	all	INVALOP
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	1024	+INF	0	1 . . . 1	1	all	INVALOP
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	31		U*	U . . . U	U	all	OVRFLO
0	1	1 . . . 1	1	1	1 . . . 1	1	2**	30		U	U . . . U	U	RP,RN	OVRFLO,INEXO
0	1	1 . . . 1	1	1	1 . . . 1	1	2**	30		0	1 . . . 1	1	RZ,RM	INEXO
0	1	1 . . . 1	1	0	0 . . . 0	0	2**	30		U	U . . . U	U	RP,RN	OVRFLO,INEXO
0	1	1 . . . 1	1	0	0 . . . 0	0	2**	30		0	1 . . . 1	1	RZ,RM	INEXO
0	1	1 . . . 1	0	1	1 . . . 1	1	2**	30		U	U . . . U	U	RP	OVRFLO,INEXO
0	1	1 . . . 1	0	1	1 . . . 1	1	2**	30		0	1 . . . 1	1	RM,RN,RZ	INEXO
0	1	1 . . . 1	0	0	0 . . . 0	1	2**	30		U	U . . . U	U	RP	OVRFLO,INEXO
0	1	1 . . . 1	0	0	0 . . . 0	1	2**	30		0	1 . . . 1	1	RM,RN,RZ	INEXO
0	1	1 . . . 1	0	0	0 . . . 0	0	2**	30		0	1 . . . 1	1	all	
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	0	one	0	0 . . . 0	1	all	
0	1	1 . . . 1	1	1	1 . . . 1	1	2**	−1	one − 1LSB	0	0 . . . 0	1	RN,RP	UNDFLO,INEXO
0	1	1 . . . 1	1	1	1 . . . 1	1	2**	−1	one − 1LSB	0	0 . . . 0	0	RZ,RM	UNDFLO,INEXO
0	1	0 . . . 0	0	0	0 . . . 0	1	2**	−1	1/2 + 1LSB	0	0 . . . 0	1	RN,RP	UNDFLO,INEXO
0	1	0 . . . 0	0	0	0 . . . 0	1	2**	−1	1/2 + 1LSB	0	0 . . . 0	0	RZ,RM	UNDFLO,INEXO
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1	1/2	0	0 . . . 0	1	RP	UNDFLO,INEXO
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1	1/2	0	0 . . . 0	0	RM,RN,RZ	UNDFLO,INEXO
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1022	+NORM.MIN	0	0 . . . 0	1	RP	UNDFLO,INEXO
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1022	+NORM.MIN	0	0 . . . 0	0	RM,RN,RZ	UNDFLO,INEXO
0	0	0 . . . 0	0	0	0 . . . 0	1	2**	−1022	+DENORM.MIN	0	0 . . . 0	1	RP	UNDFLO,INEXO
0	0	0 . . . 0	0	0	0 . . . 0	1	2**	−1022	+DENORM.MIN	0	0 . . . 0	0	RM,RN,RZ	UNDFLO,INEXO
0	0	0 . . . 0	0	0	0 . . . 0	0		0	+ZERO	0	0 . . . 0	0	all	
1	0	0 . . . 0	0	0	0 . . . 0	1	2**	−1022	−DENORM.MIN	1	1 . . . 1	1	RM	UNDFLO,INEXO
1	0	0 . . . 0	0	0	0 . . . 0	1	2**	−1022	−DENORM.MIN	0	0 . . . 0	0	RP,RN,RZ	UNDFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1022	−NORM.MIN	1	1 . . . 1	1	RM	UNDFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1022	−NORM.MIN	0	0 . . . 0	0	RP,RN,RZ	UNDFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1	−1/2	1	1 . . . 1	1	RM	UNDFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1	−1/2	0	0 . . . 0	0	RP,RN,RZ	UNDFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	1	2**	−1	−1/2 − 1LSB	1	1 . . . 1	1	RM,RN	UNDFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	1	2**	−1	−1/2 − 1LSB	0	0 . . . 0	0	RP,RZ	UNDFLO,INEXO
1	1	1 . . . 1	1	1	1 . . . 1	1	2**	−1	−one + 1LSB	1	1 . . . 1	1	RM,RN	UNDFLO,INEXO
1	1	1 . . . 1	1	1	1 . . . 1	1	2**	−1	−one + 1LSB	0	0 . . . 0	0	RP,RZ	UNDFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	0	−one	1	1 . . . 1	1	all	
1	1	1 . . . 1	0	0	0 . . . 0	0	2**	30		1	0 . . . 0	1	all	
1	1	1 . . . 1	0	0	0 . . . 0	1	2**	30		1	0 . . . 0	0	RM	INEXO
1	1	1 . . . 1	0	0	0 . . . 0	1	2**	30		1	0 . . . 0	1	RP,RN,RZ	INEXO
1	1	1 . . . 1	0	1	1 . . . 1	1	2**	30		1	0 . . . 0	0	RM	INEXO
1	1	1 . . . 1	0	1	1 . . . 1	1	2**	30		1	0 . . . 0	1	RP,RN,RZ	INEXO
1	1	1 . . . 1	1	0	0 . . . 0	0	2**	30		1	0 . . . 0	0	RM,RN	INEXO
1	1	1 . . . 1	1	0	0 . . . 0	0	2**	30		1	0 . . . 0	1	RP,RZ	INEXO
1	1	1 . . . 1	1	1	1 . . . 1	1	2**	30		1	0 . . . 0	0	RM,RN	INEXO
1	1	1 . . . 1	1	1	1 . . . 1	1	2**	30		1	0 . . . 0	1	RP,RZ	INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	31		1	0 . . . 0	0	all	
1	1	0 . . . 0	0	0	0 . . . 0	1	2**	31		1	0 . . . 0	0	RP,RN,RZ	INEXO
1	1	0 . . . 0	0	0	0 . . . 0	1	2**	31		U	U . . . U	U	RM	OVRFLO,INEXO
1	1	0 . . . 0	0	0	1 0 . . . 0	0	2**	31		1	0 . . . 0	0	RP,RZ	INEXO
1	1	0 . . . 0	0	0	1 0 . . . 0	0	2**	31		U	U . . . U	U	RM,RN	OVRFLO,INEXO
1	1	0 . . . 0	0	0	1 1 . . . 1	1	2**	31		1	0 . . . 0	0	RP,RZ	INEXO
1	1	0 . . . 0	0	0	1 1 . . . 1	1	2**	31		U	U . . . U	U	RM,RN	OVRFLO,INEXO
1	1	0 . . . 0	1	0	0 0 . . . 0	0	2**	31		U	U . . . U	U	all	OVRFLO
1	1	0 . . . 0	1	0	0 0 . . . 0	1	2**	31		U	U . . . U	U	all	OVRFLO,INEXO
1	1	0 . . . 0	0	0	0 0 . . . 0	0	2**	32		U	U . . . U	U	all	OVRFLO
1	1	0 . . . 0	0	0	0 0 . . . 0	0	2**	1024	−INF	1	1 . . . 1	1	all	INVALOP
1	1	X . . . X	X	X	X . . . X	X	2**	1024	−NAN	1	1 . . . 1	1	all	INVALOP

*"U" denotes an undefined result.

NOTE: Heavy line indicates rounding boundary in source.

Table XVII. Conversion of 64-Bit Double-Precision Floating-Point to 32-Bit Twos-Complement Integer

Sign	HB	f51 . . . f30	f29	f28	f27 . . . f1	f0		Unbiased Expnt	Source Name	Sign	HB	f22 . . . f1	f0		Unbiased Expnt	Result Name	Rounding Modes	Status Flags
0	1	X . . . X	X	X	X . . . X	X	2**	1024	+NAN	0	1	1 . . . 1	1	2**	128	+NAN	all	INVALOP
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	1024	+INF	0	1	0 . . . 0	0	2**	128	+INF	all	
0	1	1 . . . 1	1	1	1 . . . 1	1	2**	1023	+NORM.MAX	0	1	0 . . . 0	0	2**	128	+INF	RP,RN	OVRFLO,INEXO
0	1	1 . . . 1	1	1	1 . . . 1	1	2**	1023	+NORM.MAX	0	1	1 . . . 1	1	2**	127	+NORM.MAX	RZ,RM	OVRFLO,INEXO
0	1	1 . . . 1	1	1	0 . . . 0	0	2**	127		0	1	0 . . . 0	0	2**	128	+INF	RP,RN	OVRFLO,INEXO
0	1	1 . . . 1	1	1	0 . . . 0	0	2**	127		0	1	1 . . . 1	1	2**	127	+NORM.MAX	RZ,RM	INEXO
0	1	1 . . . 1	1	0	0 . . . 0	1	2**	127		0	1	0 . . . 0	0	2**	128	+INF	RP	OVRFLO,INEXO
0	1	1 . . . 1	1	0	0 . . . 0	1	2**	127		0	1	1 . . . 1	1	2**	127	+NORM.MAX	RM,RN,RZ	INEXO
0	1	1 . . . 1	1	0	0 . . . 0	0	2**	127		0	1	1 . . . 1	1	2**	127	+NORM.MAX	all	
0	1	1 . . . 1	0	0	0 . . . 0	1	2**	127		0	1	1 . . . 1	1	2**	127	+NORM.MAX	RP	INEXO
0	1	1 . . . 1	0	0	0 . . . 0	1	2**	127		0	1	1 . . . 1	0	2**	127		RM,RN,RZ	INEXO
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	−126		0	1	0 . . . 0	0	2**	−126	+NORM.MIN	all	
0	1	1 . . . 1	1	1	1 . . . 1	1	2**	−127		0	0	1 . . . 1	1	2**	−126	+NORM.MIN	RP,RN	INEXO
0	1	1 . . . 1	1	1	1 . . . 1	1	2**	−127		0	0	1 . . . 1	1	2**	−126	+DNRM.MAX	RZ,RM	UNDFLO,INEXO
0	1	1 . . . 1	0	0	0 . . . 0	0	2**	−127		0	0	1 . . . 1	1	2**	−126	+DNRM.MAX	all	
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	−149		0	0	0 . . . 0	1	2**	−126	+DNRM.MIN	all	
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1022	+NORM.MIN	0	0	0 . . . 0	1	2**	−126	+DNRM.MIN	RP	UNDFLO, INEXO
0	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1022	+NORM.MIN	0	0	0 . . . 0	0	2**	−126	+ZERO	RM,RN,RZ	UNDFLO, INEXO
0	0	1 . . . 1	1	1	1 . . . 1	1	2**	−1022	+DNRM.MAX	0	0	0 . . . 0	1	2**	−126	+DNRM.MIN	RP	UNDFLO, INEXO
0	0	1 . . . 1	1	1	1 . . . 1	1	2**	−1022	+DNRM.MAX	0	0	0 . . . 0	0	2**	−126	+ZERO	RM,RN,RZ	UNDFLO, INEXO
0	0	0 . . . 0	0	0	0 . . . 0	1	2**	−1022	+DNRM.MIN	0	0	0 . . . 0	1	2**	−126	+DNRM.MIN	RP	UNDFLO, INEXO
0	0	0 . . . 0	0	0	0 . . . 0	1	2**	−1022	+DNRM.MIN	0	0	0 . . . 0	0	2**	−126	+ZERO	RM,RN,RZ	UNDFLO, INEXO
0	0	0 . . . 0	0	0	0 . . . 0	0		0	+ZERO	0	0	0 . . . 0	0		0	+ZERO	all	
1	0	0 . . . 0	0	0	0 . . . 0	0		0	−ZERO	1	0	0 . . . 0	0		0	−ZERO	all	
1	0	0 . . . 0	0	0	0 . . . 0	1	2**	−1022	−DNRM.MIN	1	0	0 . . . 0	1	2**	−126	−DNRM.MIN	RM	UNDFLO, INEXO
1	0	0 . . . 0	0	0	0 . . . 0	1	2**	−1022	−DNRM.MIN	1	0	0 . . . 0	0	2**	−126	−ZERO	RP,RN,RZ	UNDFLO, INEXO
1	0	1 . . . 1	1	1	1 . . . 1	1	2**	−1022	−DNRM.MAX	1	0	0 . . . 0	1	2**	−126	−DNRM.MIN	RM	UNDFLO, INEXO
1	0	1 . . . 1	1	1	1 . . . 1	1	2**	−1022	−DNRM.MAX	1	0	0 . . . 0	0	2**	−126	−ZERO	RP,RN,RZ	UNDFLO, INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1022	−NORM.MIN	1	0	0 . . . 0	1	2**	−126	−DNRM.MIN	RM	UNDFLO, INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	−1022	−NORM.MIN	1	0	0 . . . 0	0	2**	−126	−ZERO	RP,RN,RZ	UNDFLO, INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	−149		1	0	0 . . . 0	1	2**	−126	−DNRM.MIN	all	
1	1	1 . . . 1	0	0	0 . . . 0	0	2**	−127		1	0	1 . . . 1	1	2**	−126	−DNRM.MAX	all	
1	1	1 . . . 1	1	1	1 . . . 1	1	2**	−127		1	0	1 . . . 1	1	2**	−126	−NORM.MIN	RM,RN	INEXO
1	1	1 . . . 1	1	1	1 . . . 1	1	2**	−127		1	0	1 . . . 1	1	2**	−126	−DNRM.MAX	RP,RZ	UNDFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	−126		1	1	0 . . . 0	0	2**	−126	−NORM.MIN	all	
1	1	1 . . . 1	0	0	0 . . . 0	1	2**	127		1	1	1 . . . 1	1	2**	127	−NORM.MAX	RM	INEXO
1	1	1 . . . 1	0	0	0 . . . 0	1	2**	127		1	1	1 . . . 1	0	2**	127		RP,RN,RZ	INEXO
1	1	1 . . . 1	1	0	0 . . . 0	0	2**	127		1	1	1 . . . 1	1	2**	127	−NORM.MAX	all	
1	1	1 . . . 1	1	0	0 . . . 0	1	2**	127		1	1	0 . . . 0	0	2**	128	−INF	RM	OVRFLO,INEXO
1	1	1 . . . 1	1	0	0 . . . 0	1	2**	127		1	1	1 . . . 1	1	2**	127	−NORM.MAX	RP,RN,RZ	INEXO
1	1	1 . . . 1	1	1	0 . . . 0	0	2**	127		1	1	0 . . . 0	0	2**	128	−INF	RM,RN	OVRFLO,INEXO
1	1	1 . . . 1	1	1	0 . . . 0	0	2**	127		1	1	1 . . . 1	1	2**	127	−NORM.MAX	RP,RZ	INEXO
1	1	1 . . . 1	1	1	1 . . . 1	1	2**	1023	−NORM.MAX	1	1	0 . . . 0	0	2**	128	−INF	RM,RN	OVRFLO,INEXO
1	1	1 . . . 1	1	1	1 . . . 1	1	2**	1023	−NORM.MAX	1	1	1 . . . 1	1	2**	127	−NORM.MAX	RP,RZ	OVRFLO,INEXO
1	1	0 . . . 0	0	0	0 . . . 0	0	2**	1024	−INF	1	1	0 . . . 0	0	2**	128	−INF	all	
1	1	X . . . X	X	X	X . . . X	X	2**	1024	−NAN	1	1	1 . . . 1	1	2**	128	−NAN	all	INVALOP

