

5 MULTIPLIER

The ADSP-TS201 TigerSHARC processor core contains two computation units known as compute blocks. Each compute block contains a register file and four independent computation units—an ALU, a CLU, a multiplier, and a shifter. The multiplier is highlighted in [Figure 5-1](#). The multiplier takes its inputs from the register file, and returns its outputs to the register file.

The chapter describes:

- “Multiplier Operations” on page 5-5
- “Multiplier Examples” on page 5-22
- “Multiplier Instruction Summary” on page 5-24

The multiplier performs all *multiply operations* for the processor on fixed- and floating-point data and performs all *multiply-accumulate operations* for the processor on fixed-point data. This unit also performs all *complex multiply operations* for the processor on fixed-point data. The multiplier also executes *data compaction operations* on accumulated results when moving data to the register file in fixed-point formats.

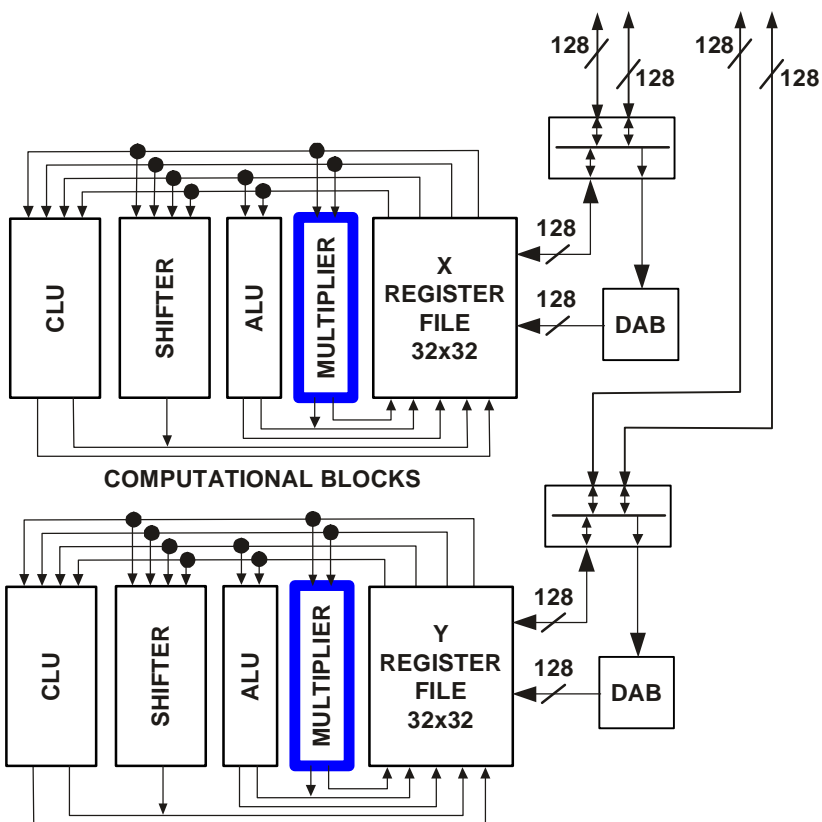


Figure 5-1. Multipliers in Compute Block X and Y

Examining the supported operands for each operation shows that the multiplier operations support these data types:

- Fixed-point fractional and integer *multiply operations* and *multiply-accumulate operations* support:
 - Eight 16-bit (short) input operands with four 16- or 32-bit results
 - Two 32-bit (normal) input operands with a 32- or 64-bit result
- Floating-point fractional *multiply operations* support:
 - Two 32-bit (single-precision) input operands (IEEE standard) with 32-bit result
 - Two 40-bit (extended precision) input operands with 40-bit result
- Fixed-point *data compaction operations* support:
 - 20-bit (short) input operands
 - 40-bit (normal) input operands
 - 80-bit (long) input operands
 - output 16- or 32-bit results

Fixed-point formats include these data size distinctions:

- The multiplier can operate on two 32-bit normal words producing either a 64-bit or a 32-bit result or operate on eight 16-bit short words producing either four 32-bit or four 16-bit results. There is no byte word support in the multiplier.
- The result of a multiplier operation (with the exception of the compress instruction) is always either the same size as the operands, or larger.
 - Normal word multiplication results in either a normal word or a long word result.
 - Quad short word multiplication always results in either a quad short-word or a quad-word results.

