3 DATA ARITHMETIC LOGIC UNIT

3.1 DATA ALU ARCHITECTURE

The Data ALU (see Figure 3-1) performs all the arithmetic and logical operations on data operands in the DSP56300 Core.

The Data ALU registers may be read or written over the XDB and the YDB as 24- or 48-bit operands. The source operands for the Data ALU, which may be 24, 48, or 56 bits, always originate from Data ALU registers. The results of all Data ALU operations are stored in an accumulator. A sixteen bit arithmetic mode of operation is available by setting the SA bit in the status register (SR).

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 an instruction per clock cycle. The destination of every arithmetic operation can be used as a source operand for the immediate following operation without penalty.

The components of the Data ALU are as follows:

- Four 24-bit input registers
- A parallel, fully pipelined multiply-accumulator unit (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

3.1.1 Data ALU Input Registers (X1, X0, Y1, Y0)

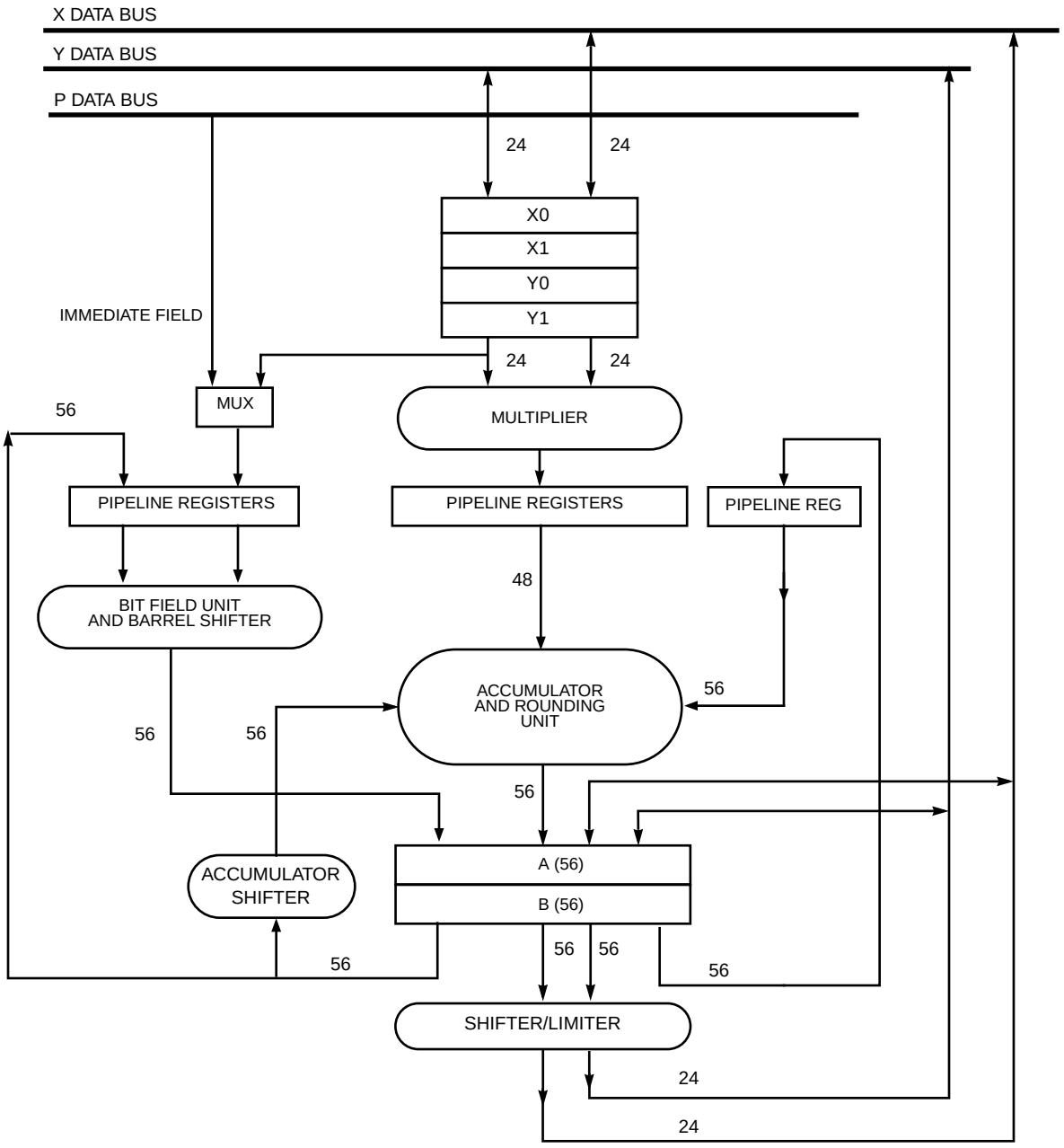
X1, X0, Y1, and Y0 are four 24-bit, general-purpose data registers. They can be treated as four independent 24-bit registers or as two 48-bit registers called X and Y, formed by concatenation of X1:X0 and Y1:Y0, respectively. X1 is the most significant word in X and Y1 is the most significant word in Y. The registers serve as input buffer registers between the XDB or 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 register contents are used by the current instruction. The registers may also be read back out to the appropriate data bus.