NOTE: Heavy line indicates rounding boundary in source.

Table XVIII. Conversion of 64-Bit Double-Precision Floating-Point to 32-Bit Single-Precision Floating-Point (IEEE Mode)

point pass instructions to reset the unmodified NAN's sign bit to zero.

Wrap

Wrap instructions (SWRAPA/B and DWRAPA/B) convert a denormal to a wrapped number readable by a multiplier/divider or the ADSP-3222 ALU in division and square root operations. Since the wrapped format has an additional bit of precision (the hidden bit), all wrapping is exact. If the operand is ZERO, then UNDFLO will be set. If the operand is neither a DNRM nor ZERO, INVALOP will be set.

Unwrap

Unwrapping instructions (SUNWRAPA/B and DUNWRAPA/B) convert a wrapped number to the IEEE denormal format. After rounding, the result can turn out to be NORM.MIN or ZERO.

WRAP.MAX, whose infinitely precise value is between NORM.MIN and DNRM.MAX, will round to NORM.MIN or DNRM.MAX, depending on rounding mode:

+WRAP.MAX → NORM.MIN (RN, RP modes)
+WRAP.MAX → DNRM.MAX (RZ, RM modes)
−WRAP.MAX → −NORM.MIN (RN, RM modes)
−WRAP.MAX → −DNRM.MAX (RZ, RP modes)

INEXO will always be set when unwrapping WRAP.MAX. If the unwrapping operation, after rounding, shifts all ones out of the DNRM destination format, ZERO will result. In unwrapping numbers, UNDFLO and INEXO are set if 1) unwrapping shifts ones out of the DNRM destination format or 2) INEXIN is asserted and no ones are shifted out.

The UNDFLO condition for unwrapping is based on the IEEE definition in terms of loss of accuracy when representing a denormal (see "Underflow" in "Status Flags" above). That is, UNDFLO will only be set in the ALU when the unbounded, post-rounded result cannot be expressed exactly in the destination denormal format. UNDFLO will always be set in conjunction with INEXO when unwrapping.

The ADSP-3222 determines inexactness by whether or not there was a loss of accuracy when unwrapping the operand supplied to the ALU and also whether the multiplication, division, or square root that generated the wrapped number caused a loss of accuracy. It determines this information by reading the INEXIN flag input to the ALU.

The INEXIN is essential to the unwrapping operation. The state of INEXIN input when wrapping should reflect the state of INEXO when the wrapped number was generated during multiplication, division, or square root. The ADSP-3222 uses this information to determine if the operation creating the wrapped number was inexact. When the ADSP-3222 unwraps a wrapped number, its INEXO will be asserted if *either* the originating operation *or* the unwrapping operation caused a loss of accuracy.

Copy Sign

The SSIGN and DSIGN operations copy the sign of the B operand to the A operand. The result is [sign B, exponent A, fraction A]. Rounding modes have no effect on this operation since the precision of the result is exactly that of the source, i.e., all "roundings" are exact. The only condition that generates a flag is a NAN as the A operand; INVALOP will be set. This instruction is useful for quadrant normalization of trigonometric functions. Trigonometric identities allow mapping an angle of interest to a quadrant for which lookup tables exist. SSIGN and DSIGN simplify this mapping. For example, sin (−37°) = − sin (37°). By looking up sin (37°) and transferring the sign of the angle (−37°, the B operand) to the value from the lookup table (0.60182, the A operand), the correct result is obtained (−0.60182).

Exponent Subtraction

Exponent subtraction (SXSUB and DXSUB) subtracts the exponent of the B operand from the A operand. The A operand is the destination format: [sign A, (expt A − expt B), fraction A]. INFs and NANs are valid inputs to the SXSUB/DXSUB operations; INVALOP is never asserted. If the unbounded result is greater than that of NORM.MAX, INF will be produced and OVRFLO will be set. If the unbounded result is less than that of NORM.MIN, ZERO will be produced and UNDFLO will be set.

Logical Downshift

The mantissa of a floating-point A operand (with hidden bit restored) can be downshifted logically to an unsigned-magnitude integer destination format using the SITRN and DITRN operations. (See Figures 25 and 26.) The source mantissa is treated as a right-justified unsigned integer. The unbiased (i.e., the "true" exponent after the bias has been subtracted) exponent of the B operand determines the amount of the downshift. The unbiased B exponent is interpreted as an unsigned number which indicates how many bit positions the mantissa should be downshifted. (A negative unbiased exponent will cause a very large downshift. The mantissa will be completely shifted out of range, and the result will be zero.) The result will be a left-zero-filled unsigned-magnitude integer. Like all fixed-point results, it will appear in the most significant bit positions of the output register.

Logical downshift is only defined for NORMs. Results from operands that are not normals are undefined. A NAN A-operand input to SITRN/DITRN will cause INVALOP and produce all-ones NANs of the same sign. Round-toward-Zero (RZ) must be specified for SITRN and DITRN. Otherwise, the result is undefined. If the shifted result *before* rounding is all zeros, UNDFLO will be set. (Actually, with RZ, the shifted result before rounding is the same as the shifted result after rounding.) If any bits are shifted out of the range of the destination format, INEXO will be set.

The logical downshift operations can be useful to generate table lookup addresses. In this application, the most significant mantissa bits would be used as table addresses. Because different B exponents can be applied to the same A mantissa, the same datum can be used to address multiple tables with differently sized address fields.

Division and Square Root

The ADSP-3222 ALU supports multicycle division (SDIV, 16 cycles, and DDIV, 30 cycles) and square root (SSQR, 29 cycles, and DSQR, 58 cycles) operations. (The ADSP-3212 performs faster division; the ADSP-3222 supports division for code-compatibility with the earlier ADSP-3221.) Tables XIX and XX illustrate the resultant data types and status conditions for division. Table XXI serves a similar role for square root. Neither operation can accept denormal inputs directly. The ADSP-3222 will process DNRMs if they are first wrapped to the wrapped format. Otherwise, DNRM inputs to division and square root operations will cause the simultaneous assertion of UNDFLO and INVALOP in IEEE mode. For divisions, INEXO HI indicates that the dividend is a DNRM; INEXO LO indicates that the divisor or both operands are DNRMs. In FAST mode, only INVALOP will be asserted.

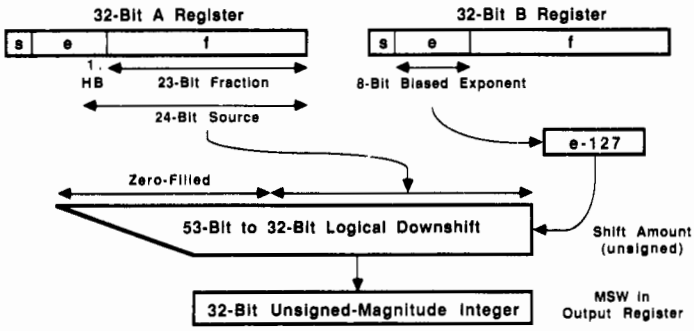


Figure 25. ADSP-3222 SITRN Instruction

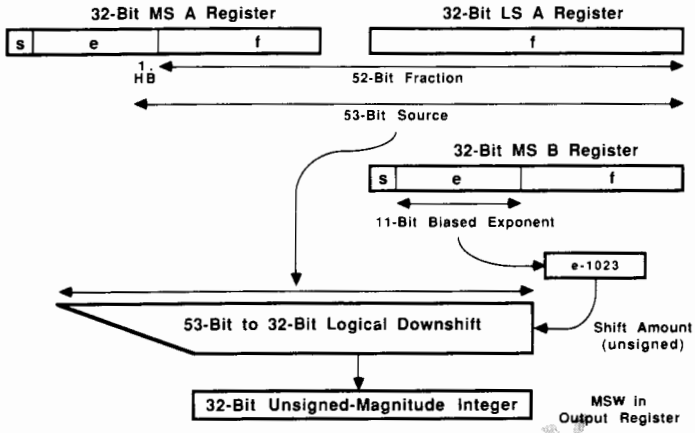


Figure 26. ADSP-3222 DITRN Instruction

A operand \ B operand	ZERO		DNRM		WRAP		NORM		INF		NAN	
	result	*status*	*result*	*status*	*result*	*status*	*result*	*status*	*result*	*status*	*result*	*status*
ZERO	NAN	INVALOP	ZERO		ZERO		ZERO		ZERO		NAN	INVALOP INEXO
DNRM	INF	OVRFLO& INVALOP	NAN	UNDFLO& INVALOP	NAN	UNDFLO INVALOP	NAN	UNDFLO INVALOP	ZERO		NAN	INVALOP INEXO
WRAP	INF	OVRFLO& INVALOP	NAN	UNDFLO& INVALOP	NORM		NORM WRAP UNRM	UNDFLO UNDFLO	ZERO		NAN	INVALOP INEXO
NORM	INF	OVRFLO& INVALOP	NAN	UNDFLO& INVALOP	INF,[1] NORM.MAX NORM	OVRFLO	INF,[1] NORM.MAX NORM WRAP UNRM	OVRFLO UNDFLO UNDFLO	ZERO		NAN	INVALOP INEXO
INF	INF		INF		INF		INF		NAN	INVALOP	NAN	INVALOP INEXO
NAN	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP INEXO

