

# Chapter 3

## Data Arithmetic Logic Unit

### 3.1 Introduction

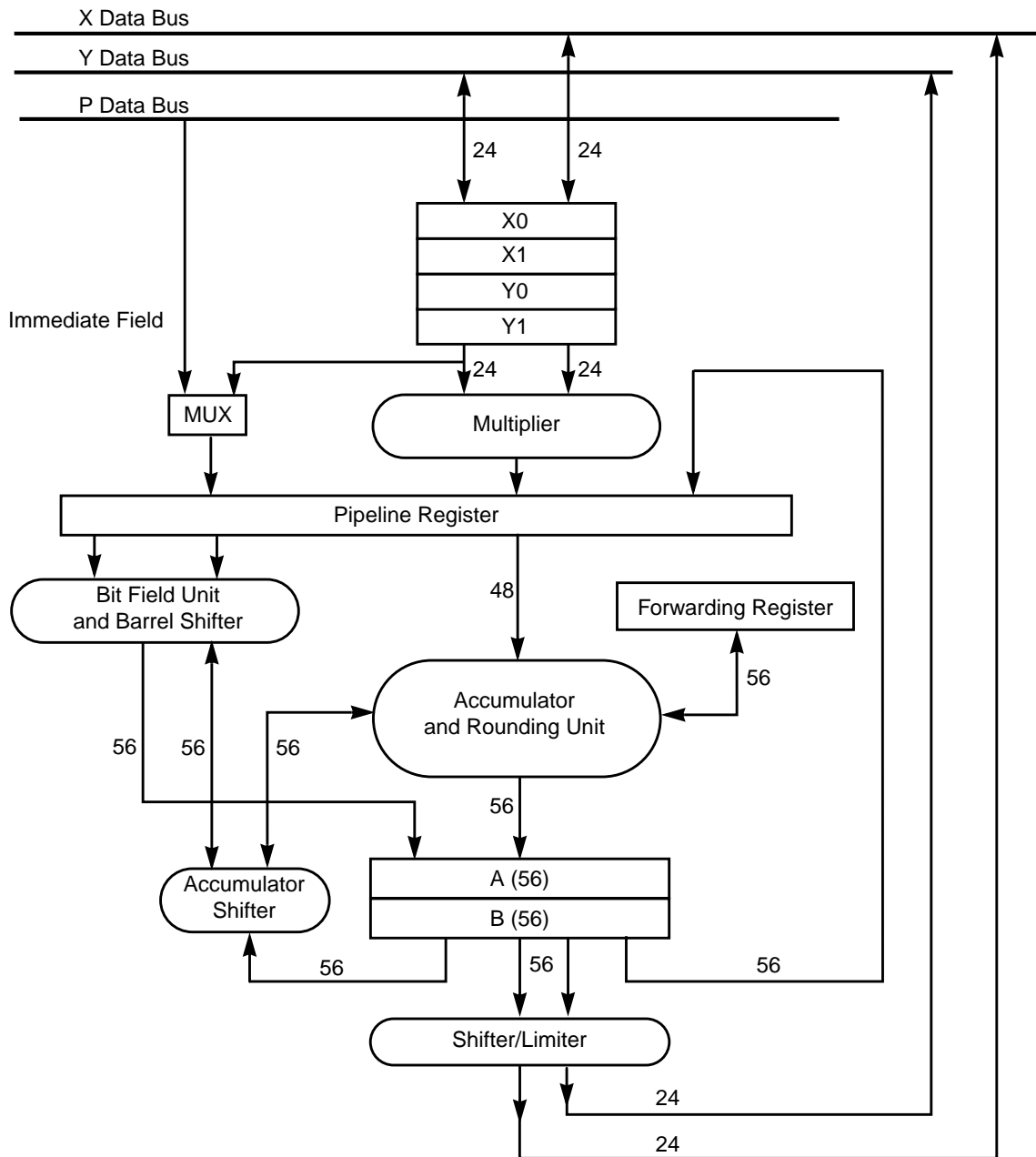
This section describes the architecture and the operation of the Data Arithmetic Logic Unit (Data ALU), the block where all the arithmetic and logical operations on data operands are performed.

### 3.2 Data ALU Architecture

The Data ALU contains the following components:

- Four 24-bit input registers
- A fully pipelined Multiplier-Accumulator (MAC)
- Two 48-bit accumulator registers
- Two 8-bit accumulator extension registers
- A Bit Field Unit (BFU) with a 56-bit barrel shifter
- An accumulator shifter
- Two data bus shifter/limiter circuits

**Figure 3-1** is a block diagram of the Data ALU.



**Figure 3-1.** Data ALU Block Diagram

The Data ALU registers can be read or written over the X Data Bus (XDB) and the Y Data Bus (YDB) as 24- or 48-bit operands. The source operands for the Data ALU, which can be 24, 48, or 56 bits, always originate from Data ALU registers. The results of all Data ALU operations are stored in an accumulator. The Data ALU runs in 16-bit Arithmetic mode when the SA bit in the Status Register (SR) is set. For details on the SR, see **Chapter 5, "Program Control Unit,"**.

---

All the Data ALU operations are performed in two clock cycles in pipeline fashion so that a new instruction can be initiated in every clock, yielding an effective execution rate of one instruction per clock cycle.

### 3.2.1 Data ALU Input Registers (X1, X0, Y1, Y0)

X1, X0, Y1, and Y0 are four 24-bit, general-purpose data registers. They can be treated as four independent 24-bit registers or as two 48-bit registers called X and Y, formed by concatenation of X1:X0 and Y1:Y0, respectively. X1 is the most significant word in X, and Y1 is the most significant word in Y. The registers serve as input buffers between the X Data Bus (XDB) or Y Data Bus (YDB) and the MAC unit or barrel shifter. They are used as Data ALU source operands, allowing new operands to be loaded for the next instruction while the current contents are used by the current instruction. The registers can also be read back out to the appropriate data bus.

### 3.2.2 Multiplier-Accumulator (MAC) Unit

The Multiplier-Accumulator (MAC) unit is the main arithmetic processing unit of the DSP56300 core. It accepts up to three input operands and outputs one 56-bit result of the following form:

Extension:Most Significant Product:Least  
Significant Product (EXT:MSP:LSP)

The operation of the MAC unit occurs independently and in parallel with XDB and YDB activity, and its registers facilitate buffering for both Data ALU inputs and outputs. Latches on the MAC unit input permit writing new data to an input register while the Data ALU processes the current data. The input to the multiplier can come only from the X or Y registers. The multiplier executes 24-bit x 24-bit, parallel fractional multiplies, between two's-complement signed, unsigned, or mixed operands. The 48-bit product is right-justified into 56 bits and added to the 56-bit contents of either the A or B accumulator.

The 56-bit sum is stored back in the same accumulator. The multiply/accumulate operation is fully pipelined and takes two clock cycles to complete. In the first clock the multiply is performed and the product is stored in the pipeline register. In the second clock the accumulator is added or subtracted. If a multiply without accumulation (MPY) is specified in the instruction, the MAC clears the accumulator and then adds the contents to the product. When a 56-bit result is to be stored as a 24-bit operand, the LSP can simply be truncated, or it can be rounded into the MSP. Rounding is performed if specified in the DSP instruction, for example, in the signed multiply-accumulate and round (MACR) instruction; the rounding is either convergent rounding (round-to-nearest-even) or

twos-complement rounding. The type of rounding is specified by the rounding bit in the Status Register (SR). The bit in the accumulator that is rounded is specified by the scaling mode bits in the SR.

The arithmetic unit's result going into the accumulator can be saturated so that it fits into 48 bits (MSP and LSP). This process is commonly referred to as arithmetic saturation. It is activated by the Arithmetic Saturation Mode (SM) bit in the SR. The purpose of this mode is to provide for algorithms that do not recognize or cannot take advantage of the extension accumulator (EXT). For details, refer to **Section 3.3.3**, "Arithmetic Saturation Mode," on page 3-11.