3.1.2 MAC Unit

The MAC unit comprise the main arithmetic processing unit of the DSP56300 Core and perform all of the calculations on data operands. In the case of arithmetic instructions, the unit 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 are provided on the MAC unit input to permit writing an input register, which is the source for a Data ALU operation in the same instruction. The input to the multiplier can only come 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 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 be simply truncated, or it can be rounded into the MSP. Rounding is performed if specified in the DSP instruction (e.g., the signed multiply-accumulate and round (MACR) instruction). The rounding performed is either convergent rounding (round-to-nearest-even) or two's-complement rounding. The type of rounding is specified by the rounding bit in the status register. The bit in the accumulator that is rounded is specified by the scaling mode bits in the status register.

Figure 3-1. Data ALU Block Diagram



It is possible to saturate the arithmetic unit's result going into the accumulator so that it would fit 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 status register (SR). The purpose of this mode is to provide for algorithms which do not recognize or cannot take advantage of the extension accumulator (EXT). For further details refer to Section 3.2.3.

3.1.3 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.

Reading the A or B accumulators over the XDB and YDB is protected against overflow by substituting a limiting constant for the data that is being transferred. The content of A or B is not affected should limiting occur; 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 have been shifted according to the scaling mode. Shifting and limiting will be 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 A0, A1, A2, B0, B1, or B2 are specified as 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 specified as 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 MSP (A1 or B1) portion of the accumulator is written with the word operand, the LSP (A0 or B0) portion is zero filled, and the EXT (A2 or B2) portion 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 is included in each accumulator register to support operation of the data shifter/limiter circuits. This test logic is used to detect overflows out of the data shifter so that the limiter can substitute one of several constants to minimize errors due to the overflow.

3.1.4 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 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

3.1.5 Bit Field Unit (BFU)

The bit field unit 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

3.1.6 Data Shifter/Limiter

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

3.1.7 Scaling

The data shifters (in the shifters/limiters unit), controlled by the scaling mode bits in the status register, are capable of shifting data one bit to the left (scale up) or one bit to the right (scale down) as well as passing the data unshifted (no scaling). 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 to be implemented in a regular fashion.

3.1.8 Limiting

In the DSP56300 Core, the Data ALU accumulators A and B have eight extension bits. Limiting will occur 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 can be represented without overflow in the destination operand size (i.e. 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 cannot be represented without overflow in the destination operand size, the data limiter will substitute a limited data value having maximum magnitude (saturated) and having the same sign as the source accumulator contents: $7FFFFF for 24-bit or $7FFFFF FFFFFF for 48-bit positive numbers, $800000 for 24-bit or $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 status register.

3.2 DATA ALU ARITHMETIC AND ROUNDING

3.2.1 Data Representation

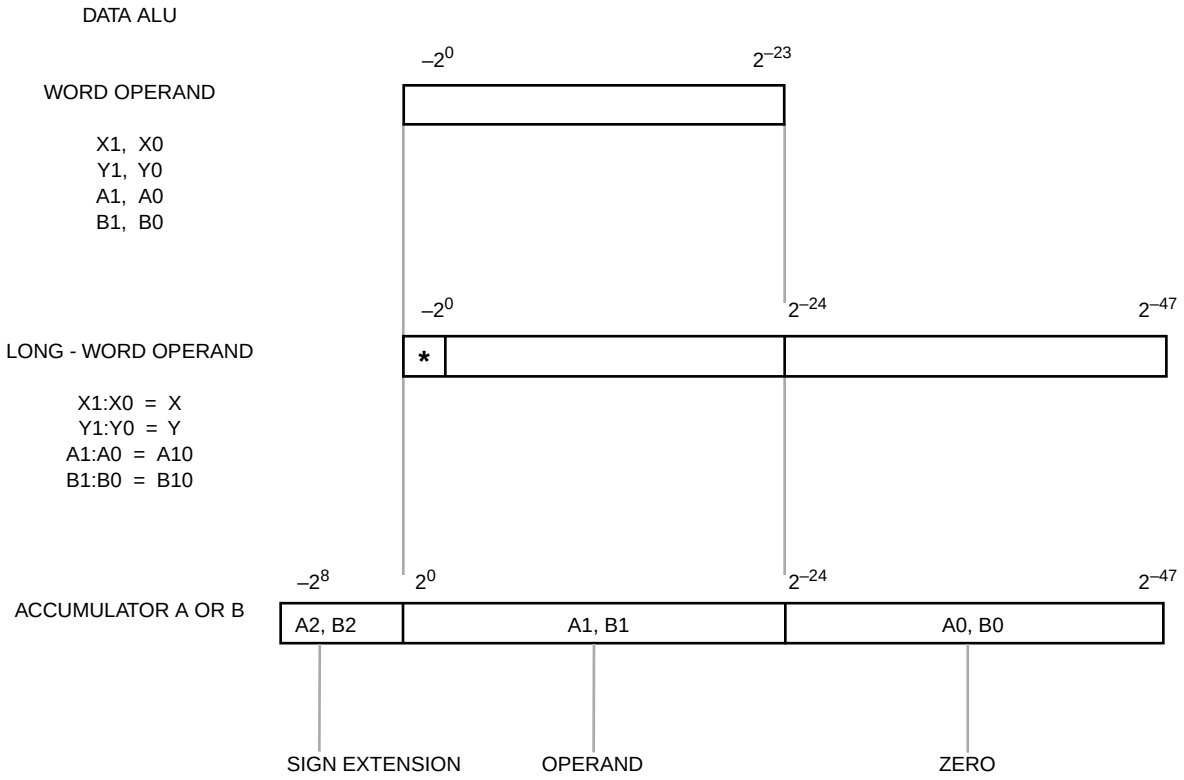
The DSP56300 Core uses a fractional data representation for all Data ALU operations. Figure 3-2 shows the bit weighting of words, long words, and accumulator operands for this representation. The decimal points are all aligned and are left justified.