1. Either INF or NORM.MAX, depending on rounding mode. See "Round Controls."

Table XIX. ADSP-3222 Floating-Point Division (A÷B) (IEEE Mode)

A operand \ B operand	ZERO		DNRM		NORM		INF		NAN	
	result	*status*	*result*	*status*	*result*	*status*	*result*	*status*	*result*	*status*
ZERO	NAN	INVALOP	NAN	INVALOP	ZERO		ZERO		NAN	INVALOP INEXO
DNRM	NAN	INVALOP	NAN	INVALOP	ZERO		ZERO		NAN	INVALOP INEXO
NORM	INF	OVRFLO& INVALOP	INF,[1] NORM.MAX	OVRFLO& INVALOP	INF,NORM.MAX[1] NORM ZERO	OVRFLO UNDFLO	ZERO		NAN	INVALOP INEXO
INF	INF		INF		INF		NAN	INVALOP	NAN	INVALOP INEXO
NAN	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP	NAN	INVALOP INEXO

In FAST mode, WRAP inputs are illegal.

1. Either INF or NORM.MAX, depending on rounding mode. See "Round Controls."

Table XX. ADSP-3222 Floating-Point Division (A÷B) (FAST Mode)

Mode \ B operand	B<ZERO		±ZERO		+DNRM		+WRAP		+NORM		+INF		±NAN	
	result	status	result	status	result	status	result	status	result	status	result	status	result	status
IEEE	-NAN	INVALOP	±ZERO		+NAN	UNDFLO& INVALOP	NORM		NORM		+INF		±NAN	INVALOP
FAST	-NAN	INVALOP	±ZERO		+ZERO		NORM		NORM		+INF		±NAN	INVALOP

Table XXI. ADSP-3222 Floating-Point Square Root ($\sqrt{B}$)

The square root of any nonnegative normal or wrapped number will be an IEEE normal number. The square root of a negative number is an all-ones (−NAN). The square root of +INF is +INF without exception. The square root of a NAN is a same-signed all-ones NAN.

Division can produce wrappeds and unnormals; these must be passed back to the ALU for unwrapping. Note that division using the SDIV and DDIV instructions of the ALU can produce wrapped results that are slightly different than those produced by the multiplier/divider; this difference is explained in the *Special Flags for Unwrapping* section. When the results are unwrapped with the correct flag inputs, the same denormal number is produced.

INF dividends cause correctly signed INFs without flags except when the divisor is also an INF. Either ±INF divided by either ±INF or any NAN input will generate INVALOP and an all-ones NAN. For division operations, the sign of the NAN will be the exclusive OR of the signs of the dividend and the divisor.

Output Control – SHLP (REG), OEN (ASYN), MSWSEL (ASYN) and $\overline{\text{HOLD}}$ (ASYN)

All members of the ADSP-3212/ADSP-3222 chipset have a 64-bit output register. The output registers are clocked every cycle, except for multicycle operations (division and square root), when $\overline{\text{HOLD}}$ is LO, and when the ADSP-3222 is executing NOP. Output registers are clocked at the conclusion of multicycle operations and not before.

Results appear in the multiplier/divider's output register as follows:

Bit 63 ... 32	31 ... 0
SP FltgPt Product	not meaningful
DP FltgPt Most Significant Product	DP FltgPt Least Significant Product
FxdPt Most Significant Product	FxdPt Least Significant Product

Figure 27. ADSP-3212 Multiplier/Divider Output Registers

When the destination format from multiplication is single-precision floating-point, the fraction bits that are less than the least significant bit in the destination format are stored in the least significant half of the output register.

The multiplier/divider has a pipelined, registered fixed-point shift-left control, SHLP. When HI, SHLP will cause a one-bit left shift in the 64-bit product that appears in the multiplier/divider's output register. The least significant bit in the output register will be zero. See "32-Bit Fixed-Point Data Formats" above for more details of the effects of SHLP. SHLP has no effect on floating-point multiplications or divisions. Note that SHLP should be setup at the clock edge when the multiplication operands are read into the multiplier array.

Results appear in the ALU's output register as follows:

Bit 63 ... 32	31 ... 0
SP FltgPt Product	not meaningful
DP FltgPt Most Significant Product	DP FltgPt Least Significant Product
FxdPt Result	not meaningful

Figure 28. ADSP-3222 ALU Output Registers

Both chips have an asynchronous output enable control, OEN. When HI, outputs are enabled; when LO, output drivers at $DOUT_{31-0}$ are put into a high impedance state. Note that status flags are always driven off-chip, regardless of the state of OEN. See Figure T1 for the timing of OEN.

Both chips also have an asynchronous MSW select control, MSWSEL. When outputs are enabled and MSWSEL is HI, the most significant half (Bits 63 through 32) of the output register will be driven to the output port, $DOUT_{31-0}$. When outputs are enabled and MSWSEL is LO, the least significant half (Bits 31 through 0) of the output register will be driven to the output port, $DOUT_{31-0}$. The operation of MSWSEL is illustrated in all timing diagrams where 64-bit outputs are produced.

The ADSP-3212 Multiplier/Divider has a synchronous, active LO control, $\overline{\text{HOLD}}$, that prevents the output register from being updated. The IEEE/FAST pin on the ADSP-3222 ALU can be redefined to a $\overline{\text{HOLD}}$ pin by executing the HOLDEN instruction. (To change the pin back to IEEE/FAST, you must reset the ADSP-3222.) $\overline{\text{HOLD}}$ must be set up prior to the clock edge when the output register would have otherwise been updated. See Figure T3. For normal operations where the output register is updated, $\overline{\text{HOLD}}$ must be held HI.

TIMING

Timing diagrams are numbered T1 through T16. Three-state timing for DOUT is shown in T1. Output disable time, t_{DIS}, is measured from the time OEN reaches 1.5V to the time when all outputs have ceased driving. This is calculated by measuring the time, $t_{measured}$, from the same starting point to when the output voltages have changed by 0.5V toward +1.5V. From the tester capacitive loading, C_L, and the measured current, i_L, the decay time, t_{DECAY}, can be approximated to first order by:

$$t_{DECAY} = \frac{C_L \bullet 0.5V}{i_L}$$

from which

$$t_{DIS} = t_{measured} - t_{DECAY}$$

is calculated. Disable times are longest at the highest specified temperature.

The minimum output enable time, minimum t_{ENA}, is the earliest that outputs begin to drive. It is measured from the control signal OEN reaching 1.5V to the point at which the fastest outputs have changed by 0.1V from $V_{tristate}$ toward their final output voltages. Minimum enable times are shortest at the lowest specified temperature.

The maximum output enable time, maximum t_{ENA}, is also measured from OEN at 1.5V to the time when all outputs have reached TTL input levels (V_{OH} or V_{OL}). This could also be considered as "data valid." Maximum enable times are longest at the highest specified temperature.

Reset timing is shown in T2. $\overline{RESET}$ must be LO for at least t_{RS}. In addition, $\overline{RESET}$ must return HI at least t_{SU} before the first rising clock edge of operation. Hold timing is shown in T3. $\overline{HOLD}$ must go LO t_{HS} before the rising edge at which the output register is *not* updated. $\overline{HOLD}$ must also be held t_{HH} after the clock edge.

Figure T4 shows IPORT timing. IPORT must be set up at least t_{PS} before each data load (on a rising or a falling clock edge) and be held at least t_{PH} after the data load.

All data, registered and latched controls, and instructions shown in T5 through T16 must be set up t_{DS} before the rising edge and held t_{DH}. Data is shown loaded for minimum latency. Other sequencing options are possible and may be more convenient, depending on the system. These other options, however, require that data be loaded to the input registers earlier than as shown in these diagrams and not overwritten. See "Input Register Loading and Operand Storage" above for constraints on register loading and operand storage that must be observed.

The operation time, t_{OPD}, is the time required to advance the internal pipelines one stage. It reflects the pipelined throughput of the device for that operation. The latency, t_{LAD}, is the time it takes for the chip to produce a valid result at DOUT from valid data at its input ports. (Latency is the true measure of the internal speed of the chip.) Latency is referenced from data valid of the earliest required input to data valid of the first 32-bit output.

The asynchronous MSWSEL control's delay is t_{ENO}. The maximum specification for t_{ENO} is the delay which guarantees valid data. The minimum specification for t_{ENO} is the earliest time after the MSWSEL control is changed that data can change.

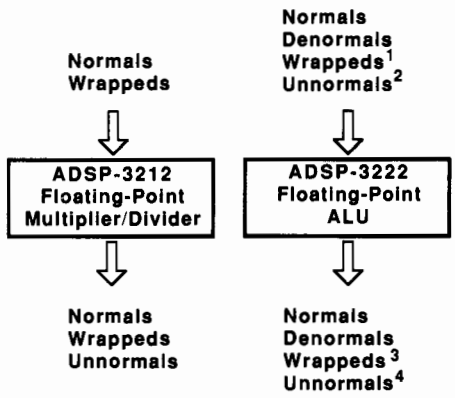
Status flags have a maximum output delay of t_{SO} referenced from the clock rising edge. All status flags except the multiplier/divider's DENORM are available in parallel with their associated output results. DENORM is available earlier to speed up recovery from a denormal input exception. Note that DENORM is LO except in the cycles indicated in Figures T7 through T10.

For all operations (Figures T7 through T16), a new operation can begin the cycle before output results and status flags (other than DENORM) results from the previous operation are driven off chip. This feature leads to improved pipeline throughput.

GRADUAL UNDERFLOW AND IEEE EXCEPTIONS

The data types that each chip operates on directly are shown in Figure 29.

Denormals are detected by the multiplier/divider when read into its processing circuitry. The ADSP-3212 will produce a flag output, DENORM, when one or both of the operands read into the array are denormals. The occurrence of DENORM should trigger exception processing. (See "Status Flags" above for a discussion of DENORM and its timing.) Controlling hardware must recover the denormal(s) that was input to a multiplier/divider and present it to an ALU for wrapping.



1. for unwrapping, division, and square root
2. for unwrapping only
3. from wrapping and division
4. from division

Figure 29. Data Types Directly Supported by the ADSP-3212/3222

The ADSP-3222 ALU will also detect denormals when read into internal circuitry for division or square root operations. The UNDFLO and INVALOP flags will both be asserted on the ADSP-3222 to signal the presence of a denormal input to these operations. INEXO will indicate whether the denormal input is the A operand or B operand. (See "Status Flags" above for a fuller discussion of denormal detection in the ADSP-3222.)

The ALU wraps denormals with its SWRAP or DWRAP instructions. Note from Tables II and IV that any denormal can be represented as a wrapped without loss of precision (hence triggers no exception flags in the ALU).