The ADSP-TS201 processor supports complex multiply-accumulates. Complex numbers are represented by a pair of short words in a 32-bit register. The least significant bits of the input operands (Rm_L , Rn_L) represent the real part, and the most significant bits of the input operands (Rm_H , Rn_H) represent the imaginary part. The result of a complex multiplication is always stored in a pair of MR registers. The complex multiply-accumulate (indicated with the $**$ operator) is defined as:

$$Real\ Result = (Real_{Rm_L} \times Real_{Rn_L}) - (Imaginary_{Rm_H} \times Imaginary_{Rn_H})$$

$$Imaginary\ Result = (Real_{Rm_L} \times Imaginary_{Rn_H}) + (Imaginary_{Rm_H} \times Real_{Rn_L})$$

Complex multiply-accumulate operations have an option to multiply the first complex operand times the complex conjugate of the second. This complex conjugate operation is defined as:

$$Real\ Result = (Real_{Rm_L} \times Real_{Rn_L}) + (Imaginary_{Rm_H} \times Imaginary_{Rn_H})$$

$$Imaginary\ Result = (Real_{Rm_L} \times Imaginary_{Rn_H}) - (Imaginary_{Rm_H} \times Real_{Rn_L})$$

The complex conjugate option is denoted with a (J) following the instruction. (See [“Complex Conjugate Option” on page 5-17.](#))

The ADSP-TS201 TigerSHARC processor is compatible with the IEEE 32-bit single-precision floating-point data format with minor exceptions. For more information, see [“IEEE Single-Precision Floating-Point Data Format” on page 2-15.](#)

Within instructions, the register name syntax identifies the input operand and output result data size and type. For information on data type selection for multiplier instructions, see [“Register File Registers” on page 2-5.](#) For information on data size selection for multiplier instructions, see the examples in [“Multiplier Operations” on page 5-5.](#)



Note that multiplier instruction conventions for selecting input operand and output result data size differ slightly from the conventions for the ALU and shifter.

The remainder of this chapter presents descriptions of multiplier instructions, options, and results using instruction syntax. For an explanation of the instruction syntax conventions used in multiplier and other instructions, see [“Instruction Line Syntax and Structure” on page 1-22.](#) For a list of multiplier instructions and their syntax, see [“Multiplier Instruction Summary” on page 5-24.](#)

Multiplier Operations

The multiplier performs fixed-point or floating point multiplication and fixed-point multiply-accumulate operations. The multiplier supports several data types in fixed- and floating-point. The floating-point formats are float and float-extended. The input operands and output result of most operations is the compute block register file.

Multiplier Operations

The multiplier has one special purpose, five-word register—the *MR* register—for accumulated results. The ADSP-TS201 processor uses the *MR* register to store the results of fixed-point multiply-accumulate operations. Also, the multiplier can transfer the contents of the *MR* register to the register file before an accumulate operation. The upper 32 bits of the *MR* register (*MR4*) store overflow for multiply accumulate operations. For more information on the register files and register naming syntax for selecting data type and width, see [“Register File Registers” on page 2-5](#).

[Figure 5-2 on page 5-8](#) through [Figure 5-4 on page 5-9](#) show the data flow for multiplier operations. The following are multiplier instructions that demonstrate multiply and multiply-accumulate operations.

```
XR2 = R1 * R0 ;;
```

```
/* This is a fixed-point multiply of two signed fractional 32-bit  
input operands XR1 and XR0; the DSP places the rounded 32-bit  
result in XR2. */
```

```
YR1:0 = R3 * R2 ;;
```

```
/* This is a fixed-point multiply of two signed fractional 32-bit  
input operands YR3 and YR2; the DSP places the 64-bit result in  
YR1:0. */
```



For fixed-point multiply operations, single register names (*Rm*, *Rn*) for input operands select 32-bit input operands. Single versus double register names output for select 32- versus 64-bit output results.


```
XR3:2 = R5:4 * R1:0 ;;
```

```
/* This is a fixed-point multiply of eight signed fractional  
16-bit operands:  
XR5_upper_half * XR1_upper_half,  
XR5_lower_half * XR1_lower_half,  
XR4_upper_half * XR0_upper_half,  
XR4_lower_half * XR0_lower_half;
```

The DSP places the four rounded 16-bit results in XR3_upper_half, XR3_lower_half, XR2_upper_half, and XR2_lower_half, respectively */

```
YR3:0 = R7:6 * R5:4 ;;
```

/* This is similar to the previous example of a quad 16-bit multiply, but the selection of a quad register for output produces 32-bit (instead of 16-bit) results; the DSP places the four results in YR3, YR2, YR1, and YR0. */


 For fixed-point multiply operations, double register names (*Rmd*, *Rnd*) for input operands select 16-bit input operands. Double versus quad register names for output select 16- versus 32-bit output results.

```
XFR2 = R1 * R0 ;;
```

/* This is a floating-point multiply of two single precision input operands XR1 and XR0 (IEEE format); the DSP places the single precision result in XR2. */

```
YFR1:0 = R5:4 * R3:2 ::
```

/* This is a floating-point multiply of two extended precision 40-bit input operands YR5:4 and YR3:2; the DSP places the 40-bit result in YR1:0. */

 For floating-point multiply operations, single register names (*Rm*, *Rn*) for input operands select 32-bit input operands and 32-bit output result. For floating-point multiply operations, double register names (*Rmd*, *Rnd*, *Rsd*) for input and output operands select 40-bit input operands and 40-bit output result.