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 $7FFFFF or $1-2^{-23}$ and the most positive long word is $7FFFFFFFFFFF 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 that can be used is approximately 256 and the most negative number is -256.

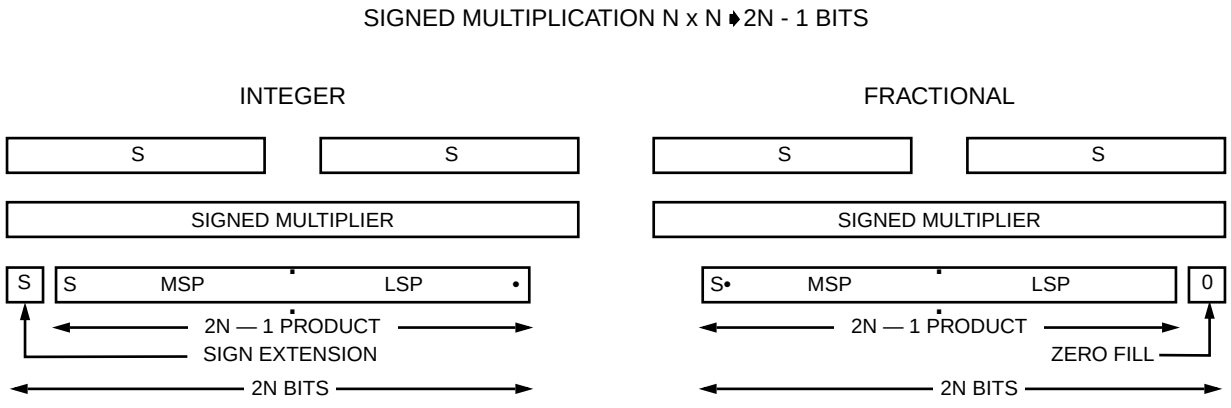
To maintain alignment of the binary point when a word operand is written to accumulator A or B, the operand is written to the most significant accumulator register (A1 or B1), and its MSB is automatically sign extended through the accumulator extension register (A2 or B2). The least significant accumulator 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 ± 1. To convert from an integer to a fractional number, the integer must be multiplied by a scaling factor so the result will always be between ± 1. The representation of integer and fractional numbers is the same if the numbers are added or subtracted but is different if the numbers are multiplied or divided. An example of two numbers multiplied together is given in Figure 3-3.

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 should be 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 should be right aligned, and the sign bit should be duplicated, to maintain integer representation. Since the DSP56300 Core incorporates a fractional array multiplier, it always aligns the 2N-1 significant product bits to the left. The user should be aware of this when multiplying integer numbers.

3.2.2 Rounding Modes

The DSP56300 Core's DATA ALU performs rounding of 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 S0 and S1 in the status register (SR). Two types of rounding are implemented: convergent rounding and two's complement rounding. The type of rounding is selected by the rounding mode bit (RM) in the EMR portion of the status register.

3.2.2.1 Convergent Rounding

This is the default rounding mode. Convergent rounding is also called round-to-nearest (even) number. The usual rounding method rounds up any value above one-half and rounds down any value below one-half. The question arises as to which way one-half should be rounded. If it is always rounded one way, the results will eventually be 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 status register, the rounding position is updated to reflect the alignment of the result when it will be put on the data bus. However, the contents of the register are not scaled.

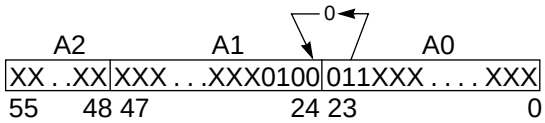
3.2.2.2 Two's Complement Rounding

When twos-complement rounding is selected by setting the rounding mode bit in the MR, all values equal or above one-half are rounded up and all values below one-half are rounded down. Therefore a small positive bias is introduced. Figure 3-4 shows the four cases for rounding a number in the A1 (or B1) register. If scaling is set in the status register, the rounding position is updated to reflect the alignment of the result when it will be put on the data bus. However, the contents of the register are not scaled.

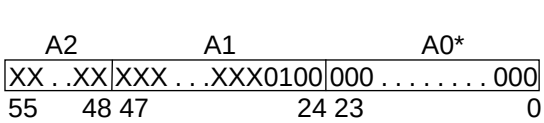
Figure 3-4. Convergent Rounding (no scaling)

CASE I: IF A0 < $800000 (1/2), THEN ROUND DOWN (ADD NOTHING)

BEFORE ROUNDING

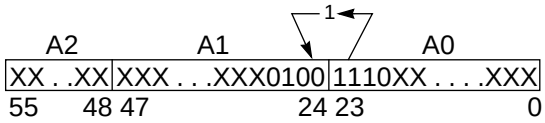


AFTER ROUNDING

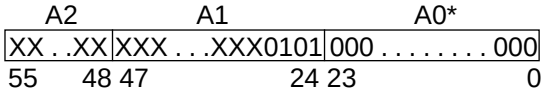


CASE II: IF A0 > $800000 (1/2), THEN ROUND UP (ADD 1 TO A1)