The wrapped equivalent from the ALU must now be passed to the multiplier/divider for multiplication or division or the ADSP-3222 ALU for division or square root. The controlling system must tell the multiplier/divider to interpret the wrapped input as wrapped by asserting WRAPA/B when it is read into the multiplier/divider's processing circuitry. For ALU division and square root, the controlling system must tell the ALU to interpret the wrapped operand A as wrapped by asserting INEXIN when it is read into the ALU's processing circuitry and to interpret the wrapped operand B as wrapped by asserting RNDCARI. The result of the multiplication or division can be a normal, a wrapped, or an unnormal. (See Tables V through VIII, XIX, and XX.) Square root on IEEE numbers only produces normals. (See Table XXI.) An underflowed result (wrapped or unnormal) from either multiplier/divider or ALU will be indicated by the UNDFLO flag and must be passed to the ALU for unwrapping. Note that the ALU and the multiplier/divider may produce slightly different wrapped results from the same division operation. When these results are unwrapped with the correct flag inputs, however, they produce the same number. See *Special Flags for Unwrapping* for an explanation of this difference.

For full conformance to the IEEE Standard, all wrapped and unnormal results must be unwrapped in an ALU (with the

SUNWRAP and DUNWRAP instructions) to an IEEE sanctioned destination format before any further operations on the data. If the result from unwrapping is a DNRM, then that data will have to be wrapped before it can be used in multiplication, division or square root operations.

The reason why WRAPs and UNRMs should always be unwrapped upon their production is that the wrapped and unnormal data formats often contain "spurious" accuracy, i.e., more precision than can be represented in the normal and denormal data formats. If WRAPs or UNRMs produced by the system were used directly as inputs to multiplication, division or square root operations, the results could be more accurate than, and hence incompatible with, the IEEE Standard.

When unwrapping, additional information about underflowed results must accompany their input to the ALU. See "Special Flags for Unwrapping" in "Status Flags" above for details of how INEXO and RNDCARO status flag outputs must be used with INEXIN and RNDCARI inputs.

A final point about conformance with IEEE Standard 754 pertains to NANs. The Standard distinguishes between signalling NANs and quiet NANs, based on differing values of the fraction field. Signalling NANs can represent uninitialized variables or specialized data values particular to an implementation. Quiet NANs provide diagnostic information resulting from invalid data or results. The ADSP-3212/ADSP-3222 generally produce all-ones outputs from invalid operations resulting from NAN inputs. So a system that implements operations on quiet and signalling NANs will have to modify the NAN output from these chips externally. See Section 6.2 of Standard 754-1985 for the details of these operations.

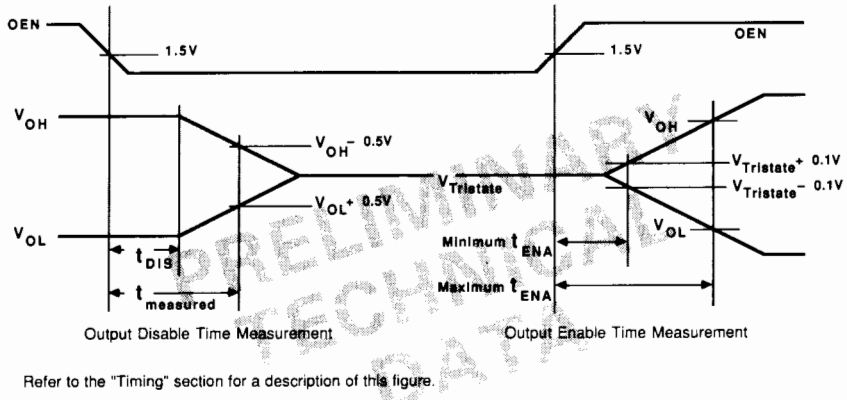


Figure T1. ADSP-3212/ADSP-3222 Three-State Enable and Disable Timing

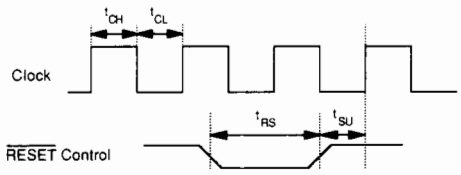


Figure T2. ADSP-3212/ADSP-3222 Reset Timing

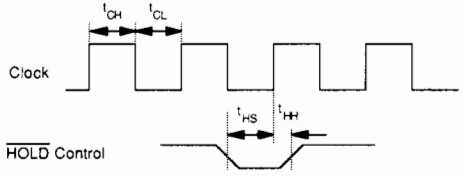


Figure T3. ADSP-3212/ADSP-3222 Output Register Hold Timing

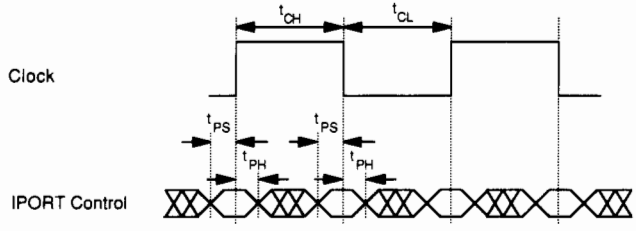


Figure T4. ADSP-3212/ADSP-3222 IPORT Timing

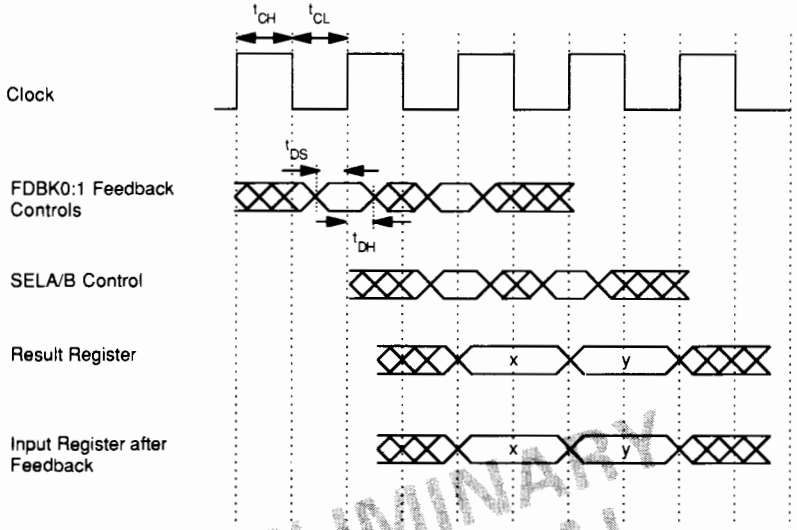


Figure T5. ADSP-3212/ADSP-3222 Feedback Timing

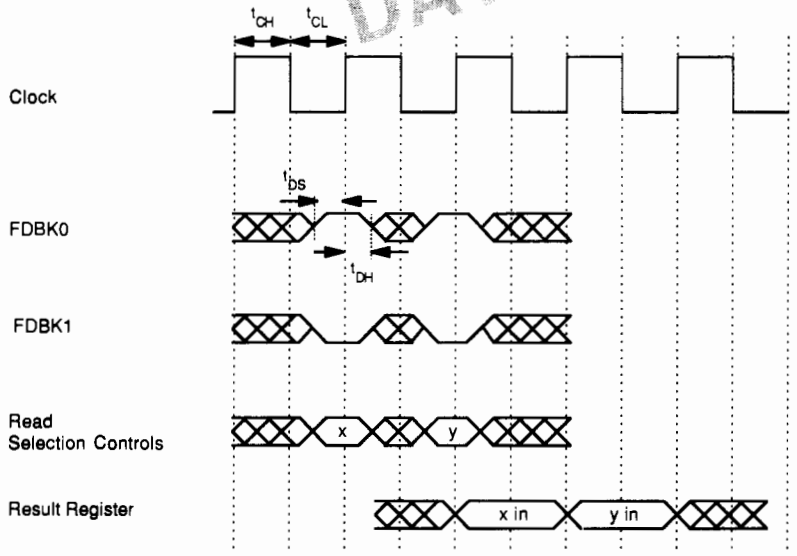
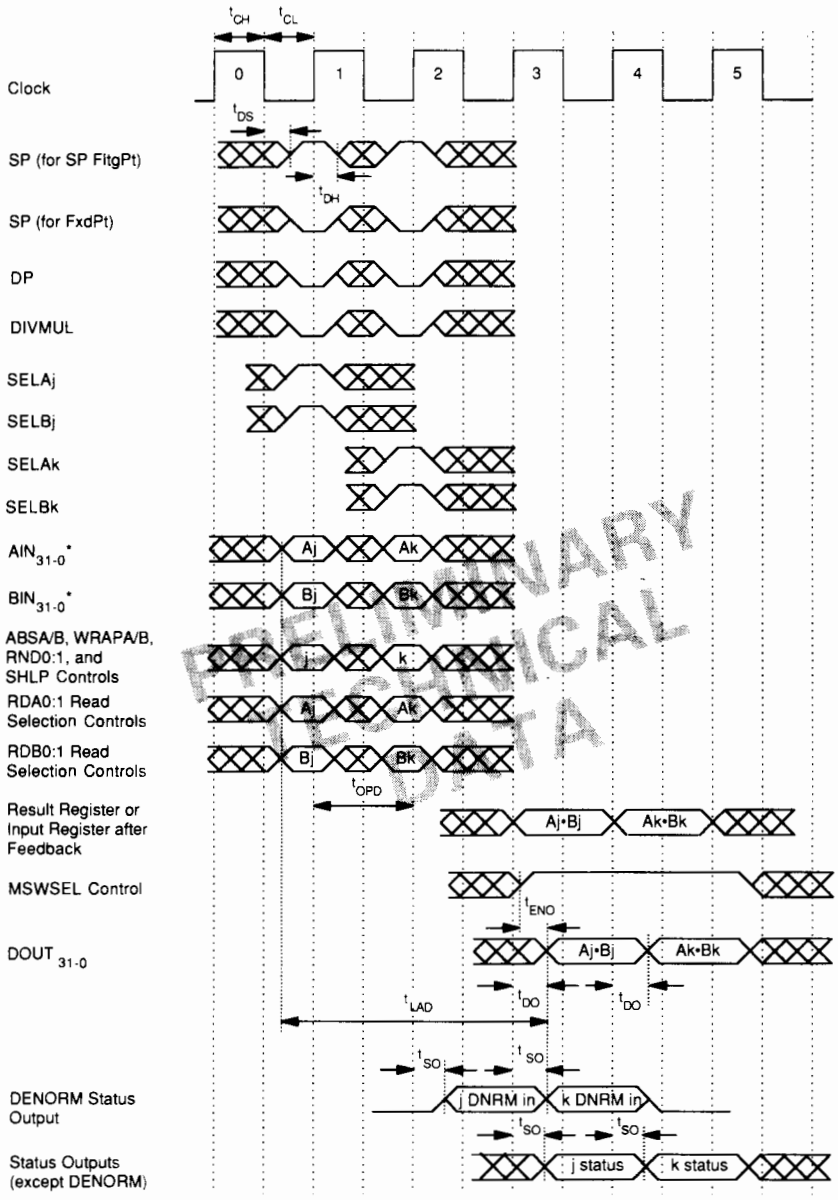
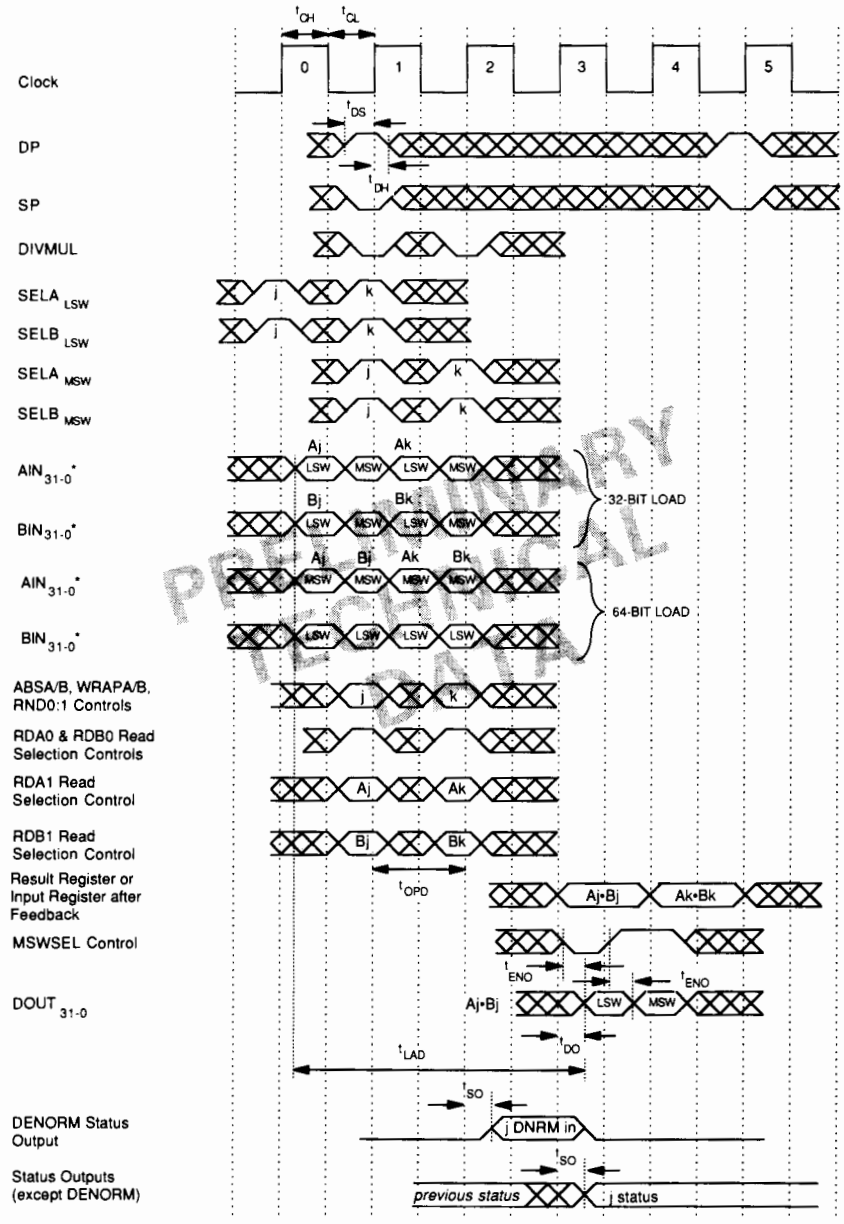