Multiplier Operations

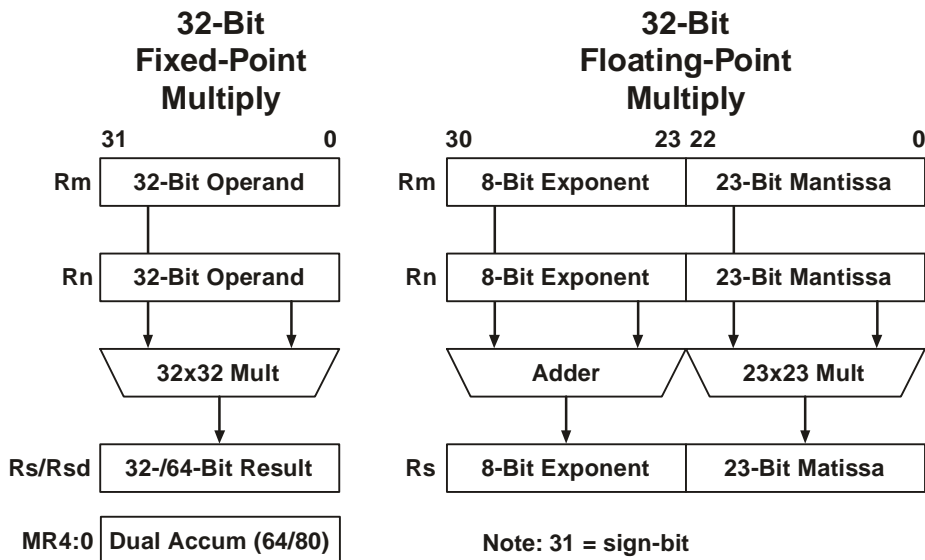


Figure 5-2. 32-Bit Multiplier Operations

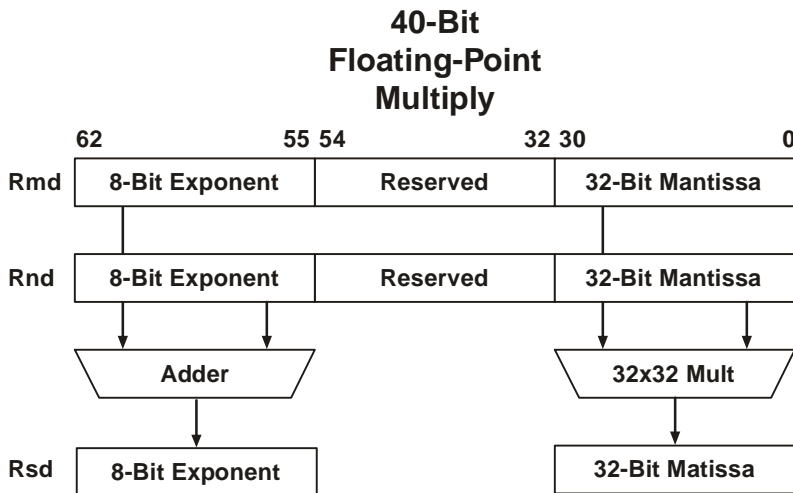


Figure 5-3. 40-Bit Multiplier Operations

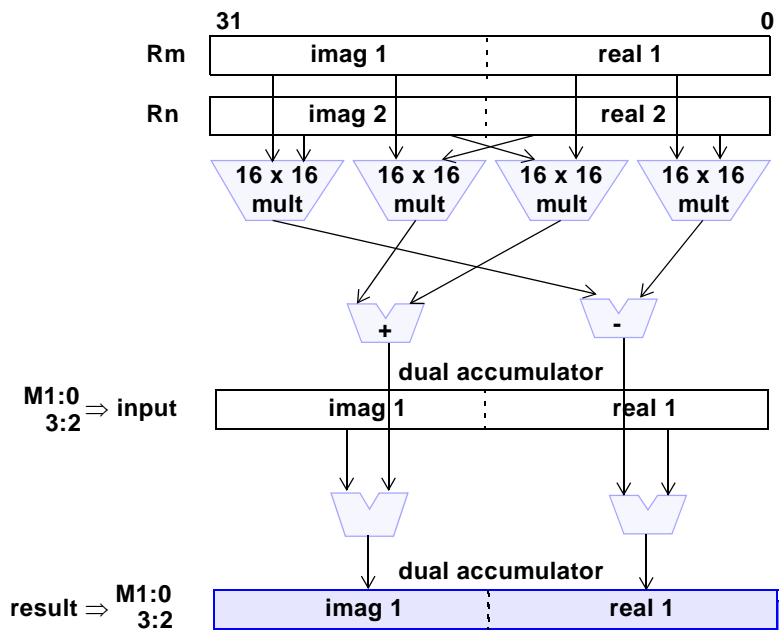


Figure 5-4. 16-Bit Complex Multiplier Operations

Multiplier Instruction Options

Most of the multiplier instructions have options associated with them that permit flexibility in how the instructions execute. It is important to note that these options modify the detailed execution of instructions and options that are particular to a group of instructions—not all options are applicable to all instructions. Instruction options appear in parenthesis at the end of the instruction’s slot.

For a list indicating which options apply for particular multiplier instructions, see [“Multiplier Instruction Summary” on page 5-24](#).