BEFORE ROUNDING

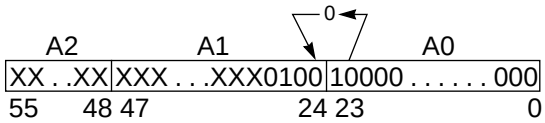


AFTER ROUNDING

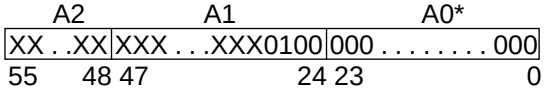


CASE III: IF A0 = $800000 (1/2), AND THE LSB OF A1 = 0, THEN ROUND DOWN (ADD NOTHING)

BEFORE ROUNDING

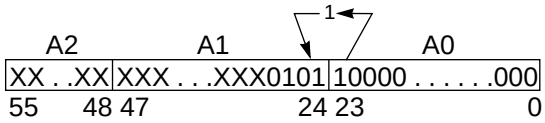


AFTER ROUNDING

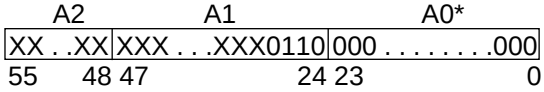


CASE IV: IF A0 = $800000 (1/2), AND THE LSB = 1, THEN ROUND UP (ADD 1 TO A1)

BEFORE ROUNDING



AFTER ROUNDING

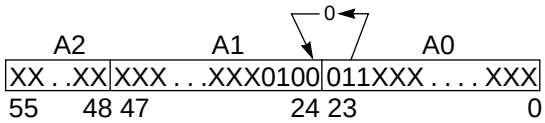


*A0 is always clear; performed during RND, MPYR, MACR

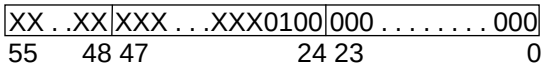
Figure 3-5. Two's Complement Rounding (no scaling)

CASE I: IF A0 < $800000 (1/2), THEN ROUND DOWN (ADD NOTHING)

BEFORE ROUNDING

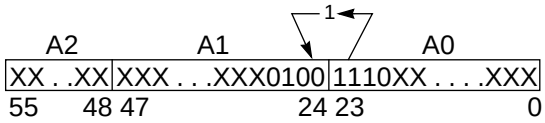


AFTER ROUNDING

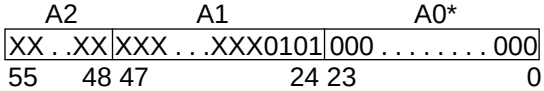


CASE II: IF A0 > $800000 (1/2), THEN ROUND UP (ADD 1 TO A1)

BEFORE ROUNDING

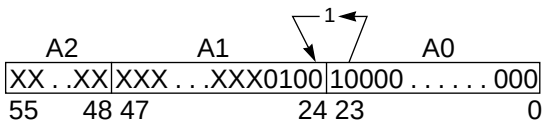


AFTER ROUNDING

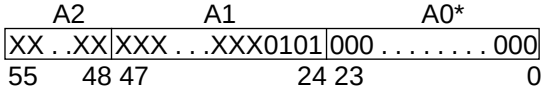


CASE III: IF A0 = $800000 (1/2), AND THE LSB OF A1 = 0, THEN ROUND UP (ADD 1 TO A1)

BEFORE ROUNDING

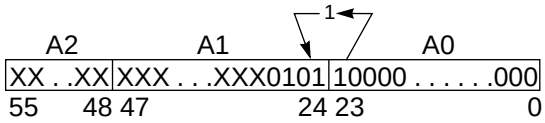


AFTER ROUNDING

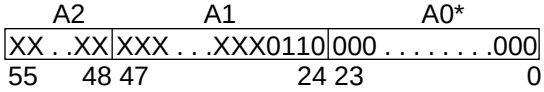


CASE IV: IF A0 = $800000 (1/2), AND THE LSB OF A1 = 1, THEN ROUND UP (ADD 1 TO A1)

BEFORE ROUNDING



AFTER ROUNDING



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

3.2.3 Arithmetic Saturation Mode

By setting the Arithmetic Saturation Mode (SM) bit in the status register (SR), the arithmetic unit's result is limited to 48 bits (MSP and LSP). The highest dynamic range of the machine is then limited to 48 bits. The purpose of this bit is to provide a saturation mode for algorithms which do not recognize or cannot take advantage of the extension accumulator.

The arithmetic saturation logic operates by checking three 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 7FFFFF FFFFFF
0	1	0	$00 7FFFFF FFFFFF
0	1	1	$00 7FFFFF FFFFFF
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 $007FFFFFFFFFFF and $FF800000000000 are not affected by the scaling mode. In the same way, the rounding of the saturation constant (during MPYR, MACR, RND) is independent of the scaling mode: $007FFFFFFFFFFF is rounded to $007FFFFF000000 and $FF800000000000 to $FF800000000000.

When in Arithmetic Saturation Mode, the Overflow Bit (V bit) in the status register 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 status register is set when an arithmetic saturation occurs.

Caution: The arithmetic saturation mode is **ALWAYS** disabled during the execution of the following instructions: TFR, Tcc, DMACsu, DMACuu, MACsu, MACuu, MPYsu, MPYuu, CMPU, and all BIT FIELD UNIT operations (see 3.1.5). If the result of these instructions should be saturated, a MOVE A,A (or B,B) instruction must be added following the original instruction (provided no scaling is set). However, the "V" bit of the status register will never be set by the arithmetic saturation of the accumulator during the MOVE A,A (or B,B). Only the "L" bit will then be set.