Figure T6. ADSP-3212/ADSP-3222 Feedforward Timing



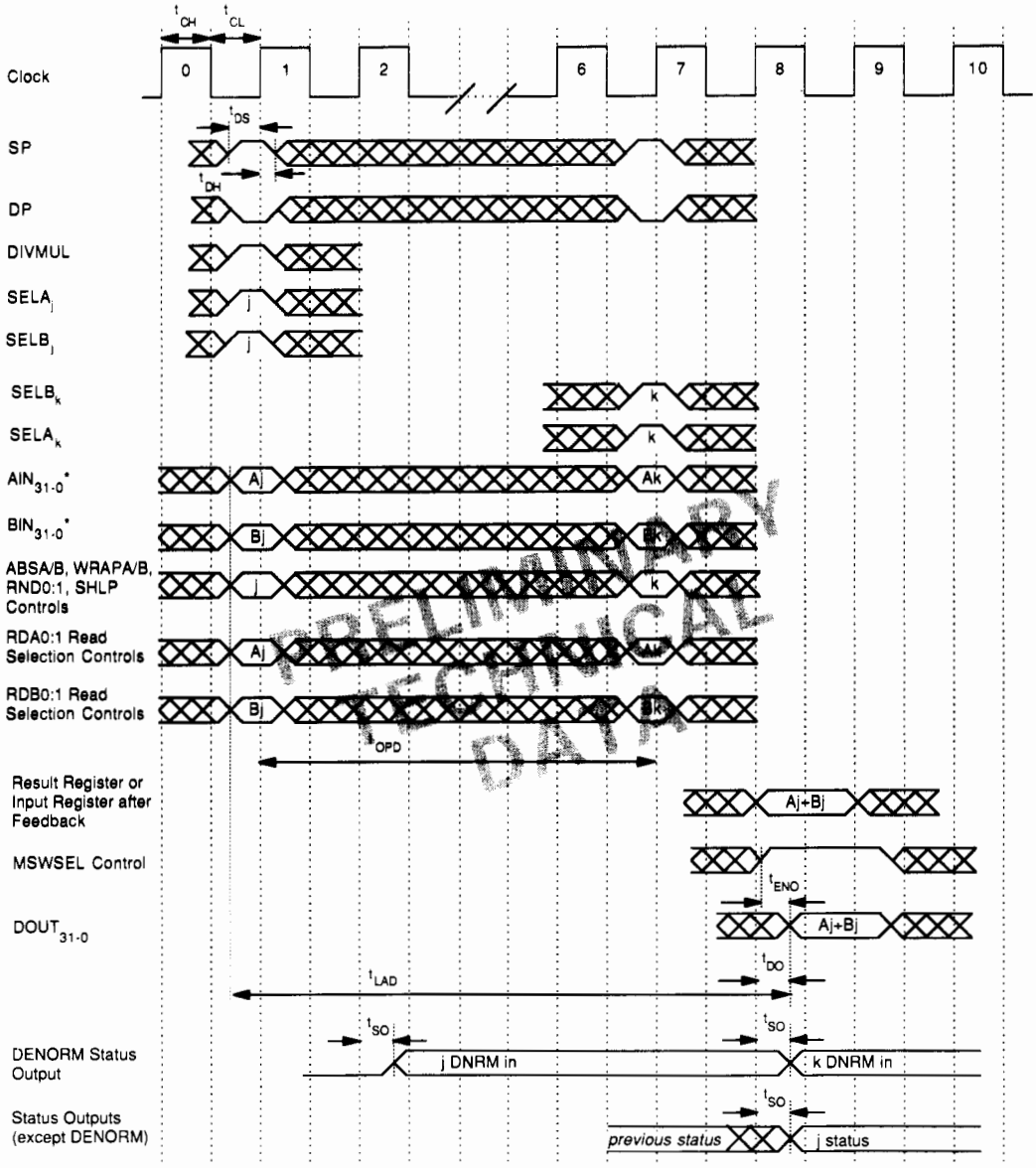
* See "Timing" section for additional sequencing options.

Figure T7. ADSP-3212 32-Bit Single-Precision Floating-Point and Fixed-Point Multiplication



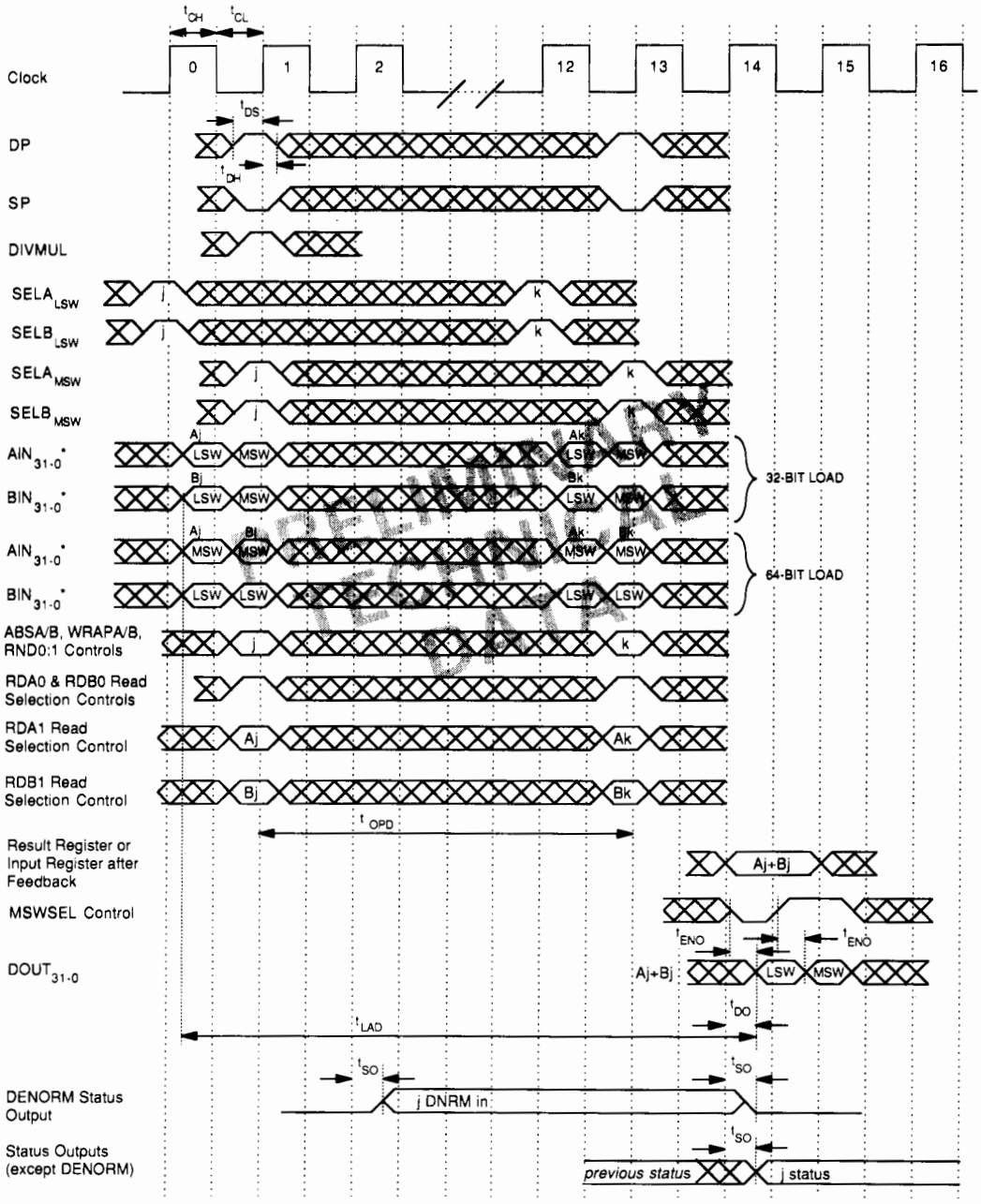
* See "Timing" section for additional sequencing options.