Multiplier Operations

The multiplier instruction options include:

- () signed operation, no saturation¹, round-to-nearest even², fractional mode³
- (U) unsigned operation, no saturation¹, round-to-nearest even²
- (nU) signed/unsigned input
- (I) signed operation, integer mode³
- (S) signed operation, saturation¹
- (T) signed operation, truncation⁴
- (C) clear operation
- (CR) clear/round operation
- (J) complex conjugate operation

The following are multiplier instructions that demonstrate multiply and multiply-accumulate operations with options applied.

```
XR2 = R1 * R0 (U) ;;
```

```
/* This is a fixed-point multiply of two unsigned fractional  
32-bit input operands XR1 and XR0; the DSP places the rounded  
unsigned 32-bit result in XR2. */
```

¹ Where saturation applies

² Where rounding applies

³ Where applies for fixed-point operations

⁴ Where truncation applies

```
YR1:0 = R3 * R2 (I) ::
```

```
/* This is a fixed-point multiply of two integer 32-bit input
operands YR3 and YR2; the DSP places the 64-bit result in YR1:0.
*/
```

```
XFR2 = R1 * R0 (T) ;;
```

```
/* This is a floating-point multiply of two fractional 32-bit
input operands XR1 and XR0 (IEEE format); the DSP places the
truncated 32-bit result in XR2. */
```

Signed/Unsigned Option

The processor always represents fixed-point numbers in 8, 16, 32, or 64 bits, using up to four 32-bit data registers. Fixed-point data in the multiplier may be unsigned or two's-complement. Floating-point data in the multiplier is always signed-magnitude. For information on the supported numeric formats, see [“Numeric Formats” on page 2-15](#).

All fixed-point multiplier instructions may use signed or unsigned data types. The options are:

- () . Both input operands signed (default)
- (U) . Both input operands unsigned
- (nU) . Rm is signed, Rn is unsigned; this option is valid only for $R_s = R_m * R_n$ or $R_{sd} = R_m * R_n$

Fractional/Integer Option

The processor always represents fixed-point numbers in 8, 16, 32, or 64 bits, using up to four 32-bit data registers. In the multiplier, fractional or integer format is available for the fixed-point multiply, multiply-accumulate and COMPACT instructions. For information on the supported numeric formats, see [“Numeric Formats” on page 2-15](#).

Multiplier Operations

The integer and fractional option are defined for the fixed-point operations:

- () . Data is fractional (default)
- (I) . Data is integer

Saturation Option

Saturation is performed on fixed-point operations if option (S) is active¹ when the result *overflows*—crosses the maximum value in which the result format can be represented. In these cases, the returned result is the extreme value that can be represented in the format of the operation following the direction of the correct result. For example, if the format of the result is 16-bit signed and the full result is -0×100000 , the saturated result would be 0×8000 . If the operation is unsigned, the result would be 0×0 ; however, there could not a negative result for the unsigned operation. This can occur in three types of operations:

- Multiply operations

When multiplying integer data and the actual result is outside the range of the destination format, the largest representable number in the destination format is returned. When multiplying fractional data, the special case of -1 times -1 (for example, $0 \times 80 \dots 0$ times $0 \times 80 \dots 0$) always returns the largest representable fraction (for example, $0 \times 7F \dots F$), if saturation is chosen. (See Note 1).

- Multiply-accumulate operations

Saturation affects both integer and fractional data types. Accumulation values are kept at 80-, 40-, and 20-bit precision and are stored in the combination of MR3:0 and MR4 registers (See “[Multiplier Examples](#)” on page 5-22). When performing saturation in a

¹ Except for $R_{sd} = R_m * R_n$, $R_{sq} = R_{md} * R_{nd}$, where it is not optional and implied in the instruction.

multiply-accumulate operation, the resulting value out of the multiplier (at 64, 32, or 16 bits) is added to the current accumulation value. When the accumulation value overflows past 80, 40, or 20 bits, it is substituted by the maximum or minimum possible value. Note that multiply-accumulate operations always saturate.

- MR register transfers

See [“Multiplier Examples” on page 5-22](#).

The final saturated result at 32 --bits for all operations is:

- $0 \times 7F \dots F$ – if operation is signed and result is positive
- $0 \times 80 \dots 0$ – if operation is signed and result is negative
- $0 \times FF \dots F$ – if operation is unsigned and result is positive
- $0 \times 00 \dots 0$ – if operation is unsigned and result is negative (only in signed MR $\text{--} R_m * R_n$)

Saturation option exists for any fixed-point multiplications that may overflow. The following options are available:

- () . No saturation (default)
- (S) . Saturation is active

Truncation Option

For multiplier instructions that support truncation as the (T) option, this option permits selection of the results rounding mode. The processor supports two modes of rounding—round-toward-zero and round-toward-nearest. The rounding modes comply with the IEEE 754 standard and have these definitions:

- Round-Toward-Nearest—not using (T) option. If the result before rounding is not exactly representable in the destination format, the rounded result is the number that is nearer to the result before

Multiplier Operations

rounding. If the result before rounding is exactly halfway between two numbers in the destination format (differing by an LSB), the rounded result is the number that has an LSB equal to zero.