3.2.4 Multiprecision Arithmetic Support

A set of Data ALU operations is provided in order to facilitate multi-precision multiplications. When these instructions are used, the multiplier accepts some combinations of signed two's-complement format and unsigned format. These instructions are:

- **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

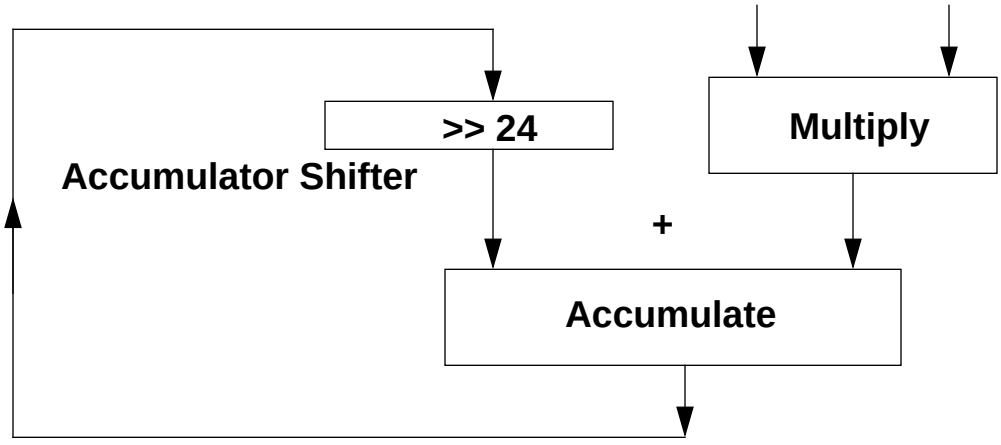
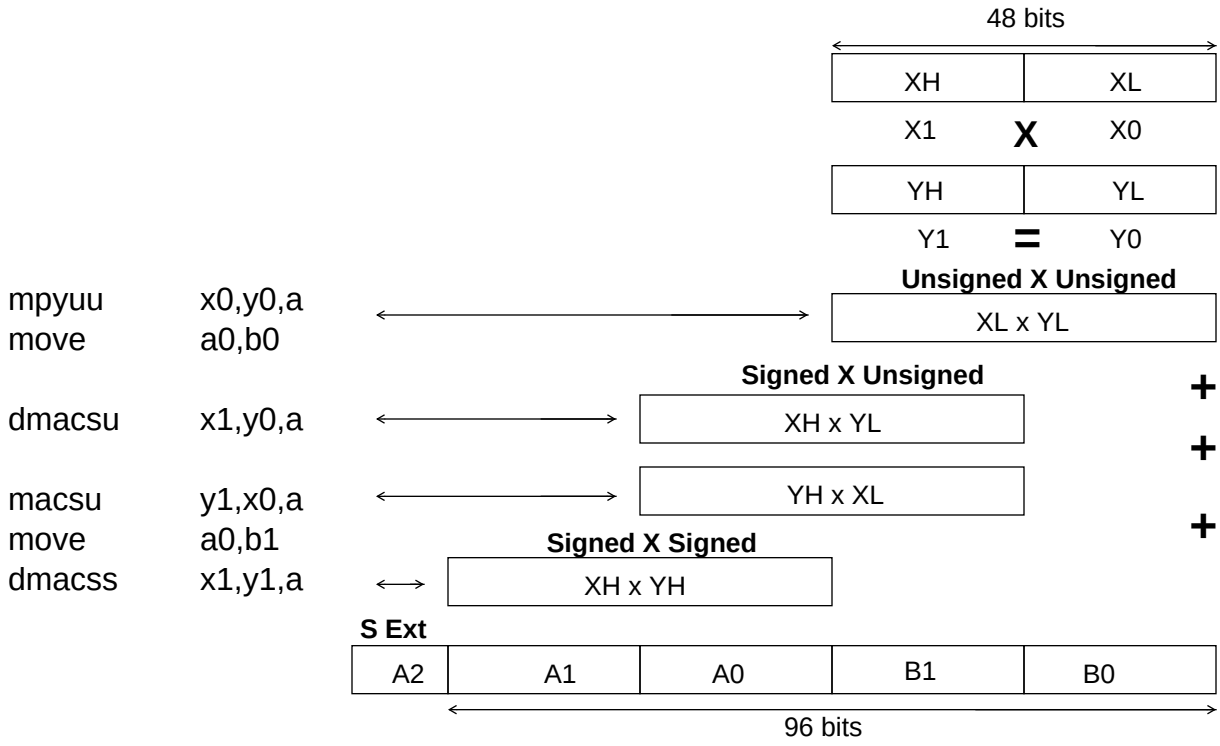


Figure 3-7 illustrates the use of these instructions in the case of a double precision multiplication. The signed x signed operation is used to multiply or multiply-accumulate the two upper, signed, portions of two signed double precision numbers. The unsigned x signed operation is used to multiply or multiply-accumulate the upper, signed, portion of one double precision number with the lower, unsigned, portion of the other double precision number. The unsigned x unsigned operation is used to multiply or multiply-accumulate 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



3.2.4.1 Double Precision Multiply Mode

To support existing 56K code, double precision multiply can also be performed by a double precision algorithm which uses four multiply operations, after entering a dedicated “Double Precision Multiply” mode. The mode is entered by setting bit 14 (DM) of the Status Register (bit 6 of the MR register). The mode is disabled by clearing the DM bit.