Figure T8. ADSP-3212 64-Bit Double-Precision Floating-Point Multiplication



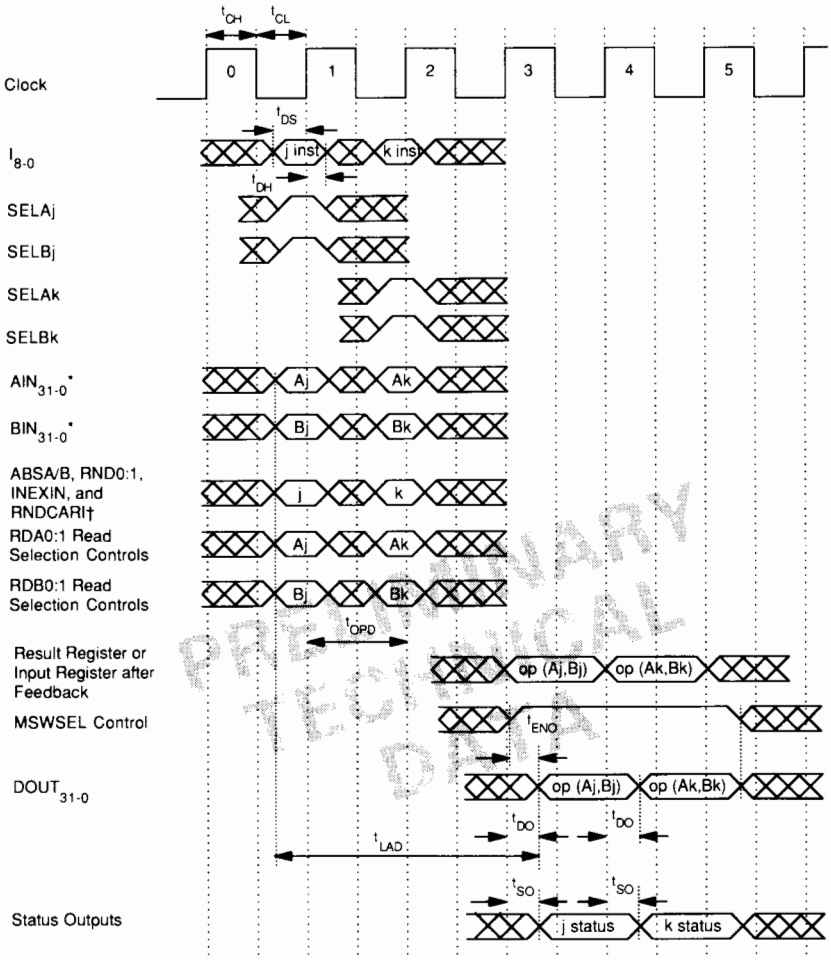
* See "Timing" section for additional sequencing options.

Figure T9. ADSP-3212 32-Bit Single-Precision Floating-Point Division



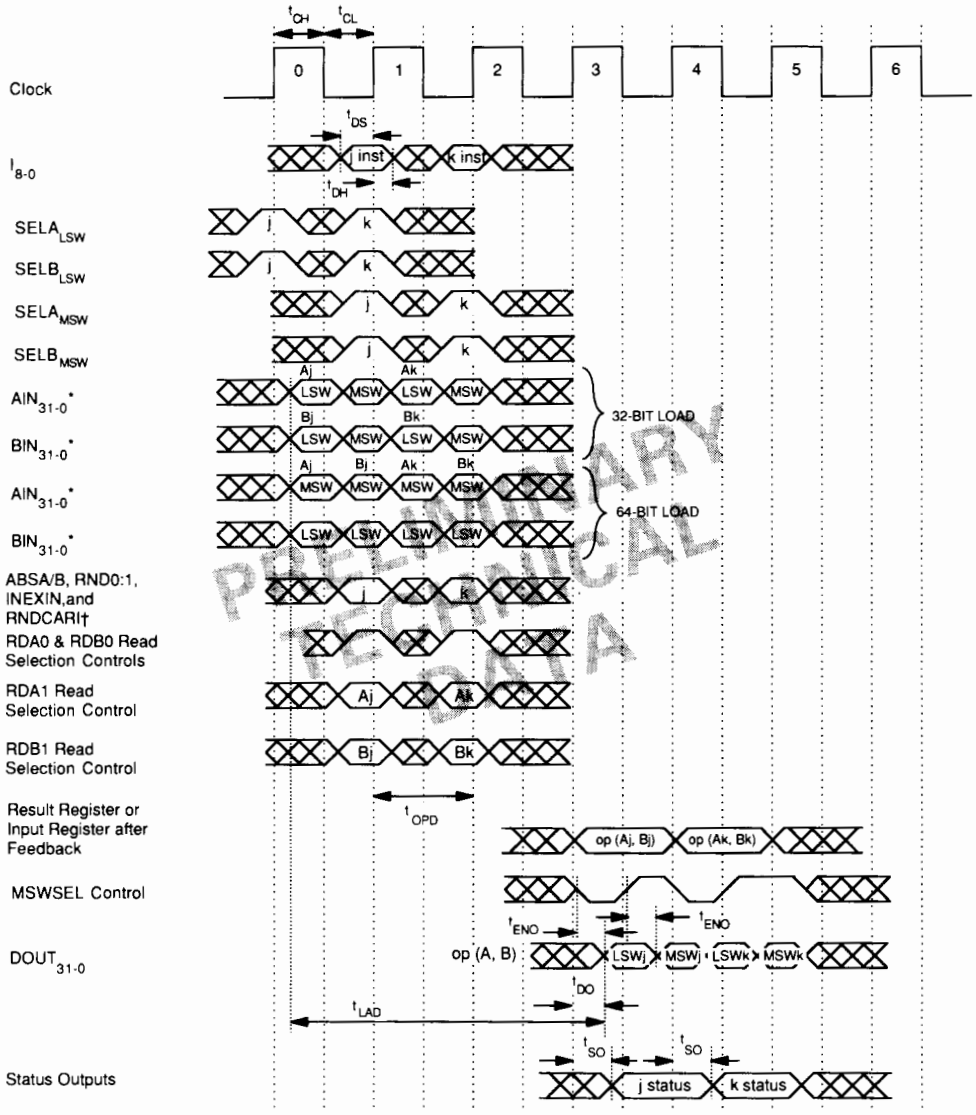
* See "Timing" section for additional sequencing options.

Figure T10. ADSP-3212 64-Bit Double-Precision Floating-Point Division



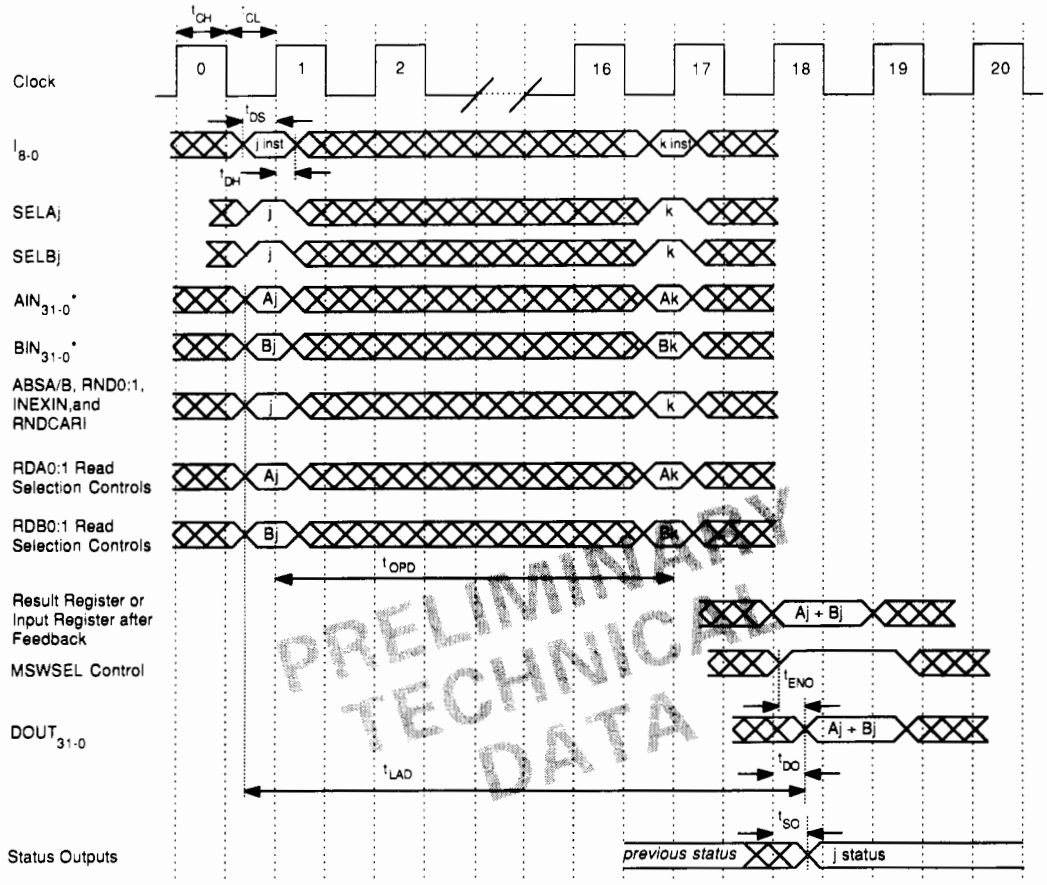
* See "Timing" section for additional sequencing options.
† RNDCARI and INEXIN should be LO except for unwrap, division, and square root operations.

Figure T11. ADSP-3222 32-Bit Single-Precision Floating-Point Logical and Fixed-Point ALU Operations



* See "Timing" section for additional sequencing options.
† RNDCARI and INEXIN should be LO except for unwrap, division, and square root operations.

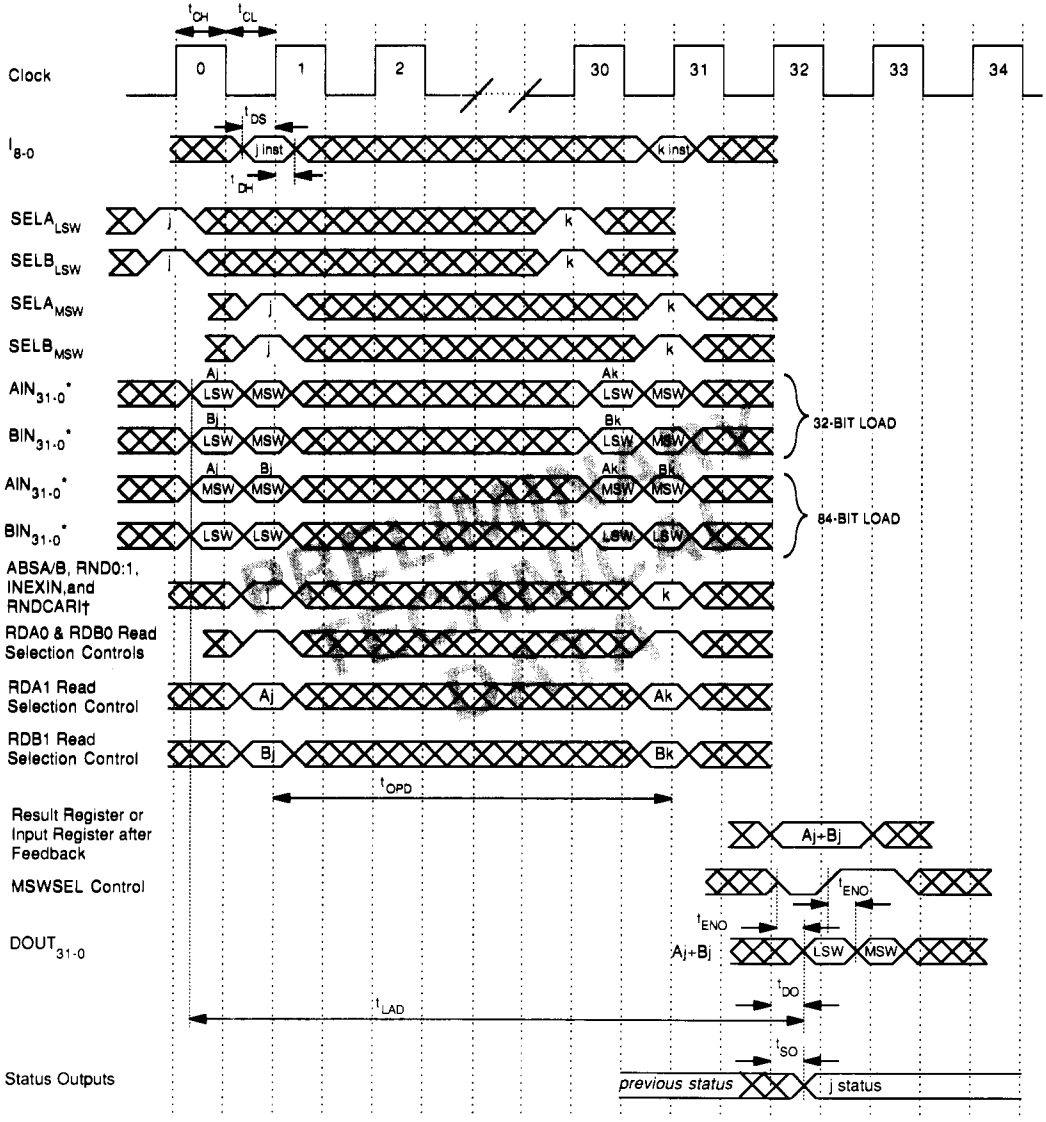
Figure T12. ADSP-3222 64-Bit Double-Precision Floating-Point ALU Operations



* See "Timing" section for additional sequencing options.
† RNDCARI and INEXIN should be LO except for unwrap, division, and square root operations.