### **3.2.3 Data ALU Accumulator Registers (A2, A1, A0, B2, B1, B0)**

The six Data ALU registers (A2, A1, A0, B2, B1, and B0) form two general-purpose, 56-bit accumulators, A and B. Each of these two accumulators consists of three concatenated registers (A2:A1:A0 and B2:B1:B0, respectively). The 24-bit MSP is stored in A1 or B1; the 24-bit LSP is stored in A0 or B0. The 8-bit EXT is stored in A2 or B2. If an ALU operation results in overflow into A2 (or B2), reading the A (or B) accumulator over the XDB or YDB substitutes a limiting constant in place of the value in the accumulator. The content of A or B is not affected if limiting occurs; only the value transferred over the XDB or YDB is limited. This process is commonly referred to as transfer saturation and should not be confused with the Arithmetic Saturation mode.

The overflow protection is performed after the contents of the accumulator are shifted according to the Scaling mode. Shifting and limiting is performed only when the entire 56-bit A or B register is specified as the source for a parallel data move over the XDB or YDB. When A2, A1, A0, B2, B1, or B0 is the source for a parallel data move, shifting and limiting are not performed. When the 8-bit wide accumulator extension register (A2 or B2) is the source for a parallel data move, it is sign-extended to produce the full 24-bit wide word. The accumulator registers (A or B) serve as buffer registers between the arithmetic unit and the XDB and/or YDB. These registers are used as both Data ALU source and destination operands.

Automatic sign extension of the 56-bit accumulators is provided when the A or B register is written with a smaller operand. Sign extension can occur when writing A or B from the XDB and/or YDB or with the results of certain Data ALU operations such as the transfer conditionally (Tcc) or transfer Data ALU register (TFR) instructions. If a word operand is to be written to an accumulator register (A or B), the Most Significant Product (MSP)—A1 or B1—of the accumulator is written with the word operand, the Least Significant Product (LSP)—A0 or B0—is zero-filled, and the Extended (EXT) portion—A2 or B2—is sign-extended from MSP. Long-word operands are written into the low-order portion, MSP:LSP, of the Accumulator Register, and the EXT portion is

---

sign-extended from MSP. No sign extension is performed if an individual 24-bit register is written (A1, A0, B1, or B0). Test logic in each accumulator register supports operation of the data shifter/limiter circuits. This test logic detects overflows out of the data shifter so that the limiter can substitute one of several constants to minimize errors due to the overflow.

### 3.2.4 Accumulator Shifter

The accumulator shifter is an asynchronous parallel shifter with a 56-bit input and a 56-bit output that is implemented immediately before the MAC unit accumulator input. The source accumulator shifting operations are as follows:

- No shift (unmodified)
- 24-bit right shift (arithmetic) for DMAC
- 16-bit right shift (arithmetic) for DMAC in Sixteen-bit Arithmetic mode
- Force to zero

### 3.2.5 Bit Field Unit (BFU)

The Bit Field Unit (BFU) contains a 56-bit parallel bidirectional shifter with a 56-bit input and a 56-bit output, mask generation unit and logic unit. The bit field unit is used in the following operations:

- Multibit left shift (arithmetic or logical) for ASL, LSL
- Multibit right shift (arithmetic or logical) for ASR, LSR
- 1-Bit rotate (right or left) for ROR, ROL
- Bit field merge, insert and extract for MERGE, INSERT, EXTRACT and EXTRACTU
- Count leading bits for CLB
- Fast normalization for NORMF
- Logical operations for AND, OR, EOR, and NOT

### 3.2.6 Data Shifter/Limiter

The data shifter/limiter circuits provide special post-processing on data read from the ALU accumulator registers A and B out to the XDB or YDB. Each of the two independent shifter/limiter circuits (one for XDB and one for the YDB) consists of a shifter followed by a limiting circuit.

### 3.2.6.1 Scaling

The data shifters in the shifters/limiters unit can perform the following data shift operations:

- Scale up—shift data one bit to the left
- Scale down—shift data one bit to the right
- No scaling—pass the data unshifted

Each data shifter has a 24-bit output with overflow indication. These shifters permit dynamic scaling of fixed-point data without modifying the program code. For example, this permits block floating-point algorithms such as Fast Fourier Transforms (FFTs) to be implemented in a regular fashion. The data shifters are controlled by the Scaling Mode bits (S0 and S1, bits 11 and 10) in the SR.

### 3.2.6.2 Limiting

In the DSP56300 core, the Data ALU accumulators A and B have eight extension bits. Limiting occurs when the extension bits are in use and either A or B is the source being read over XDB or YDB. The limiters in the DSP56300 core place a shifted and limited value on XDB or YDB without changing the contents of the A or B registers. Having two limiters allows two-word operands to be limited independently in the same instruction cycle. The two data limiters can also be combined to form one 48-bit data limiter for long-word operands.

If the contents of the selected source accumulator are represented without overflow in the destination operand size (that is, signed integer portion of the accumulator is not in use), the data limiter is disabled, and the operand is not modified. If the contents of the selected source accumulator are not represented without overflow in the destination operand size, the data limiter substitutes a limited data value having maximum magnitude (saturated) and having the same sign as the source accumulator contents:

- \$7FFFFFFF for 24-bit positive numbers
- \$7FFFFFFF FFFFFFFF for 48-bit positive numbers
- \$800000 for 24-bit negative numbers
- \$800000 000000 for 48-bit negative numbers

This process is called transfer saturation. The value in the accumulator register is not shifted or limited and can be reused within the Data ALU. When limiting does occur, a flag is set and latched in the SR.

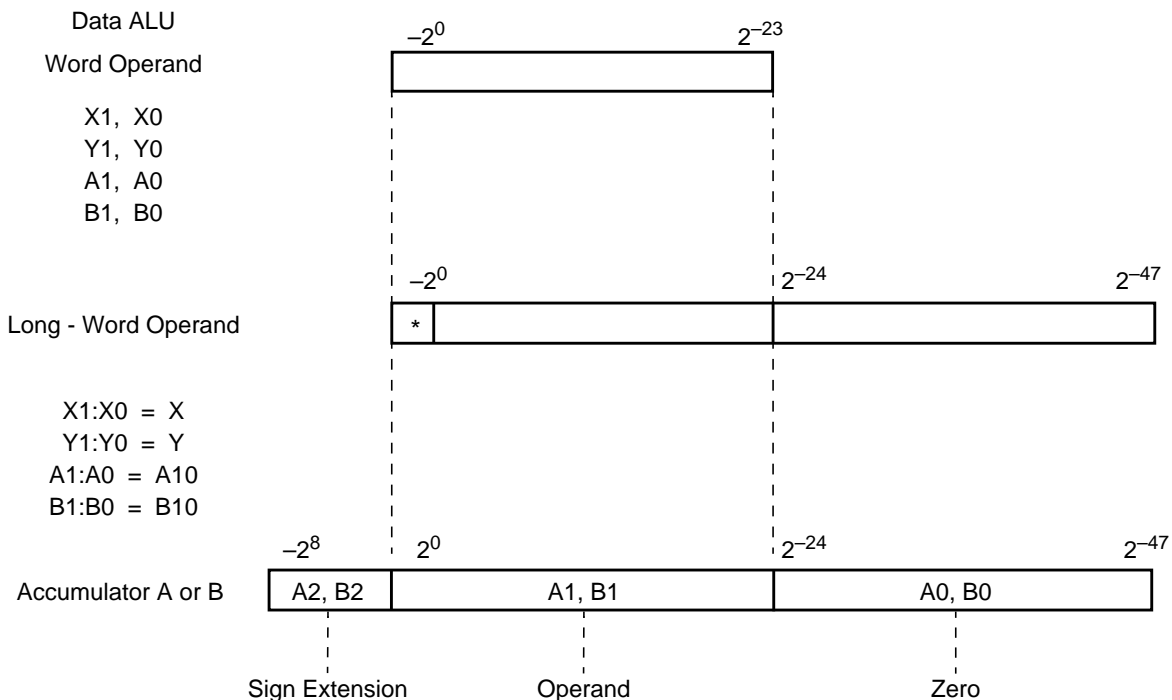
### 3.3 Data ALU Arithmetic and Rounding

The following paragraphs describe the Data ALU data representation, rounding modes, and arithmetic methods.