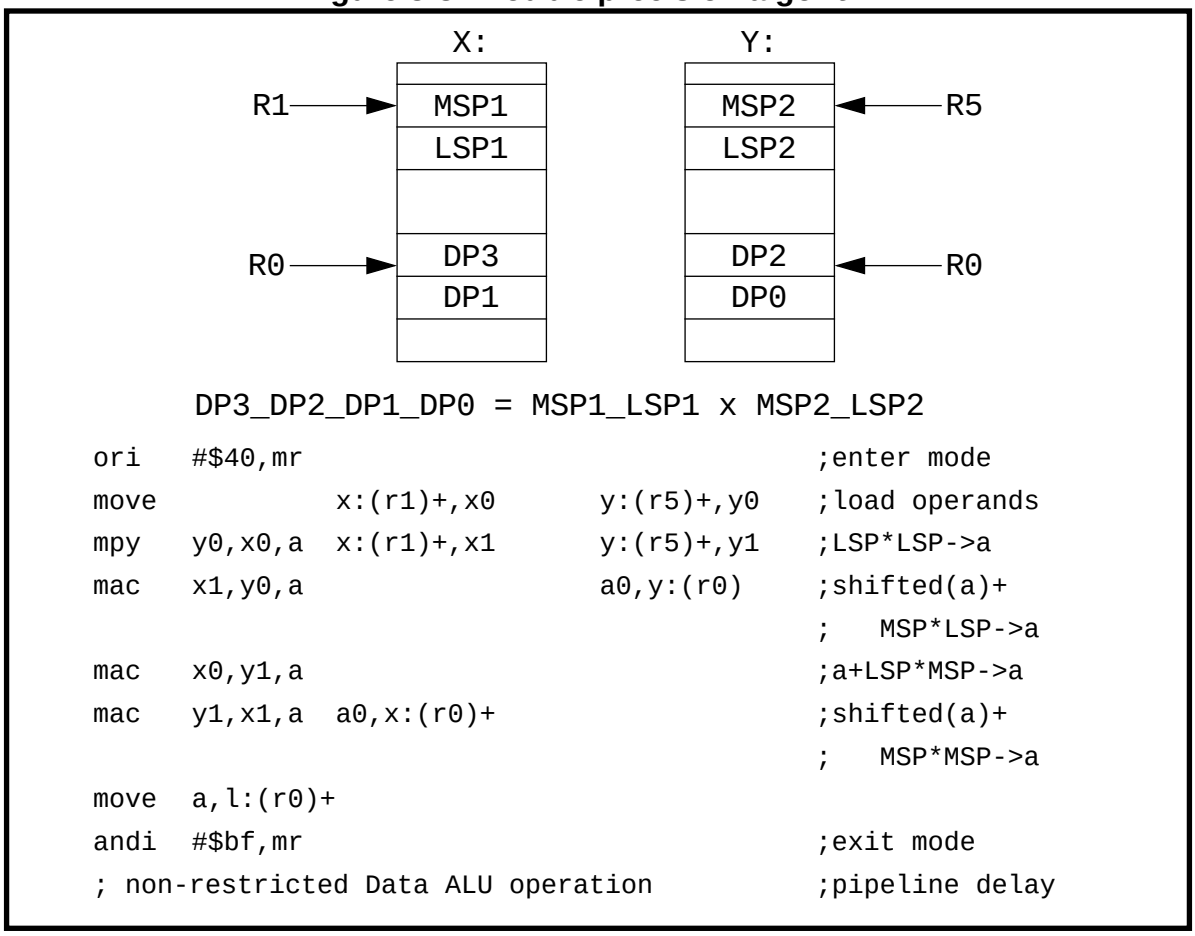
The algorithm is shown in Figure 3-8. The ORI instruction sets the DM mode bit in the MR register, but due to the instruction execution pipeline the Data ALU enters the Double Precision Multiply mode only after one cycle. The ANDI instruction clears the DM mode bit in the MR register, but due to the instruction execution pipeline the Data ALU leaves the mode after one cycle; the ANDI instruction should not be immediately followed by a restricted Data ALU instruction, to allow for the pipeline delay.

While in Double Precision Multiply mode, the behavior of the four specific operations listed in the double precision algorithm is modified. Therefore these operations (with those specific register combinations) should not be used, while in Double Precision Multiply mode, for any other purpose but for the double precision multiply algorithm. All other Data ALU operations (or the four listed operations but with other register combination) may not be used.

The double precision multiply algorithm uses the Y0 register at all stages. Therefore Y0 should not be changed when running the double precision multiply algorithm. If the use of

the Data ALU is required in an interrupt service routine, Y0 should be saved together with other Data ALU registers to be used, and should be restored before leaving the interrupt routine.