- Round-Toward-Zero—using (T) option. If the result before rounding is not exactly representable in the destination format, the rounded result is the number that is nearer to zero. This is equivalent to truncation.

Statistically, rounding up occurs as often as rounding down, so there is no large sample bias. Because the maximum floating-point value is one LSB less than the value that represents Infinity, a result that is halfway between the maximum floating-point value and Infinity rounds to Infinity in this mode.

Though these rounding modes comply with standards set for floating-point data, they also apply for fixed-point multiplier operations on fractional data. The same two rounding modes are supported, but only the round-to-nearest operation is actually performed by the multiplier. Using its local result register for fixed-point operations, the multiplier round-to-minus infinity by reading only the upper bits of the result and discarding the lower bits.

Round-to-nearest even is executed by adding the LSB to the truncated result if the first bit under the LSB is set and all bits under are cleared—for example, multiplying two short operands. The real result is a normal word. If the expected result is short, it has to be rounded.

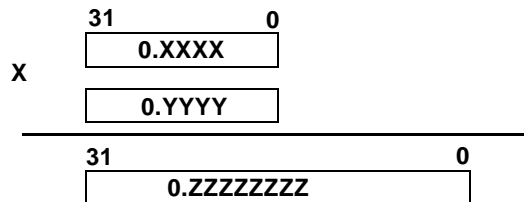


Figure 5-5. Rounding Multiplier Results

Bits 31–16 should be returned, and bits 15–0 should be rounded. The rounding is set according to bit 15 (*round bit*), bit 16 (LSB), and whether bits 14–0 (lower bits) are zero or non-zero:

- If bit 15 is zero, the result is not incremented
- If bit 15 is 1 and bits 14–0 are non-zero, the LSB is incremented by one
- If bit 15 is 1 and bits 14–0 are zero, add bit 16 to the result 31–16

There is no support for round-to-nearest even for multiply-accumulate instructions that also transfer the current MR contents to the register file. If round-to-nearest even is required, transfer the MR registers to the register file as a whole and use the ALU COMPACT instruction. As an alternative, if round-to-nearest (not even) is sufficient, this can be achieved by using the clear and round option in the first multiply-accumulate instruction of the series. [For more information, see “Clear/Round Option” on page 5-15.](#)

Rounding options are:

- () . Round-to-nearest even (default)
- (T) . Truncate—round-to-zero for floating-point and round-to-minus infinity for fixed-point format

Clear/Round Option

Multiply operations and multiply-accumulate with MR register move operations support the clear MR (C) option. Using this option forces the multiplier to clear (=0) the relevant MR register part before the accumulate operation.

Multiply-accumulate operations (without an MR move¹) also support the clear and round (CR) options as an alternative to the clear option.


¹ This is not a critetion. The distinction is the destination result width:
 $RS = MRA, MRA += Rm * Rn (R).$

Multiplier Operations

Using the CR option forces the multiplier to clear MR and set the round bit before the accumulate operation. For more information about rounding and the round bit, see [“Truncation Option” on page 5-13](#).

The clear and clear/round options are:

- () . No change of MR register prior to multiply-accumulate operation (default)
- (C) . Set target MR to zero prior to multiply-accumulate operation
- (CR) . Set target MR to zero and set round bit prior to multiply-accumulate operation

 The CR options may be used only for fractional arithmetic, for example when the option (I) is not used.

When this option is set, the MR registers are set to an initial value of 0x00000000 80000000 for 32-bit fractional multiply-accumulate and 0x00008000 in each of the MR registers for quad 16-bit fractional multiply-accumulate. After this initialization, the result is rounded up by storing the upper part of the result in the end of the multiply-accumulate sequence.

For example with the CR option, assume a sequence of three quad short fractional multiply-accumulate operations (with quad short result) such that the multiplication results are:

Result 1 = 0x0024 0048, 0x0629 4501

Result 2 = 0x0128 0128, 0x2470 2885

Result 3 = 0x1011 fffe, 0x4A30 6d40

Sum = 0x115d 016e, 0x74c9 dac6

In this example, the bottom 16 bits are not to be used if only a short result is expected. Extracting the top 16 bits will give a truncated result, which is 0x115d for the first short and 0x74c9 for the second short. The rounded result is 0x115d for the first short (no change) and 0x74ca (increment) for the second short. If the MR registers are initialized to 0x00008000, the sum result will be:

Sum = 0x115d 816e, 0x74ca 5ac6

The two shorts are exactly the expected results. Note that this is round-to-nearest, and not round-to-nearest even.

The high short is exactly as expected. The rounding method is round-to-nearest, not round-to-nearest even. For information on rounding, see “[Truncation Option](#)” on page 5-13.