#### 3.3.1 Data Representation

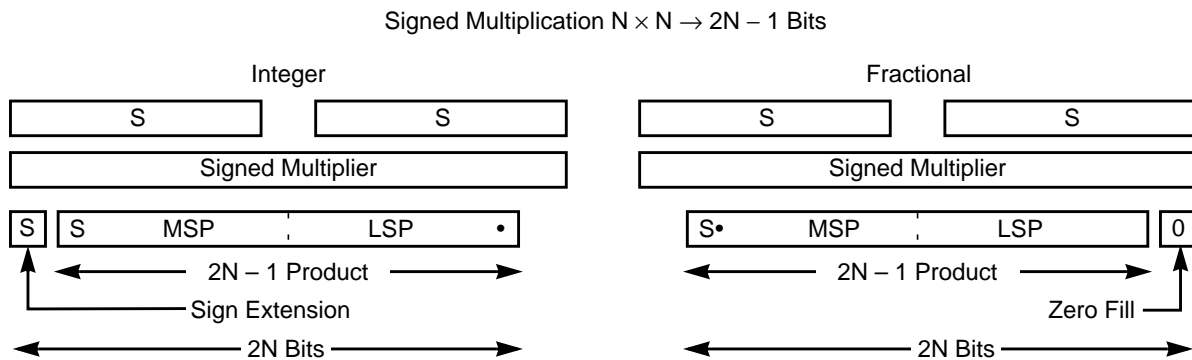
The DSP56300 core uses a fractional data representation for all Data ALU operations.

**Figure 2** shows the bit weighting of words, long words, and accumulator operands for this representation. The decimal points are all aligned and are left-justified. For words and long words, the most negative number that can be represented is  $-1.0$  whose internal representation is  $\$800000$  and  $\$800000000000$ , respectively. The most positive word is  $\$7FFFFFFF$  or  $1 - 2^{-23}$ , and the most positive long word is  $\$7FFFFFFFFFFFFFFF$  or  $1 - 2^{-47}$ . These limitations apply to all data stored in memory and to data stored in the Data ALU input buffer registers. The extension registers associated with the accumulators allow word growth so that the most positive number is approximately 256, and the most negative number is  $-256$ . To maintain alignment of the binary point when a word operand is written to accumulator A or B, the operand is written to the most significant accumulator register (A1 or B1), and its most significant byte is automatically sign-extended through the accumulator extension register (A2 or B2). The least significant accumulator register (A0 or B0) is automatically cleared. When a long-word operand is written to an accumulator, the least significant word of the operand is written to the least significant accumulator register (see **Figure 3-2**).



**Figure 3-2.** Bit Weighting and Alignment of Operands

The number representation for integers is between  $\pm 2^{(N-1)}$ ; whereas, the fractional representation is limited to numbers between  $\pm 1$ . To convert from an integer to a fractional number, the integer must be multiplied by a scaling factor so the result is always between  $\pm 1$ . The representation of integer and fractional numbers is the same if the numbers are added or subtracted, but it is different if the numbers are multiplied or divided. An example of two numbers multiplied together is given in **Figure 3**.



**Figure 3-3.** Integer/Fractional Multiplication

The key difference is in the alignment of the  $2N - 1$  bit product. In fractional multiplication, the  $2N - 1$  significant product bits are left-aligned, and a zero is filled in the Least Significant Bit (LSB), to maintain fractional representation. In integer multiplication, the  $2N - 1$  significant product bits are right-aligned, and the sign bit should be duplicated to maintain integer representation.

**Note:** Be aware when multiplying integer numbers that since the DSP56300 core incorporates a fractional array multiplier, it always aligns the  $2N - 1$  significant product bits to the left.

### 3.3.2 Rounding Modes

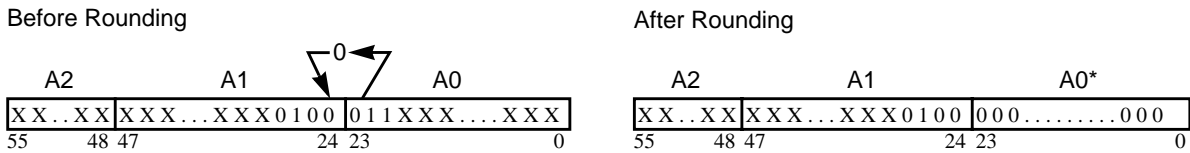
The DSP56300 core Data ALU rounds the accumulator register to single precision if requested in the instruction. The upper portion of the accumulator is rounded according to the contents of the lower portion of the accumulator. The boundary between the lower portion and the upper portion is determined by the scaling mode bits  $S_0$  and  $S_1$  in the Status Register (SR). Two types of rounding are implemented: convergent rounding and twos-complement rounding. The type of rounding is selected by the Rounding Mode (RM) bit in the EMR portion of the SR.

#### 3.3.2.1 Convergent Rounding

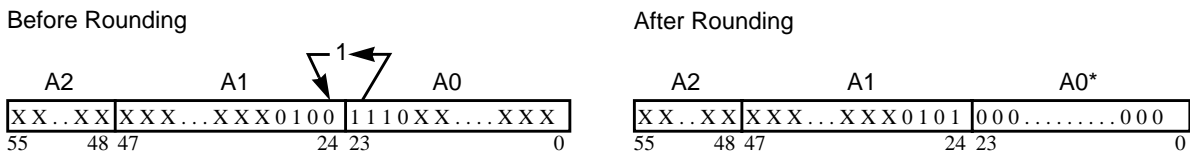
Convergent rounding (also called round-to-nearest even number) is the default rounding mode. The traditional rounding method rounds up any value greater than one-half and

rounds down any value less than one-half. The question arises as to which way one-half should be rounded. If it is always rounded one way, the results are eventually biased in that direction. Convergent rounding solves the problem by rounding down if the number is even (LSB = 0) and rounding up if the number is odd (LSB = 1). **Figure 3-4** shows the four cases for rounding a number in the A1 (or B1) register. If scaling is set in the SR, the rounding position is updated to reflect the alignment of the result when it is put on the data bus. However, the contents of the register are not scaled.