Note: The ADSP-3212 performs faster division. See Figure T9.

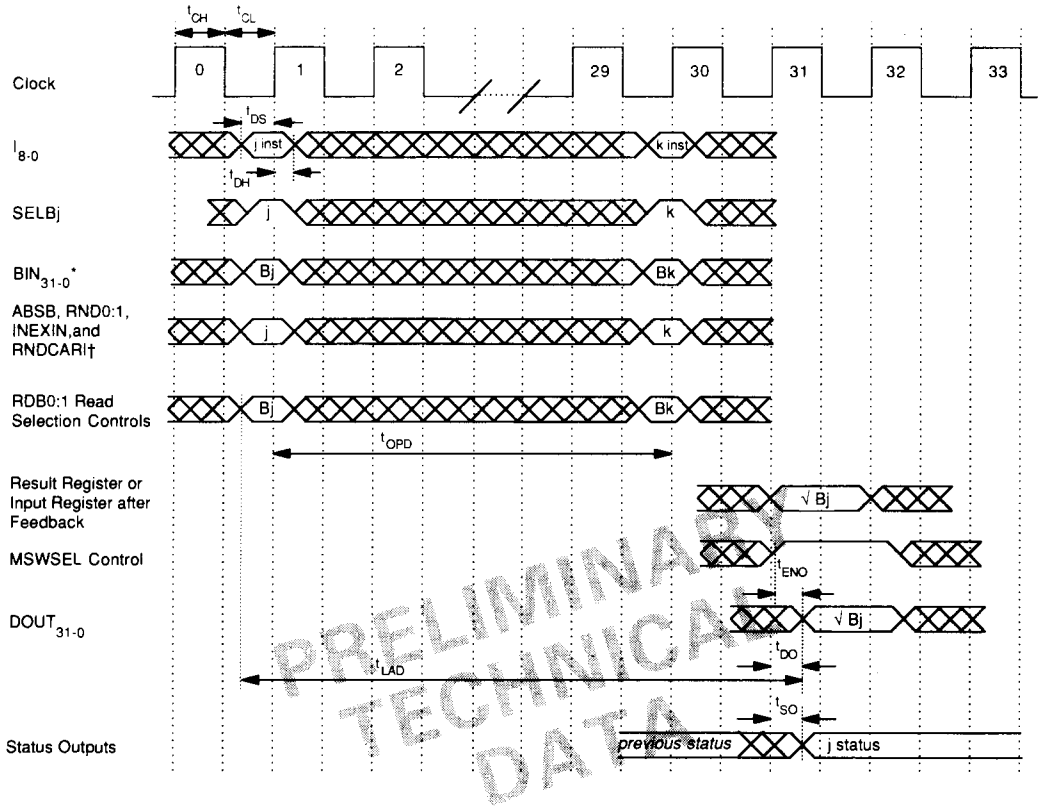
Figure T13. ADSP-3222 32-Bit Single-Precision Floating-Point Division



* See "Timing" section for additional sequencing options.
† RNDCARI and INEXIN should be LO except for unwrap, division, and square root operations.

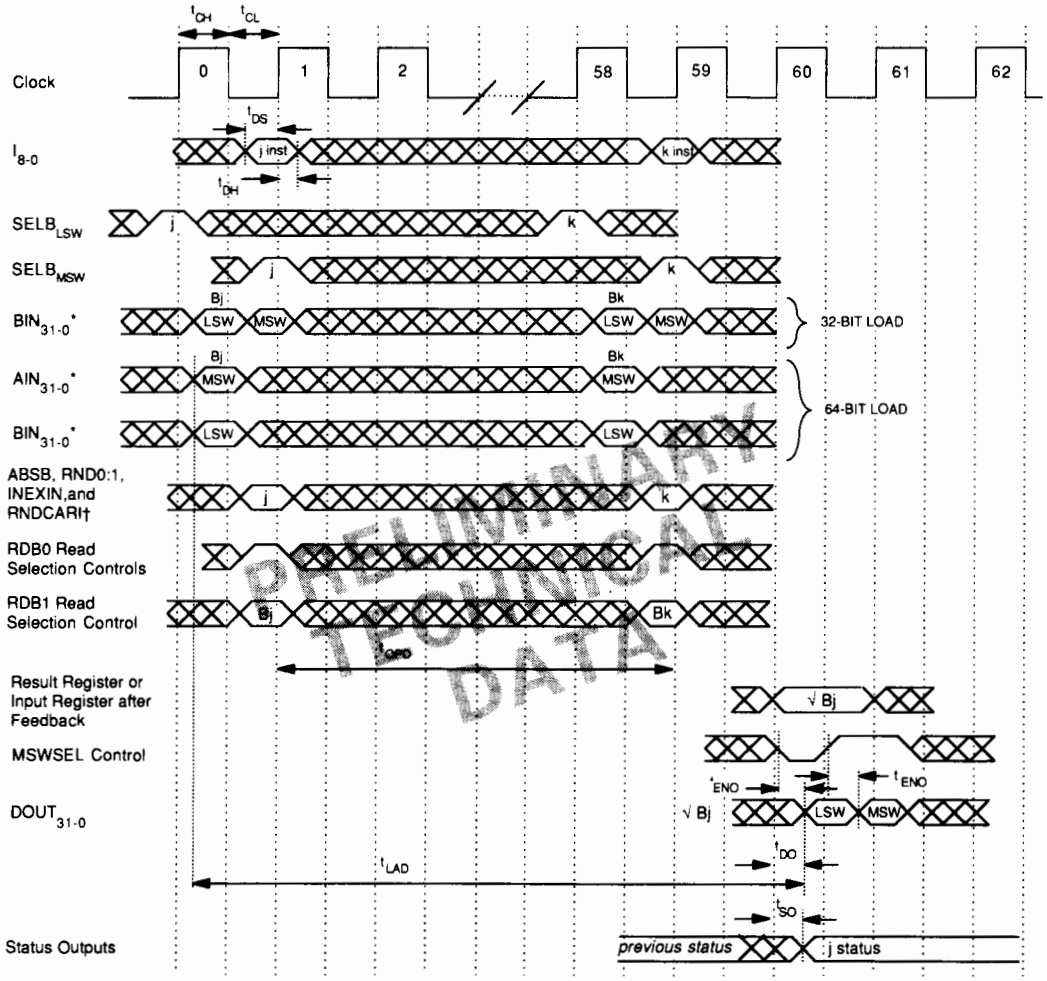
Note: The ADSP-3212 performs faster division. See Figure T10.

Figure T14. ADSP-3222 64-Bit Double-Precision Floating-Point Division



* See "Timing" section for additional sequencing options.
† RNDCARI and INEXIN should be LO except for unwrap, division, and square root operations.

Figure T15. ADSP-3222 32-Bit Single-Precision Floating-Point Square Root



* See "Timing" section for additional sequencing options.
† RNDCARI and INEXIN should be LO except for unwrap, division, and square root operations.

Figure T16. ADSP-3222 64-Bit Double-Precision Floating-Point Square Root

SPECIFICATIONS[1]

RECOMMENDED OPERATING CONDITIONS

Parameter		ADSP-3212/ADSP-3222 J, K Grades Min	J, K Grades Max	S, T Grades[2] Min	S, T Grades[2] Max	Unit
V_{DD}	Supply Voltage	4.75	5.25	4.5	5.5	V
T_{AMB}	Operating Temperature (ambient)	0	70	−55	+125	°C

ELECTRICAL CHARACTERISTICS

Parameter		ADSP-3212/ADSP-3222 J, K Grades Min	J, K Grades Max	S, T Grades Min	S, T Grades Max	Unit
V_{IH}	High Level Input Voltage @V_{DD}=max	2.0		2.0		V
V_{IHA}	High Level Input Voltage, CLK and Asynchronous Controls @V_{DD}=max	2.6		3.0		V
V_{IL}	Low Level Input Voltage @V_{DD}=min		0.8		0.8	V
V_{OH}	High Level Output Voltage @V_{DD}=min and I_{OH}=−1.0mA	2.4		2.4		V
V_{OL}	Low Level Output Voltage @V_{DD}=min and I_{OL}=4.0mA		0.5		0.6	V
I_{IH}	High Level Input Current, All Inputs @V_{DD}=max and V_{IN}=5.0V		10		10	μA
I_{IL}	Low Level Input Current, All Inputs @V_{DD}=max and V_{IN}=0.0V		10		10	μA
I_{OZ}	Three-State Leakage Current @V_{DD}=max; High Z; V_{IN}=0V or max		50		50	μA
I_{DD}	Supply Current @max clock rate; TTL inputs		200		250	mA
I_{DD}	Supply Current-Quiescent All V_{IN}=2.4V		50		60	mA

NOTES
[1]All min and max specifications are over power supply and temperature ranges indicated.
[2]S and T grade parts are available processed and tested in accordance with MIL-STD-883, Class B. The processing and test methods used for S/883B and T/883B versions of the ADSP-3212/ADSP-3222 can be found in Analog Devices' *Military Products Databook*. Regular S and T grade parts are tested at +125°C.
[3]Input levels are GND and +3.0V. Rise times are 5ns max. Input timing reference levels and output reference levels are 1.5V, except for (1) t_{ENA} and t_{DIS} which are as indicated in Figure T1 and (2) t_{DS} and t_{DH} which are measured from clock V_{IH} or V_{IL} crossing points.

Specifications subject to change without notice.