Figure 3-8. Double precision algorithm



DP3_DP2_DP1_DP0 = MSP1_LSP1 x MSP2_LSP2

```
ori   #$40,mr                                    ;enter mode
move            x:(r1)+,x0      y:(r5)+,y0       ;load operands
mpy   y0,x0,a   x:(r1)+,x1      y:(r5)+,y1       ;LSP*LSP->a
mac   x1,y0,a                   a0,y:(r0)        ;shifted(a)+
                                                 ;   MSP*LSP->a
mac   x0,y1,a                                    ;a+LSP*MSP->a
mac   y1,x1,a   a0,x:(r0)+                       ;shifted(a)+
                                                 ;   MSP*MSP->a
move  a,l:(r0)+
andi  #$bf,mr                                    ;exit mode
; non-restricted Data ALU operation              ;pipeline delay
```

3.2.5 Block Floating Point FFT Support

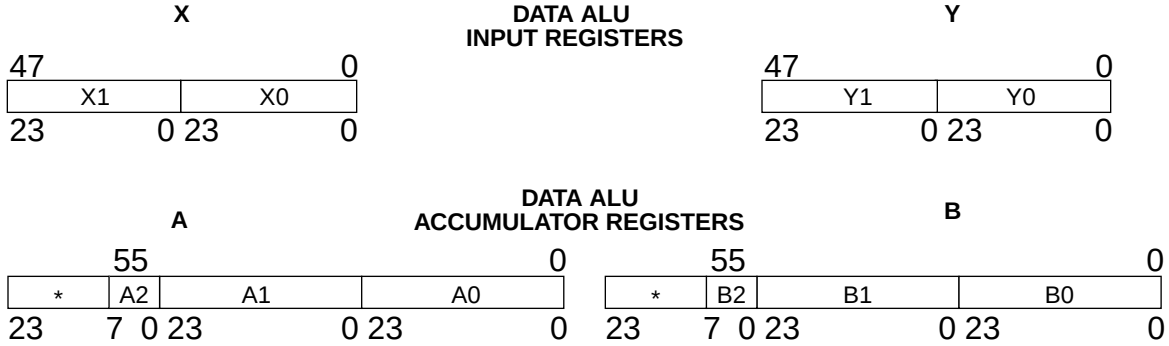
Block Floating Point FFT operation requires the early detection of data growth between FFT butterfly passes. If data growth is detected, suitable down scaling must be applied to ensure that no overflow will occur during the next butterfly calculation pass. The total scaling applied is the block exponent 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 status register. The “FFT scaling bit” S (bit 7) of the status register is set upon moving a result from accumulator A or B to the XDB or YDB bus (during an accumulator to memory or accumulator to register move) and will remain set until explicitly cleared, that is, the “S” bit is a “sticky” bit.

3.3 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. DSP56300 Core Programming Model



*Read as sign extension bits, written as don't care.

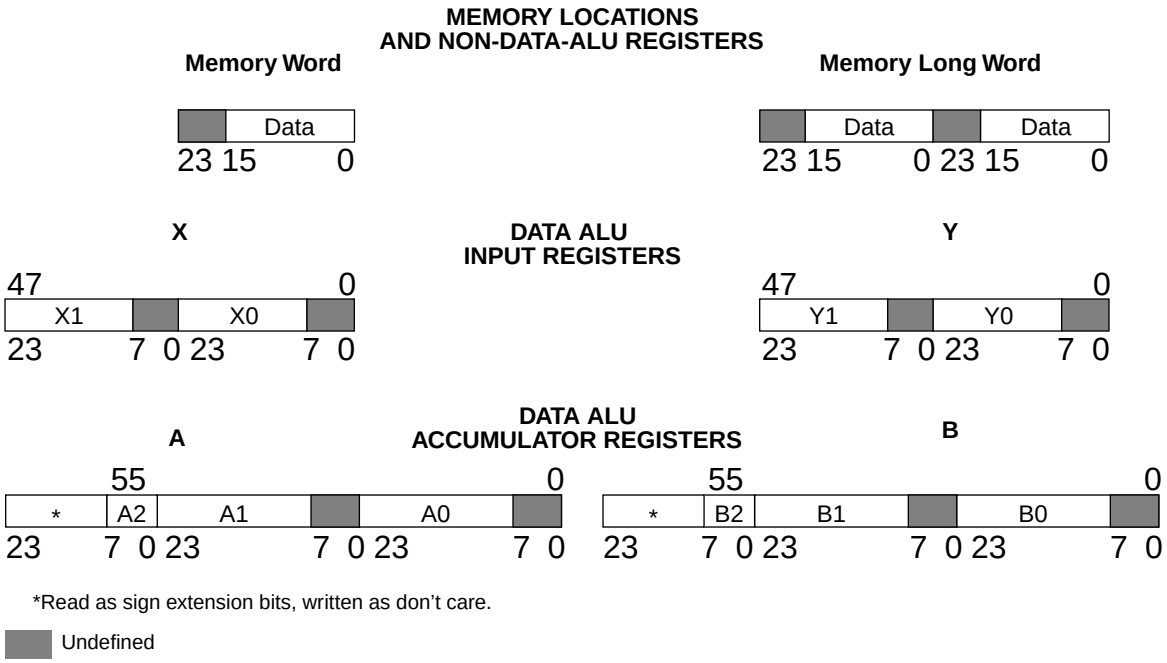
3.4 SIXTEEN BIT ARITHMETIC MODE

Setting the SA bit in the status register (SR) enables the Sixteen Bit Arithmetic mode of operation. The 16 bit data is right aligned in the 24 bit memory word, that is in the 16 least significant bits of the 24 bit word. The user may use 16 bit wide data memories with either leaving the 8 most significant bits unconnected, or tieing these bits to GND.

In the Sixteen Bit Arithmetic mode of operation the source operands can be 16, 32 or 40 bit. The numerical results have 40 bits accuracy. These 40 bits are composed of 16 bit LSP, 16 bit MSP and 8 bit EXT.

Figure 3-10 shows the bit positions in the memory and Data ALU registers when in Sixteen Bit Arithmetic mode.