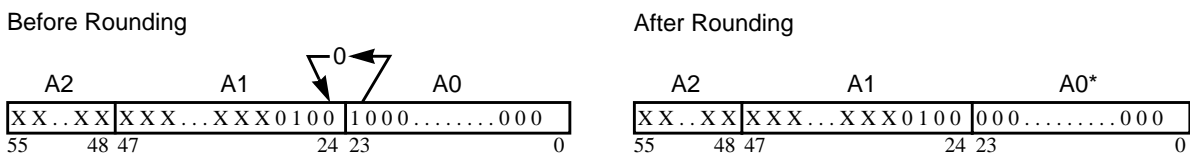
**Case I:** If  $A0 < \$800000$  ( $1/2$ ), then Round Down (Add Nothing)



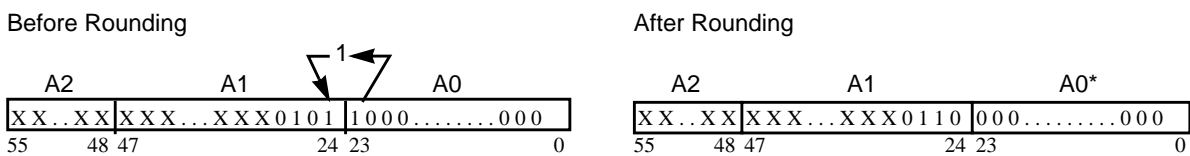
**Case II:** If  $A0 > \$800000$  ( $1/2$ ), then Round Up (Add 1 to A1)



**Case III:** If  $A0 = \$800000$  ( $1/2$ ), and the LSB of A1 = 0, then Round Down (Add Nothing)



**Case IV:** If  $A0 = \$800000$  ( $1/2$ ), and the LSB = 1, then Round Up (Add 1 to A1)



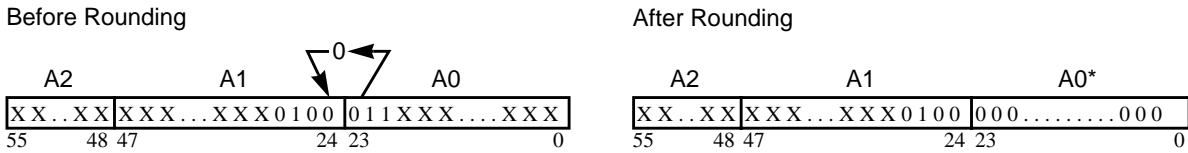
\*A0 is always clear; performed during RND, MPYR, MACR.

**Figure 3-4.** Convergent Rounding (No Scaling)

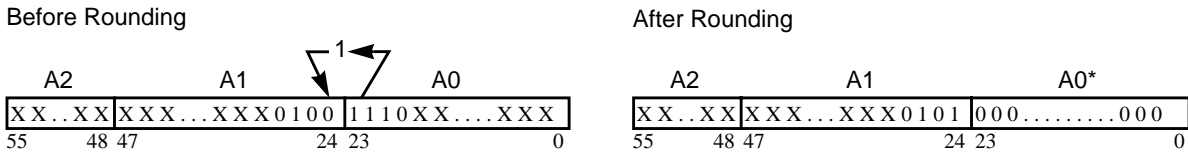
### 3.3.2.2 Twos Complement Rounding

When twos complement rounding is selected by setting the Rounding Mode (RM) bit in the SR, all values greater than or equal to one-half are rounded up, and all values less than one-half are rounded down. Therefore, a small positive bias is introduced. **Figure 3-5** shows the four cases for rounding a number in the A1 (or B1) register. If scaling is set in the SR, the rounding position is updated to reflect the alignment of the result when it is put on the data bus. However, the contents of the register are not scaled.

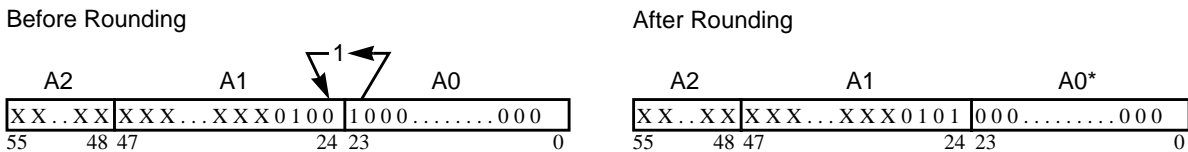
**Case I:** If  $A0 < \$800000$  (1/2), then Round Down (Add Nothing)



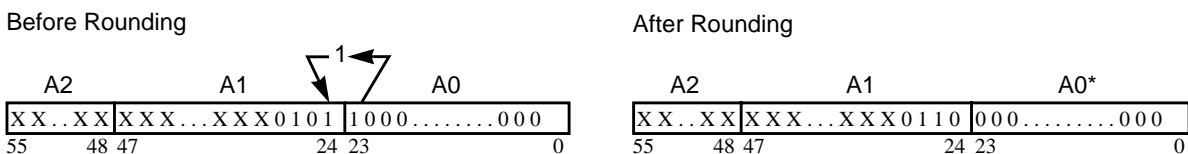
**Case II:** If  $A0 > \$800000$  (1/2), then Round Up (Add 1 to A1)



**Case III:** If  $A0 = \$800000$  (1/2), and the LSB of A1 = 0, then Round Up (Add 1 to A1)



**Case IV:** If  $A0 = \$800000$  (1/2), and the LSB of A1 = 1, then Round Up (Add 1 to A1)



\*A0 is always clear; performed during RND, MPYR, MACR.

**Figure 3-5.** Twos Complement Rounding (No Scaling)

### 3.3.3 Arithmetic Saturation Mode

Setting the Arithmetic Saturation Mode (SM) bit in the SR limits the arithmetic unit's result to 48 bits (MSP and LSP). The highest dynamic range of the machine is then limited to 48 bits. The purpose of the SM bit is to provide a saturation mode for algorithms that do not recognize or cannot take advantage of the extension accumulator. The arithmetic saturation logic operates by checking 3 bits of the 56-bit result after rounding: two bits of the extension byte (EXT[7] and EXT[0]) and one bit on the MSP (MSP[23]). The result obtained in the accumulator when SM = 1 is shown in **Table 3-1**.

**Table 3-1** Actions of the Arithmetic Saturation Mode (SM = 1)

EXT[7]	EXT[0]	MSP[23]	Result in Accumulator
0	0	0	Unchanged
0	0	1	\$00 7FFFFFFF FFFFFFFF
0	1	0	\$00 7FFFFFFF FFFFFFFF
0	1	1	\$00 7FFFFFFF FFFFFFFF
1	0	0	\$FF 800000 000000
1	0	1	\$FF 800000 000000
1	1	0	\$FF 800000 000000
1	1	1	Unchanged

The two saturation constants \$007FFFFFFF FFFFFFFF and \$FF800000 000000 are not affected by the Scaling mode. Similarly, rounding of the saturation constant during execution of MPYR, MACR, and RND instructions is independent of the scaling mode: \$007FFFFFFF FFFFFFFF is rounded to \$007FFFFFFF000000, and \$FF800000 000000 is rounded to \$FF800000 000000.

In Arithmetic Saturation mode, the Overflow bit (V bit) in the SR is set if the Data ALU result is not representable in the 48-bit accumulator (i.e., an arithmetic saturation has occurred). This also implies that the Limiting bit (L bit) in the SR is set when an arithmetic saturation occurs.

**Note:** The Arithmetic Saturation mode is *always* disabled during execution of the following instructions: TFR, Tcc, DMACsu, DMACuu, MACsu, MACuu, MPYsu, MPYuu, CMPU, and all BFU operations. If the result of these instructions should be saturated, a MOVE A,A (or B,B) instruction must be added after the original instruction if no scaling is set. However, the “V” bit of the SR is never set by the arithmetic saturation of the accumulator during execution of a MOVE A,A (or B,B) instruction. Only the “L” bit is set.

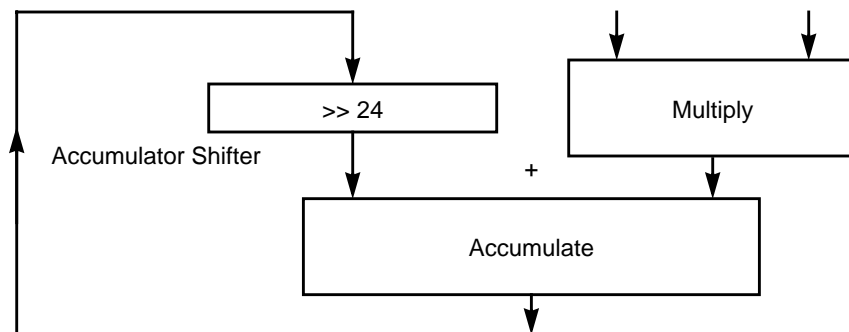
### 3.3.4 Multiprecision Arithmetic Support

A set of Data ALU operations facilitate multiprecision multiplications. When these instructions are used, the multiplier accepts some combinations of signed twos-complement format and unsigned format. **Table 3-2** shows these instructions.