Complex Conjugate Option

For complex multiply-accumulate operations (** operator), the multiplier supports the complex conjugate (J) option. The J option directs the multiplier to multiply the R_m operand with the complex conjugate of R_n operand, negating the imaginary part of R_n . For more information, see the discussion [on page 5-4](#). The options are:

- () . No conjugate
- (J) . Conjugate for complex multiply

Multiplier Result Overflow (MR4) Register

The MR4 register holds the extra bits (overflow) from a multiply-accumulate operation. MR4 register fields are assigned according to the MR register used and the size of the result. See:

- [Figure 5-6](#)—Result is a long word (80-bit accumulation)
- [Figure 5-7](#)—Result is word (40-bit accumulation)
- [Figure 5-8](#)—Result is short (20-bit accumulation)

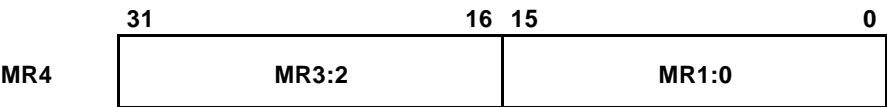


Figure 5-6. MR4 for Long Word Result (80-Bit Accumulation)

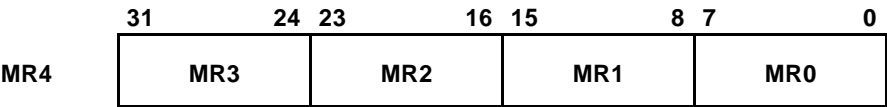
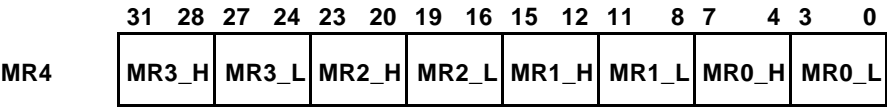


Figure 5-7. MR4 for Normal Word Result (40-Bit Accumulation)



_H indicates High short word field
_L indicates Low short word field

Figure 5-8. MR4 for Short Word Result (20-Bit Accumulation)

These bits are also used as the input to the accumulate step of the multiply-accumulate operation. The bits are cleared together with the clear of the corresponding MR register, and when stored, are used for saturation. The purpose of these bits is to enable the partial result of a multiply-accumulate sequence to go beyond the range assigned by the final result.

Multiplier Execution Status

Multiplier operations update status flags in the compute block's Arithmetic Status (XSTAT and YSTAT) register (see [Figure 2-2 on page 2-4](#) and [Figure 2-3 on page 2-5](#)). Programs can use status flags to control execution of conditional instructions and initiate software exception interrupts. For more information, see “[Multiplier Execution Conditions](#)” on [page 5-20](#).

[Table 5-1](#) shows the flags in XSTAT or YSTAT that indicate multiplier status (a 1 indicates the condition) for the most recent multiplier operation.

Table 5-1. Multiplier Status Flags

Flag	Definitions	Updated By...
MZ	Multiplier fixed-point zero and floating-point underflow or zero	All fixed- and floating-point multiplier ops, except multiply accumulate
MN	Multiplier result is negative	All fixed- and floating-point multiplier ops, except multiply accumulate
MV	Multiplier overflow	All fixed- and floating-point multiplier ops, except multiply accumulate
MU	Multiplier underflow	All floating-point multiplier ops; cleared by fixed-point ops, unchanged by multiply accumulate
MI	Multiplier floating-point invalid operation	All floating-point multiplier ops; cleared by fixed-point ops, unchanged by multiply accumulate

Multiplier Operations

Multiplier operations also update sticky status flags in the compute block's Arithmetic Status (XSTAT and YSTAT) register. Table 5-2 shows the flags in XSTAT or YSTAT that indicate multiplier sticky status (a 1 indicates the condition) for the most recent multiplier operation. Once set, a sticky flag remains high until explicitly cleared.

Table 5-2. Multiplier Sticky Status Flags

Flag	Definition	Updated By...
MUS	Multiplier underflow, sticky	All floating-point multiply ops
MVS	Multiplier floating-point overflow, sticky	All floating-point multiply ops
MOS	Multiplier fixed-point overflow, sticky	All fixed-point multiply ops
MIS	Multiplier floating-point invalid operation, sticky	All floating-point multiply ops

Flag update occurs at the end of each operation and is available on the next instruction slot. A program cannot write the arithmetic status register explicitly in the same cycle that the multiplier is performing an operation.

Multi-operand instructions (for example, $Rsd = Rmd * Rnd$) produce multiple sets of results. In this case, the processor determines a flag by ORing the result flag values from individual results.

Multiplier Execution Conditions

In a conditional multiplier instruction, the execution of the entire instruction line can depend on the specified condition at the beginning of the instruction line. Conditional multiplier instructions take the form:

```
IF cond; D0, instr.; D0, instr.; D0, instr. ;;
```

This syntax permits up to three instructions to be controlled by a condition. Omitting the D0 before the instruction makes the instruction unconditional.

Table 5-3 lists the multiplier conditions. For more information on conditional instructions, see [“Conditional Execution” on page 8-12](#).