SWITCHING CHARACTERISTICS[3]

Parameter		ADSP-3212/ADSP-3222 J Grade 0 to +70°C Min	J Grade 0 to +70°C Max	K Grade 0 to +70°C Min	K Grade 0 to +70°C Max	S Grade −55°C to +125°C Min	S Grade −55°C to +125°C Max	T Grade −55°C to +125°C Min	T Grade −55°C to +125°C Max	Unit
t_{CY}	Clock Period	60		50				58		ns
t_{CL}	Clock LO			20				23		ns
t_{CH}	Clock HI			20				23		ns
t_{DS}	Data Setup	12		7				8		ns
t_{CS}	Control Setup			10				12		ns
t_{DH}	Data and Control Hold	3		3				3		ns
t_{DO}	Data Output Delay		25		18				21	ns
t_{SO}	Status Output Delay		25		18				21	ns
t_{ENO}	MSWSEL-to-Data Delay		25		18				21	ns
t_{DIS}	Three-State Disable Delay				12				14	ns
t_{ENA}	Three-State Enable Delay			1	18			1	21	ns
t_{SU}	$\overline{\text{RESET}}$ Setup			5				6		ns
t_{RS}	$\overline{\text{RESET}}$ Pulse Duration			50				58		ns
t_{HS}	$\overline{\text{HOLD}}$ Setup	12		10				12		ns
t_{HH}	$\overline{\text{HOLD}}$ Hold	3		3				3		ns
t_{PS}	IPORT Setup	12		10				12		ns
t_{PH}	IPORT Hold			3				3		ns
t_{OPD}	Operation Time									
	32-Bit Multiplication		60		50				58	ns
	64-Bit Multiplication		60		50				58	ns
	32-Bit Division (3212)		360		300				345	ns
	64-Bit Division (3212)		720		600				690	ns
	32-Bit ALU Operations		60		50				58	ns
	64-Bit ALU Operations		60		50				58	ns
	32-Bit Division (3222)		960		800				920	ns
	64-Bit Division (3222)		1800		1500				1725	ns
	32-Bit Square Root		1740		1450				1668	ns
	64-Bit Square Root		3480		2900				3335	ns
t_{LAD}	Total Latency									
	32-Bit Multiplication		157		130				150	ns
	64-Bit Multiplication		187		155				179	ns
	32-Bit Division (3212)		457		380				437	ns
	64-Bit Division (3212)		847		705				811	ns
	32-Bit ALU Operations		157		130				150	ns
	64-Bit ALU Operations		187		155				179	ns
	32-Bit Division (3222)		1057		880				1012	ns
	64-Bit Division (3222)		1927		1580				1817	ns
	32-Bit Square Root		1897		1580				1817	ns
	64-Bit Square Root		3577		2980				3427	ns

NOTES
[1]All min and max specifications are over power supply and temperature ranges indicated.
[2]S and T grade parts are available processed and tested in accordance with MIL-STD-883, Class B. The processing and test methods used for S/883B and T/883B versions of the ADSP-3212/ADSP-3222 can be found in Analog Devices' *Military Products Databook*. Regular S and T grade parts are tested at +125°C.
[3]Input levels are GND and +3.0V. Rise times are 5ns max. Input timing reference levels and output reference levels are 1.5V, except for (1) t_{ENA} and t_{DIS} which are as indicated in Figure T1 and (2) t_{DS} and t_{DH} which are measured from clock V_{IH} or V_{IL} crossing points.

Specifications subject to change without notice.

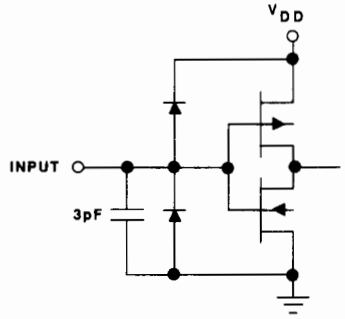


Figure 30. Equivalent Input Circuits

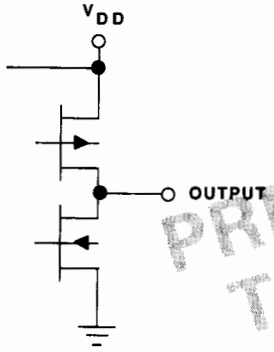


Figure 31. Equivalent Output Circuits

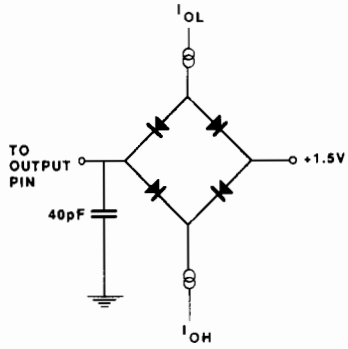


Figure 32. Normal Load for ac Measurements

ABSOLUTE MAXIMUM RATINGS

Supply Voltage . . . . . −0.3V to +7V
Input Voltage . . . . . −0.3V to V_{DD}
Output Voltage Swing . . . . . −0.3V to V_{DD}
Load Capacitance 200pF
Operating Temperature Range (Ambient) . . . . −55°C to +125°C
Storage Temperature Range . . . . . −65°C to +150°C
Lead Temperature (10sec) . . . . . +300°C

ESD SENSITIVITY

The ADSP-3212 and ADSP-3222 feature proprietary input protection circuitry to dissipate high energy discharges (Human Body Model). Per Method 3015 of MIL-STD-883, the ADSP-3212 and ADSP-3222 have been classified as Class 1 devices.

Proper ESD precautions are strongly recommended to avoid functional damage or performance degradation. Charges as high as 4000 volts readily accumulate on the human body and test equipment and discharge without detection. Unused devices must be stored in conductive foam or shunts, and the foam should be discharged to the destination socket before devices are removed. For further information on ESD precautions, refer to Analog Devices' *ESD Prevention Manual*.

ORDERING INFORMATION

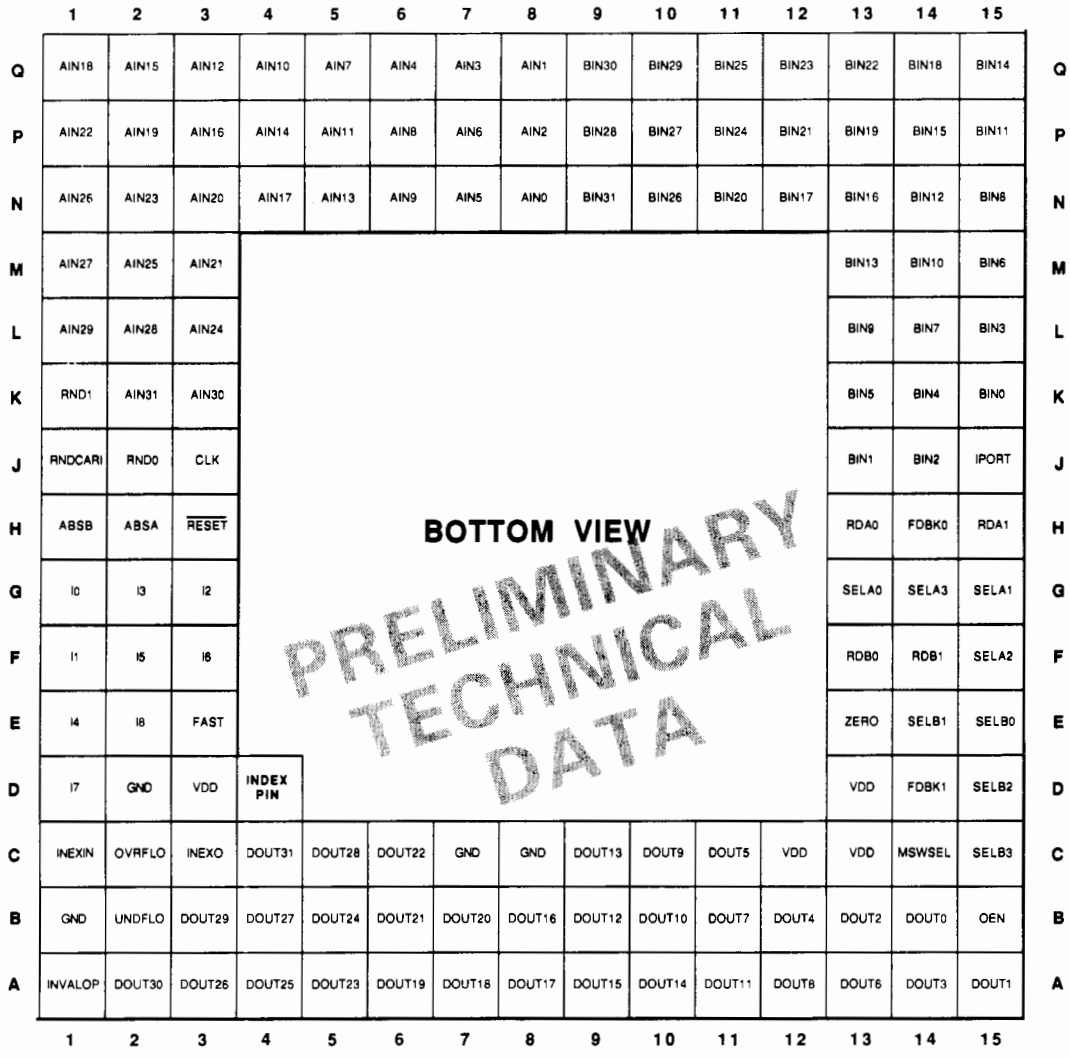
Part Number	Temperature Range	Package	Package Outline
ADSP-3212JG	0 to +70°C	144-Pin Grid Array	G-144A
ADSP-3212KG	0 to +70°C	144-Pin Grid Array	G-144A
ADSP-3212SG	−55°C to +125°C	144-Pin Grid Array	G-144A
ADSP-3212TG	−55°C to +125°C	144-Pin Grid Array	G-144A
ADSP-3212SG/883B	−55°C to +125°C	144-Pin Grid Array	G-144A
ADSP-3212TG/883B	−55°C to +125°C	144-Pin Grid Array	G-144A
ADSP-3222JG	0 to +70°C	144-Pin Grid Array	G-144A
ADSP-3222KG	0 to +70°C	144-Pin Grid Array	G-144A
ADSP-3222SG	−55°C to +125°C	144-Pin Grid Array	G-144A
ADSP-3222TG	−55°C to +125°C	144-Pin Grid Array	G-144A
ADSP-3222SG/883B	−55°C to +125°C	144-Pin Grid Array	G-144A
ADSP-3222TG/883B	−55°C to +125°C	144-Pin Grid Array	G-144A

Contact DSP Marketing in Norwood concerning the availability of other package types.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Q	AIN18	AIN15	AIN12	AIN10	AIN7	AIN4	AIN3	AIN1	BIN30	BIN29	BIN25	BIN23	BIN22	BIN18	BIN14	Q
P	AIN22	AIN19	AIN16	AIN14	AIN11	AIN8	AIN6	AIN2	BIN28	BIN27	BIN24	BIN21	BIN19	BIN15	BIN11	P
N	AIN26	AIN23	AIN20	AIN17	AIN13	AIN9	AIN5	AIN0	BIN31	BIN26	BIN20	BIN17	BIN16	BIN12	BIN8	N
M	AIN27	AIN25	AIN21										BIN13	BIN10	BIN6	M
L	AIN29	AIN28	AIN24										BIN9	BIN7	BIN3	L
K	LOAD64	AIN31	AIN30										BIN5	BIN4	BIN0	K
J	SELA3	IPORT	SELA1										BIN1	BIN2	SELB3	J
H	SELA0	RDA1	SELA2										SELB0	SELB1	SELB2	H
G	RDA0	FAST	WRAPA										RDB1	ABSB	RDB0	G
F	ABSA	MSWSEL	OEN										DIVMUL	CLK	WRAPB	F
E	SHLP	UNDFLO	INVALOP										FDBK1	DP	SP	E
D	TCA	GND	VDD	INDEX PIN									VDD	$\overline{\text{RESET}}$	RND1	D
C	OVRFLO	DENORM	DOUT29	DOUT28	DOUT25	DOUT19	GND	GND	DOUT10	DOUT6	DOUT2	VDD	VDD	FDBK0	RND0	C
B	GND	DOUT30	DOUT26	DOUT24	DOUT21	DOUT18	DOUT17	DOUT13	DOUT9	DOUT7	DOUT4	DOUT1	INEXO	$\overline{\text{HOLD}}$	TCB	B
A	DOUT31	DOUT27	DOUT23	DOUT22	DOUT20	DOUT16	DOUT15	DOUT14	DOUT12	DOUT11	DOUT8	DOUT5	DOUT3	DOUT0	RNDCARO	A
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

BOTTOM VIEW

ADSP-3212 Pinout



ADSP-3222 Pinout