**Table 3-2** Acceptable Signed and Unsigned Twos-Complement Multiplication

Instruction	Description
MPY/MAC su	Multiplication and multiply-accumulate with signed times unsigned operands
MPY/MAC uu	Multiplication and multiply-accumulate with unsigned times unsigned operands
DMACss	Multiplication with signed times signed operands and 24-bit arithmetic right shift of the accumulator before accumulation
DMACsu	Multiplication with signed times unsigned operands and 24-bit arithmetic right shift of the accumulator before accumulation
DMACuu	Multiplication with unsigned times unsigned operands and 24-bit arithmetic right shift of the accumulator before accumulation

**Figure 3-6** shows how the DMAC instruction is implemented inside the Data ALU.



**Figure 3-6.** DMAC Implementation

**Figure 3-7** illustrates the use of these instructions for a double-precision multiplication. The signed  $\times$  signed operation multiplies or multiply-accumulates the two upper signed portions of two signed double-precision numbers. The unsigned  $\times$  signed operation multiplies or multiply-accumulates the upper signed portion of one double-precision number with the lower unsigned portion of the other double-precision number. The unsigned  $\times$  unsigned operation multiplies or multiply-accumulates the lower unsigned portion of one double-precision number with the lower unsigned portion of the other double-precision number.

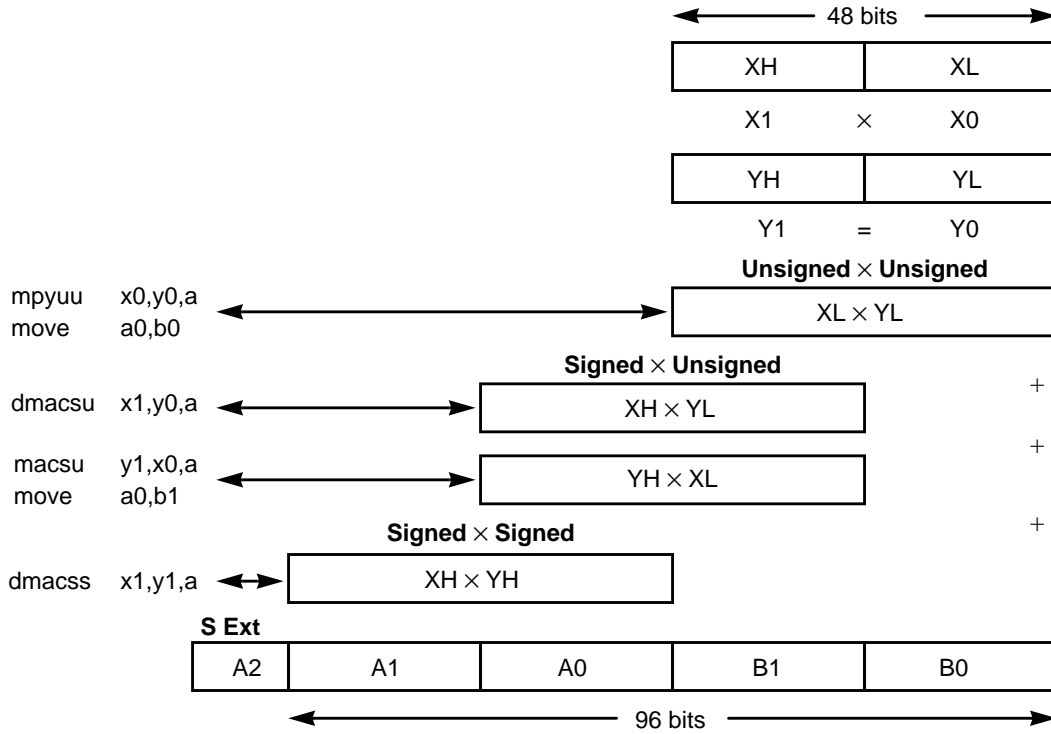


Figure 3-7. Double-Precision Multiplication Using DMAC

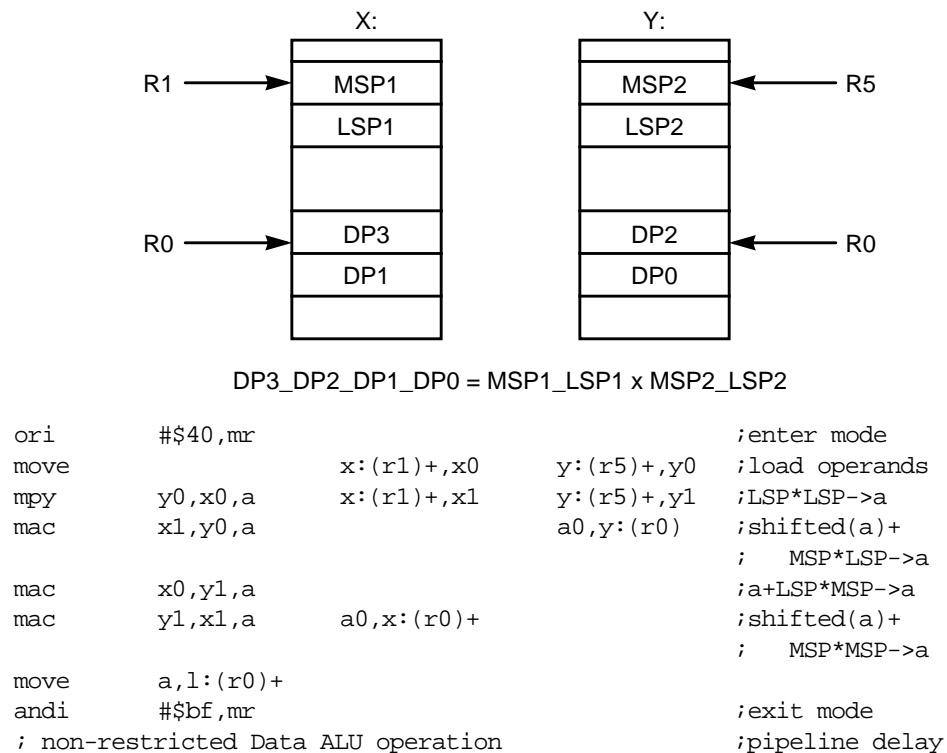
### 3.3.4.1 Double-Precision Multiply Mode

To support existing DSP56000 code, double-precision multiply operations can also be performed within a dedicated “Double-Precision Multiply” mode using a double-precision algorithm with four multiply operations. Select the Double-Precision Multiply mode by setting Bit 14 (DM) of the SR. The mode is disabled by clearing the same DM bit.

The double-precision multiply algorithm is shown in **Figure 3-8**. The ORI instruction sets the DM mode bit, but due to the instruction execution pipeline the Data ALU enters the Double-precision Multiply mode after only one cycle. The ANDI instruction clears the DM mode bit in the MR, but due to the instruction execution pipeline the Data ALU leaves the mode after one cycle. To allow for the pipeline delay, do not follow the ANDI instruction immediately with a restricted Data ALU instruction.

In Double-Precision Multiply mode, the behavior of the four specific operations listed in the double-precision algorithm is modified. Therefore, in Double-Precision Multiply mode, do not use these operations with the specified register combinations for any purpose other than the double-precision multiply algorithm. Also, in this mode, do not use any other Data ALU operations (or the four listed operations with other register combinations).

**Note:** Since the double-precision multiply algorithm uses the Y0 register for all stages, do not change Y0 when running the double-precision multiply algorithm. If the Data ALU is required by an interrupt service routine, save the contents of Y0 with the contents of the other Data ALU registers before processing the interrupt routine, and restore them before leaving the interrupt routine.