Table 5-3. Multiplier Conditions

Condition	Description	Flags Set
MEQ	Multiplier equal to zero	MZ = 1
MLT	Multiplier less than zero	MN and MZ = 1
MLE	Multiplier less than or equal to zero	MN or MZ = 1
NMEQ	NOT (Multiplier equal to zero)	MZ = 0
NMLT	NOT (Multiplier less than zero)	MN or MZ = 0
NMLE	NOT (Multiplier less than or equal to zero)	MN and MZ = 0

Multiplier Static Flags

In the program sequencer, the static flag (SFREG) can store status flag values for later usage in conditional instructions. With SFREG, each compute block has two dedicated static flags X/YSCF0 (condition is SF0) and X/YSCF1 (condition is SF1). The following example shows how to load a compute block condition value into a static flag register.

```
XSCF0 = XMEQ ;; /* Load X-compute block MEQ flag into XSCF0 bit
in static flags (SFREG) register */
```

```
IF SF0, XR5 = R4 * R3 ;; /* the SF0 condition tests the XSCF0
static flag */
```

For more information on static flags, see [“Conditional Execution” on page 8-12](#).

Multiplier Examples

[Listing 5-1](#) provides a number of example multiply and multiply-accumulate instructions. The comments with the instructions identify the key features of the instruction, such as fixed- or floating-point format, input operand size, and register usage.

Listing 5-1. Multiplier Instruction Examples

```
XYR4 = R6 * R8 ;;

/* This instruction is a 32-bit fractional multiply that produces
a 32-bit rounded result. */

XYR5:4 = R6 * R8 ;;

/* This instruction is a 32-bit fractional multiply that produces
a 64-bit result. */

XR11:10 = R9:8 * R7:6 ;;

/* This instruction is a quad 16-bit multiply; the input operands
are XR9_H x XR7_H, XR9_L x XR7_L, XR8_H x XR6_H, and
XR8_L x XR6_L (where _H is high half and _L is low half); the
16-bit results go to XR11_H, XR11_L, XR10_H, and XR10_L (respec-
tively). */

XMR3:2 += R1 * R0 ;;

/* This is a multiplication of source operands XR1 and XR0, and
the multiplication result is added to the current contents of the
target XMR registers, overflowing into XMR4_H. */

YMR1:0 -= R3 * R2 ;;

/* This is a multiplication of source operands YR3 and YR2, and
the multiplication result is subtracted from the current contents
of the target YMR registers, overflowing into YMR4_L. */

XR7 = MR3:2, MR3:2 += R1 * R0 ;;
```

```
/* This instruction executes a multiply-accumulate and transfers
the previous MR registers into the register file; the previous
value in the MR registers is transferred to the register file. */
```

```
YMR3:0 += R5:4 * R7:6 ;;
```

```
/* This instruction is four multiplications of four 16-bit shorts
in register pair YR5:4 and four 16-bit shorts in pair YR7:6. The
four results are accumulated in MR3:0 as a word result. The over-
flow bits are written into MR4. */
```

```
XMR3:2 += R9:8 * R7:6 ;;
```

```
/* This instruction is a quad 16-bit multiply-accumulate with
16-bit results; the input operands are XR9_H x XR7_H,
XR9_L x XR7_L, XR8_H x XR6_H, and XR8_L x XR6_L (where _H is high
half and _L is low half); the 16-bit accumulated results go to
XMR3_H, XMR3_L, XMR2_H, and XMR2_L (respectively). */
```

```
MR3:0 += R9:8 * R7:6 ;;
```

```
/* This instruction is a quad 16-bit multiply-accumulate with
32-bit results; the input operands are XR9_H x XR7_H,
XR9_L x XR7_L, XR8_H x XR6_H, and XR8_L x XR6_L (where _H is high
half and _L is low half); the 32-bit accumulated results go to
XMR3, XMR2, XMR1, and XMR0 (respectively). */
```

```
XMR1:0 += R9 ** R7 ;;
```

```
/* This instruction is a multiplication of the complex value in
XR9 and the complex value in XR7. The result is accumulated in
XMR1:0. */
```

```
XFR20 = R22 * R23 (T) ;;
```

```
/* This is a 32-bit (single precision) floating-point multiply
instruction with 32-bit result; single registers select 32-bit
operation. */
```

Multiplier Instruction Summary

```
YFR25:24 = R27:26 * R30:29 (T) ;;
```

```
/* This is a 40-bit (extended precision) floating-point multiply  
instruction with 40-bit result; double registers select 40-bit  
operation. */
```


Multiplier Instruction Summary


The following listings show the multiplier instructions' syntax:

- [Listing 5-2 on page 5-25](#) “32-Bit Fixed-Point Multiplication Instructions”
- [Listing 5-3 on page 5-26](#) “16-Bit Fixed-Point Quad Multiplication Instructions”
- [Listing 5-4 on page 5-26](#) “16-Bit Fixed-Point Complex Multiplication Instructions”
- [Listing 5-5 on page 5-26](#) “32- and 40-Bit Floating-Point Multiplication Instructions”
- [Listing 5-6 on page 5-26](#) “Multiplier Register Load Instructions”

The conventions used in these listings for representing register names, optional items, and choices are covered in detail in [“Register File Registers” on page 2-5](#). Briefly, these conventions are:

- { } – The curly braces enclose options; these braces are not part of the instruction syntax.
- | – The vertical bars separate choices; these bars are not part of the instruction syntax.
- *Rmd* – The register names in italic represent user selectable single (*Rs*, *Rm*, *Rn*), double (*Rsd*, *Rmq*, *Rnd*) or quad (*Rsq*) register names.