**Figure 3-8.** Double-Precision Algorithm

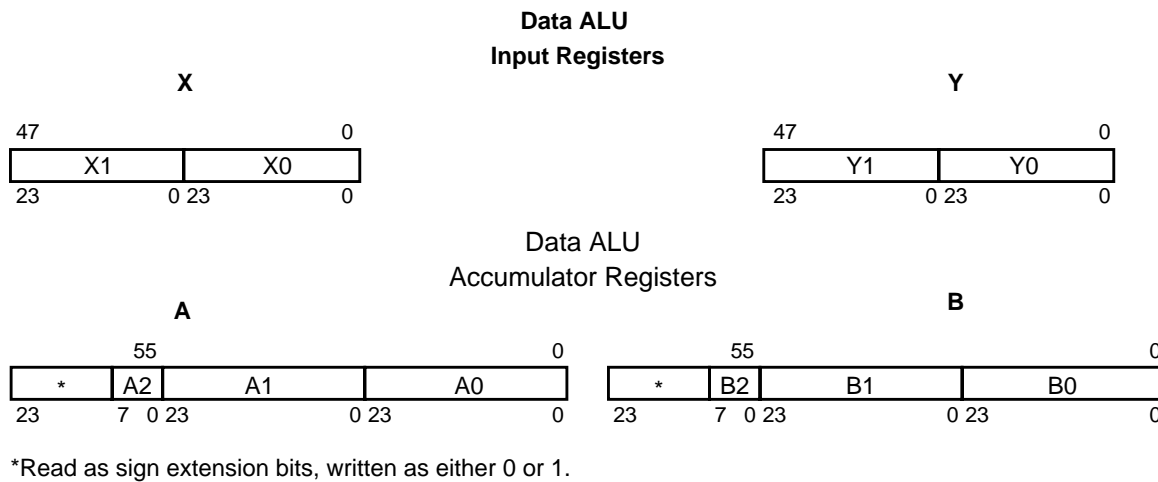
### 3.3.5 Block Floating-Point FFT Support

The Block Floating Point FFT operation requires the early detection of data growth between FFT butterfly passes. If data growth is detected, suitable down-scaling must be applied to ensure that no overflow occurs during the next butterfly calculation pass. The total scaling applied is the block exponent of the FFT output. The Block Floating Point FFT algorithm is described in the Motorola application note APR4/D, *Implementation of Fast Fourier Transforms on Motorola's DSP56000/DSP56001 and DSP96002 Digital Signal Processors*.

Data growth detection is implemented as a status bit in the SR. The FFT scaling bit S, Bit 7 of the SR, is set when a result moves from accumulator A or B to the XDB or YDB Bus (during an accumulator to memory or accumulator to register move) and remains set until explicitly cleared (i.e., the “S” bit is a “sticky” bit).

### 3.4 Data ALU Programming Model

The Data ALU features 24-bit input/output data registers that can be concatenated to accommodate 48-bit data and two 56-bit accumulators, which are segmented into three 24-bit pieces that can be transferred over the buses. **Figure 3-9** illustrates how the registers in the programming model are grouped.

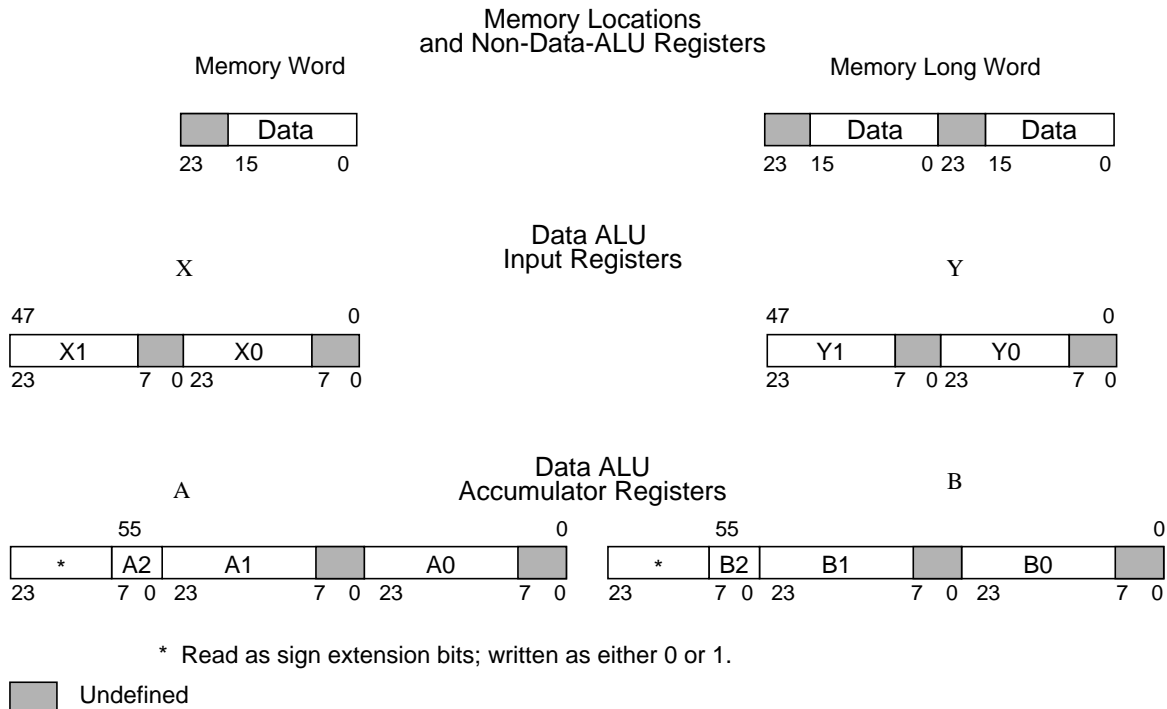


**Figure 3-9.** Data ALU Core Programming Model

### 3.5 Sixteen-Bit Arithmetic Mode

Setting the SA bit in the SR enables the Sixteen-bit Arithmetic mode of operation. In this mode, the 16-bit data is right-aligned in the 24-bit memory word, that is, in the 16 LSBs of the 24-bit word. You can use 16-bit wide data memories by either leaving the eight MSBs unconnected or by tying these bits to GND.

In the Sixteen-bit Arithmetic mode of operation, the source operands can be 16-bit, 32-bit, or 40-bit. The numerical results have a 40-bit accuracy. These 40 bits consist of a 16-bit LSP, a 16-bit MSP, and an 8-bit EXT. **Figure 3-10** shows the bit positions in the memory and Data ALU registers in Sixteen-bit Arithmetic mode.



- Notes:
1. When switching to and from Sixteen-bit Arithmetic mode, no arithmetic instruction or a MOVE instruction should be performed for two instruction cycles. The programmer must insert two NOP instructions. There is no automatic stall insertion for this change.
  2. Be cautious about exchanging data between Sixteen-bit Arithmetic mode and 24-bit arithmetic mode via write-read operations on Data ALU registers and accumulators. Since the write operations in Sixteen-bit Arithmetic mode corrupt the information in the least significant bytes of the registers or accumulators, do not use these registers or accumulators for 24-bit data without some processing.

**Figure 3-10.** Sixteen-Bit Arithmetic Mode Data Organization

### 3.5.1 Moves in Sixteen-Bit Arithmetic Mode

In Sixteen-bit Arithmetic mode, the Data ALU registers are still read or written as 24- or 48-bit operations over the XDB and the YDB. No 16- or 32-bit moves are supported. The mapping of the 16-bit data to the 24-bit buses is described in the following paragraphs.

**Table 3-3** shows the result of moving data into registers or accumulators. **Table 3-4** shows the result of moving data from registers or accumulators.

#### 3.5.1.1 Moves into Registers or Accumulators

When XDB or YDB are moved into a full Data ALU accumulator (A or B), the 16 LSBs of the bus are placed in bits 32–47 of the accumulator (16 MSBs of A1 or B1). Bits 8–23 of the accumulator (16 MSBs of A0 or B0) are cleared and the EXT of the accumulator (A2 or B2) is loaded with the sign extension. When XDB and YDB (48 bits) are moved into a full Data ALU accumulator (A or B), the 16 LSBs from XDB are placed into bits 32–47 of the accumulator (16 MSBs of A1 or B1). The 16 LSBs from YDB are placed

into bits 8–23 of the accumulator (16 MSBs of A0 or B0). The EXT of the accumulator (A2 or B2) is loaded with the sign extension.

When XDB or YDB is moved into a register (X0, X1, Y0 or Y1) or partial accumulator (A0, A1, B0 or B1), the 16 LSBs of the bus are loaded into the 16 MSBs of the destination register. No other portion of the accumulator is affected.

When XDB or YDB is moved into the accumulator extension register (A2 or B2), the 8 LSBs of the bus are loaded into the 8 LSBs of the destination register and the 16 MSBs of the bus are not used. The remaining parts of the accumulator are not affected.

When XDB and YDB are moved into a 48-bit register (X or Y) or partial accumulator (A10 or B10), the 16 LSBs of XDB bus are loaded into the 16 MSBs of the MSP (X1, Y1, A1 or B1) and the 16 LSBs of YDB bus are loaded into the 16 MSBs of the LSP (X0, Y0, A0 or B0). The EXT part of the accumulator (A2 or B2) is not affected.

**Table 3-3** Moves into Registers or Accumulators

Data Source	Destination	Result
XDB or YDB	Full Data ALU accumulator (A or B)	<ul style="list-style-type: none"> <li>16 LSBs of bus into bits 32-47 of accumulator</li> <li>Accumulator bits 8-23 cleared</li> <li>EXT of accumulator (A2 or B2) loaded with sign extension</li> </ul>
XDB and YDB	Full Data ALU accumulator (A or B)	<ul style="list-style-type: none"> <li>16 LSBs of XDB into bits 32-47 of accumulator</li> <li>16 LSBs of YDB into bits 8-23 of the accumulator</li> <li>EXT of accumulator (A2 or B2) loaded with sign extension</li> </ul>
XDB or YDB	Register (X0, X1, Y0, or Y1) or partial accumulator (A0, A1, B0, or B1)	<ul style="list-style-type: none"> <li>16 LSBs of bus into 16 MSBs of destination register</li> <li>Remaining parts of accumulator not affected</li> </ul>
XDB or YDB	Accumulator extension register (A2 or B2)	<ul style="list-style-type: none"> <li>8 LSBs of bus into 8 LSBs of destination register</li> <li>16 MSBs of bus not used</li> <li>Remaining parts of accumulator not affected</li> </ul>
XDB and YDB	48-bit register (X or Y) or partial accumulator (A10 or B10)	<ul style="list-style-type: none"> <li>16 LSBs of XDB into 16 MSBs of MSP</li> <li>16 LSBs of YDB into 16 MSBs of LSP</li> <li>EXT of accumulator (A2 or B2) not affected</li> </ul>

**3.5.1.2 Moves from Registers or Accumulators**

When a partial accumulator (A0, A1, B0 or B1) is moved to the XDB or YDB, the 16 MSBs of the source are transferred to the 16 LSBs of the bus with eight zeros in the MSBs. No scaling or limiting is performed. When the source is the accumulator extension register (A2 or B2), it occupies the 8 LSBs of the bus while the next 16 bits are the sign extension of Bit 7.

When a partial accumulator (A10 or B10) is moved to XDB and YDB, the 16 MSBs of the MSP of the source (A1 or B1) are transferred to the 16 LSBs of XDB with eight zeros in the MSBs, while the 16 MSBs of the LSP of the source (A0 or B0) are transferred to the 16 LSBs of YDB with eight zeros in the MSBs. No scaling or limiting is performed.

When a full Data ALU accumulator (A or B) is moved to XDB or YDB, scaling and limiting is performed, and then the 16-bit scaled and limited word is placed on the 16 LSBs of the bus and the sign extension is placed in the eight MSBs on the bus.

When a full Data ALU accumulator (A or B) is moved to XDB and YDB, scaling and limiting is performed, and then the 16 MSBs of the 32-bit scaled and limited double word are placed on XDB 16 LSBs, and the sign extension is placed in the eight MSBs on the bus. The 16 LSBs of the 32-bit scaled and limited double word are placed on the 16 LSBs of the YDB with eight zeros on the eight MSBs of the bus.

When a register (X0, X1, Y0 or Y1) is moved to XDB or YDB, the 16 MSBs of the source are transferred to the 16 LSBs of the bus with eight zeros in the MSBs.

When a 48-bit register (X or Y) is moved to XDB and YDB, the 16 MSBs of the high register (X1 or Y1) are placed on the 16 LSBs of the XDB, and eight zeroes are placed on the eight MSBs of the bus. The 16 LSBs of the low register (X0 or Y0) are placed on the 16 LSBs of the YDB with eight zeros on the eight MSBs of the bus.

**Note:** When a read operation of a Data ALU register (X, Y, X0, X1, Y0 or Y1) immediately follows a write operation to the same register, the value placed on the eight MSBs of the XDB or YDB is undefined.

**Table 3-4** Moves from Registers or Accumulators

Data Source	Destination	Result
Partial accumulator (A0, A1, B0, or B1)	XDB or YDB	<ul style="list-style-type: none"> <li>■ 16 MSBs of source into 16 LSBs of bus with eight zeros in MSBs</li> <li>■ No scaling or limiting</li> </ul>
Accumulator extension register (A2 or B2)	XDB or YDB	<ul style="list-style-type: none"> <li>■ Source occupies 8 LSBs of bus</li> <li>■ Next 16 bits are sign extension of Bit 7</li> </ul>
Partial accumulator (A10 or B10)	XDB and YDB	<ul style="list-style-type: none"> <li>■ 16 MSB of MSP of source (A1 or B1) transferred to 16 LSBs of XDB with eight zeros in MSBs</li> <li>■ 16 MSBs of the LSP of source (A0 or B0) transferred to 16 LSBs of YDB with eight zeros in the MSBs.</li> <li>■ No scaling or limiting</li> </ul>
Full Data ALU accumulator (A or B)	XDB or YDB	<ul style="list-style-type: none"> <li>■ Scaling and limiting performed</li> <li>■ 16-bit scaled word placed on 16 LSBs of bus</li> <li>■ Sign extension placed in eight MSBs of bus</li> </ul>

**Table 3-4** Moves from Registers or Accumulators (Continued)

Data Source	Destination	Result
Full Data ALU accumulator (A or B)	XDB and YDB	<ul style="list-style-type: none"> <li>■ Scaling and limiting performed</li> <li>■ 16 MSBs of 32-bit scaled and limited double word placed on XDB 16 LSBs</li> <li>■ Sign extension placed in eight MSBs on bus</li> <li>■ 16 LSBs of 32-bit scaled and limited double word placed on 16 LSBs of YDB with eight zeros on the eight MSBs of bus</li> </ul>
Register (X0, X1, Y0 or Y1)	XDB or YDB	<ul style="list-style-type: none"> <li>■ 16 MSBs transferred to 16 LSBs of bus with eight zeros in MSBs</li> </ul>
48-bit register (X or Y)	XDB and YDB	<ul style="list-style-type: none"> <li>■ 16 MSBs of high register (X1 or Y1) placed on 16 LSBs of XDB with eight zeros on eight MSBs of bus</li> <li>■ 16 LSBs of low register (X0 or Y0) placed on 16 LSBs of YDB with eight zeros on eight MSBs of bus</li> </ul>

### 3.5.1.3 Short Immediate moves

When an Immediate Short Data MOVE is performed in Sixteen-bit Arithmetic mode and the destination register is A0, A1, B0, or B1, the 8-bit immediate short operand is interpreted as an unsigned integer and is therefore stored in Bits 15–8 of the register (which correspond to the eight LSBs of a 16-bit number). If the destination register is A2 or B2, the 8-bit immediate short operand is stored in Bits 7–0 of the register.