 Each instruction presented here occupies one instruction slot in an instruction line. For more information about instruction lines and instruction combination constraints, see [“Instruction Line Syntax and Structure” on page 1-22](#) and [“Instruction Parallelism Rules” on page 1-26](#).

 The MR3:0 registers are four 32-bit accumulation registers. They overflow into MR4, which stores two 16-bit overflows for 32-bit multiples, four 8-bit overflows for quad 16-bit multiples, or eight 4-bit overflows for quad-short 16-bit multiples.

Listing 5-2. 32-Bit Fixed-Point Multiplication Instructions

$$\begin{aligned} \{X|Y|XY\}Rs &= Rm * Rn \{(\{U|nU\}\{I\}\{T\}\{S\})\} ;^1 \\ \{X|Y|XY\}Rsd &= Rm * Rn \{(\{U|nU\}\{I\})\} ; \\ \{X|Y|XY\}MRa &+= Rm * Rn \{(\{U\}\{I\}\{C|CR\})\} ;^2 \\ \{X|Y|XY\}MRa &- = Rm * Rn \{(\{I\}\{C|CR\})\} ; \\ \{X|Y|XY\}Rs &= MRa, MRa += Rm * Rn \{(\{U\}\{I\}\{C|CR\})\} ; \textit{dual op} \end{aligned}$$

¹ Options include: (): fractional, signed, and no saturation; (S): saturation, signed, (SU): saturation, unsigned

² Options include: (): signed, round-to-nearest even, (T): signed, truncate, (U): unsigned, round-to-nearest even, (TU): unsigned, truncate

Multiplier Instruction Summary

```
{X|Y|XY}Rsd = MRa, MRa += Rm * Rn {{U}{I}{C}} ; dual op  
/* where MRa is either MR1:0 or MR3:2 */
```

Listing 5-3. 16-Bit Fixed-Point Quad Multiplication Instructions

```
{X|Y|XY}Rsd = Rmd * Rnd {{U}{I}{T}{S}} ;  
{X|Y|XY}Rsq = Rmd * Rnd {{U}{I}} ;  
{X|Y|XY}MRb += Rmd * Rnd {{U}{I}{C|CR}} ;  
{X|Y|XY}Rsd = MRb, MRb += Rmd * Rnd {{I}{C|CR}} ; dual op  
/* where MRb is either MR1:0, MR3:2, or MR3:0 */
```

Listing 5-4. 16-Bit Fixed-Point Complex Multiplication Instructions

```
{X|Y|XY}MRa += Rm ** Rn {{I}{C|CR}{J}} ;  
{X|Y|XY}MRa -= Rm ** Rn {{I}{C|CR}{J}} ;  
{X|Y|XY}Rs = MRa, MRa += Rm ** Rn {{I}{C|CR}{J}} ; dual op  
{X|Y|XY}Rsd = MRa, MRa += Rm ** Rn {{I}{C|CR}{J}} ; dual op  
/* where MRa is either MR1:0 or MR3:2 */
```

Listing 5-5. 32- and 40-Bit Floating-Point Multiplication Instructions

```
{X|Y|XY}FRs = Rm * Rn {(T)} ;  
{X|Y|XY}FRsd = Rmd * Rnd {(T)} ;
```

Listing 5-6. Multiplier Register Load and Store Instructions

```
{X|Y|XY}{S|L}MRa = Rmd {(SE|ZE)} ;  
{X|Y|XY}MR4 = Rm ;  
{X|Y|XY}{S}Rsd = MRa {{U}{S}} ;  
{X|Y|XY}Rsq = MR3:0 {{U}{S}} ;  
{X|Y|XY}Rs = MR4 ;  
/* where MRa is either MR1:0 or MR3:2 */
```

```

{X|Y|XY}Rsd = SMRb {(U)} ; /* extract 2 short words */
{X|Y|XY}LRsd = MRb {(U)} ; /* extract 1 normal word */
/* where MRb is either MR0, MR1, MR2, or MR3 */
{X|Y|XY}Rsqr = SMRa {(U)} ; /* extract 4 short words */
{X|Y|XY}LRsq = MRa {(U)} ; /* extract 2 normal words */
{X|Y|XY}QRsq = LMRa {(U)} ; /* extract 1 long word from MRa */
/* where MRa is either MR1:0 or MR3:2 */
{X|Y|XY}Rs = COMPACT MRa {(U){I}{S}} ;
{X|Y|XY}SRsd = COMPACT MR3:0 {(U){I}{S}} ;
/* where MRa is either MR1:0 or MR3:2 */

```

Multiplier Instruction Summary