When the destination register is A, B, X0, X1, Y0, or Y1, the 8-bit immediate short operand is interpreted as a signed fraction and is stored in bits 47–40 of the accumulator or bits 23–16 of a register (which correspond to the eight MSBs of a 16-bit number).

### 3.5.1.4 Scaling and Limiting

If scaling is specified, the data shifter virtually concatenates the 16-bit LSP to the 16-bit MSP to provide a numerically correct shift.

During the Sixteen-bit Arithmetic mode of operation, the limiting is affected as described below:

- The maximum positive value is \$007FFF (\$007FFF00FFFF for double precision).
- The maximum negative value is \$008000 (\$008000000000 for double precision).

## 3.5.2 Sixteen-bit Arithmetic

When an operand is read from a Data ALU register or accumulator to the arithmetic unit, the 8 LSBs of the 24-bit word are ignored (that is, read as zeros). The arithmetic unit forces these bits to zero when generating a result.

The arithmetic unit virtually concatenates the 16-bit LSP with the 16-bit MSP to form a continuous number. Therefore, all arithmetic operations, including shifts, are numerically correct. The execution of Data ALU instructions in Sixteen-bit Arithmetic mode is not affected, except for the following:

- The operand and result widths are 16/32/40 instead of 24/48/56.
- The rounding, if specified by the operation, is performed on the Most Significant Bit of the 16-bit Least Significant Portion (LSP) of the result, that is on the bit corresponding to bit 23 of A0/B0 (the Scaling mode affects this position accordingly). See the RND instruction in **Chapter 13**, "*Instruction Set*," for details.
- The arithmetic saturation detection is unchanged, but the saturated values change to \$007FFF00FFFF00 and \$FF800000000000.
- In ADC/SBC instructions, the Carry bit C is added/subtracted to the LSB of the 16-bit LSP.
- Logic operations affect only the 16-bit wide word.
- Rotation in rotate instructions is performed on a 16-bit wide word.
- The possible normalization range changes, thus affecting the CLB instruction.
- The DMAC instruction performs a 16-bit arithmetic right shift of the accumulator before accumulation.
- The double-precision multiplication algorithm is not supported, even if the Double-precision Multiply mode bit is set.
- The bit parsing instructions (MERGE, EXTRACT, EXTRACTU, and INSERT) are modified by the Sixteen-bit Arithmetic mode to perform on the appropriate bit positions of the 16-bit data. For the INSERT instruction, you must update the offset by adding a bias value of 16. Refer to **Chapter 13**, "*Instruction Set*," for details on specific instructions.
- In the read-modify-write instructions (BCHG, BCLR, BSET and BTST) and in the Jump/Branch on bit instructions (BRCLR, BRSET, BSCLR, BSSET, JCLR, JSET, JSCLR, and JSSET), the bit numbering in Sixteen-bit Arithmetic mode is relative to 16-bit wide words (that is, Bit 0 is the LSB and Bit 15 is the MSB). Do not use bit numbers greater than 15.

## 3.6 Pipeline Conflicts

No pipeline dependencies exist when the result of the Data ALU is used as a source operand for the immediately following Data ALU instruction. However, Data ALU operations can produce pipeline conflicts as described in the following paragraphs.

### 3.6.1 Arithmetic Stall

Since every Data ALU instruction completes in two clock cycles, an interlock condition occurs during an attempt to read an accumulator (or parts of an accumulator) if the preceding instruction is a Data ALU instruction that specifies the same accumulator as the destination. This interlock condition, arithmetic stall, is detected in hardware, and an idle cycle (no op) is inserted, thereby guaranteeing the correctness of the result. You can optimize code by inserting a useful instruction before the read instruction. **Figure 11** describes cases in which the pipelined nature of the Data ALU generates an arithmetic stall.

```

;following example illustrates a one-clock pipeline delay when
;trying to read an accumulator as source for move:
mac    x0,y0,a                ;data ALU operation
move  a1,x:(r0)+             ;one clock delay is added to
                               ;allow mac to complete

;following example illustrates a one-clock pipeline delay when
;trying to read an accumulator as source for bset:
tfr    a,b                    ;data ALU operation
bset   #3,b                   ;one clock delay is added to
                               ;allow tfr to complete

following example illustrates a way to find useful usage of
;the pipeline delay clock:
mac    x0,y0,a                ;data ALU operation
mac    x1,y1,b                ;insert a useful instruction
move  a,x:(r0)+             ;read accumulator A without
                               ;any time penalty

```

**Figure 3-11.** Pipeline Conflicts—Arithmetic Stall

### 3.6.2 Status Stall

A second interlock condition, named *status stall*, occurs during an attempt to read the Status Register (SR) if the preceding or the second preceding instruction is a Data ALU instruction or an accumulator read that updates the Scale (S) and Limit (L) condition codes in the SR. The hardware inserts two or one idle cycles (no op) accordingly, thereby guaranteeing the correctness of the result.

**Note:** Read Status Register implies a MOVE from SR. Bit manipulation instructions (for example, BSET) act on an SR bit. Program control instructions (for example, BSCLR) test for a bit in the SR.

**Figure 3-12** describes the cases in which the pipelined nature of the Data ALU generates a status stall.

```

;following example illustrates a two-clock pipeline delay when
;trying to read the status register as source for move:
mac    x0,y0,a                ;data ALU operation
move   sr,x:(r0)+            ;TWO clock delay is added to
                                ;allow mac to update SR

;following example illustrates a one-clock pipeline delay when
;trying to read the status register as source for bit
;manipulation instruction:
move   a,x:(r0)+            ;read full accumulator
nop
btst   #5,sr                ;ONE clock delay is added (and
                                ;not two) due to the previous nop

;following example illustrates a one-clock pipeline delay when
;trying to read the status register as source for program control
;instruction:
insert x0,y1,a                ;data ALU operation
bsclr  #5,sr,$ff00ff        ;ONE clock delay is added (and not
                                ;two) since bsclr is a two word
                                ;instruction

```

**Figure 3-12.** Pipeline Conflicts—Status Stall

### 3.6.2.1 Transfer Stall

A third interlock condition, transfer stall, occurs when the source Data ALU accumulator of the move portion of an instruction is identical to the destination Data ALU accumulator of the move portion of the preceding instruction. Identical accumulators for this matter are any combination of portions (including the full width) of the same Data ALU accumulator (for example, A1 and A, A2 and A0, etc.). The hardware inserts one idle cycle (no op), thereby guaranteeing the correctness of the result.

---

```
;following example illustrates a one-clock pipeline delay when  
;trying to read an accumulator that was written by the preceding  
;instruction:
```

```
move  y:(r1)+,a1           ;write into partial accumulator  
move  a2,x:(r0)+          ;one clock delay is added
```

```
;following example illustrates a way to find useful usage of  
;the pipeline delay clock:
```

```
move  y:(r1)+,a1           ;write into partial accumulator  
mac   x1,y1,b             ;insert a useful instruction  
move  a,x:(r0)+          ;no time penalty for this read
```

### Figure 3-13. Pipeline Conflicts—Transfer Stall

**Note:** A special case of interlock occurs when a 24-bit logic instruction is used and a write operation occurs concurrently to the EXT or the LSP of the same accumulator. The hardware inserts one idle cycle (no op), thereby guaranteeing the correctness of the result. An example of this case is:

```
or    x1,a    y1,a0
```