Figure 3-10. Sixteen Bit Arithmetic Mode Data Organization



Note 1: When switching to and from sixteen bit arithmetic mode, for two instruction cycles, no arithmetic instruction or a move instruction should be performed.
Note 2: The programmer should be cautious about exchanging data between 16-bit arithmetic mode and 24-bit arithmetic mode via write-read operations on data-ALU registers and accumulators. Since the write operations in 16 bit arithmetic mode corrupt the information in the least significant bytes of the registers or accumulators, these registers or accumulator should not be used as 24-bit data without some processing.

3.4.1 Moves in sixteen bit arithmetic mode

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

3.4.1.1 Moves into registers or accumulators

When moving XDB or YDB into a full Data ALU accumulator (A or B) the 16 LS bits of the bus will be placed in bits 32-47 of the accumulator (16 MS bits of A1 or B1). Bits 8-23 of the accumulator (16 MS bits of A0 or B0) will be cleared and the EXT of the accumulator (A2 or B2) will be loaded with sign extension.

When moving XDB and YDB (48 bits) into a full Data ALU accumulator (A or B) the 16 LS bits from XDB will be placed in bits 32-47 of the accumulator (16 MS bits of A1 or B1). The

16 LS bits from YDB will be placed in bits 8-23 of the accumulator (16 MS bits of A0 or B0). The EXT of the accumulator (A2 or B2) will be loaded with sign extension.

When moving XDB or YDB into a register (X0, X1, Y0 or Y1) or partial accumulator (A0, A1, B0 or B1) the 16 LS bits of the bus will be loaded into the 16 MS bits of the destination register. No other portion of the accumulator will be affected.

When moving XDB or YDB into the accumulator extension register (A2 or B2) the 8 LS bits of the bus will be loaded into the 8 LS bits of the destination register and the 16 MS bits of the bus will not be used. The remaining parts of the accumulator will not be affected.

When moving XDB and YDB into a 48-bit register (X or Y) or partial accumulator (A10 or B10) the 16 LS bits of XDB bus will be loaded into the 16 MS bits of the MSP (X1, Y1, A1 or B1) and the 16 LS bits of YDB bus will be loaded into the 16 MS bits of the LSP (X0, Y0, A0 or B0). The EXT part of the accumulator (A2 or B2) will not be affected.

3.4.1.2 Moves from registers or accumulators

When moving a partial accumulator (A0, A1, B0 or B1) to XDB or YDB, the 16 MS bits of the source will be transferred to the 16 LS bits of the bus with 8 zeros in the MS bits. No scaling or limiting will be performed. When the source is the accumulator extension register (A2 or B2) it occupies the 8 LS bits of the bus while the next 16 bits are sign extension of bit number 7.

When moving a partial accumulator (A10 or B10) to XDB and YDB, the 16 MS bits of the MSP of the source (A1 or B1) will be transferred to the 16 LS bits of XDB with 8 zeros in the MS bits, while the 16 MS bits of the LSP of the source (A0 or B0) will be transferred to the 16 LS bits of YDB with 8 zeros in the MS bits. No scaling or limiting will be performed.

When moving a full Data ALU accumulator (A or B) to XDB or YDB, scaling and limiting will be performed, and then the 16-bit scaled and limited word will be placed on the bus LS bits and sign extension will be placed on the bus 8 MS bits.

When moving a full Data ALU accumulator (A or B) to XDB and YDB, scaling and limiting will be performed, and then the 16 MS bits of the 32-bit scaled and limited double word will be placed on XDB 16 LS bits and sign extension will be placed on the bus 8 MS bits. The 16 LS bits of the 32-bit scaled and limited double word will be placed on YDB 16 LS bits with 8 zeros on the bus 8 MS bits.

When moving a register (X0, X1, Y0 or Y1) to XDB or YDB, the 16 MS bits of the source will be transferred to the 16 LS bits of the bus with 8 zeros in the MS bits.

When moving a 48-bit register (X or Y) to XDB and YDB, the 16 MS bits of the high register (X1 or Y1) will be placed on XDB 16 LS bits and 8 zeroes will be placed on the bus 8 MS bits. The 16 LS bits of the low register (X0 or Y0) will be placed on YDB 16 LS bits with 8 zeros on the bus 8 MS bits.

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, then the value placed on the 8 MS bits of the XDB or YDB will be undefined.

3.4.1.3 Short Immediate moves

When an Immediate Short Data MOVE is performed while in Sixteen Bit Arithmetic mode of operation 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 8 LS bits 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 therefore stored in bits 47-40 of the accumulator, or bits 23-16 of a register (which correspond to the 8 MS bits of a 16-bit number).

3.4.1.4 Scaling and Limiting

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

During the Sixteen Bit Arithmetic mode of operation the limiting will be affected as described below: the maximum positive value will be $007FFF ($007FFF00FFFF for double precision). The maximum negative value will be $008000 ($008000000000 for double precision).

3.4.2 Sixteen bit arithmetic

When reading an operand from a Data ALU register or accumulator to the arithmetic unit, the 8 least significant bits of the 24 bit word, are ignored, i.e. read as zeros. The arithmetic unit will force 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 when in sixteen bit arithmetic mode is not affected, except for the following:

1. The operands and results width (16/32/40 instead of 24/48/56).
2. The rounding, if specified by the operation, will be performed on the most significant bit of the 16-bit LS portion of the result, that is on the bit corresponding to bit number twenty-three of A0/B0 (the scaling mode will affect this position accordingly). See RND instruction for details.
3. The arithmetic saturation detection is unchanged, but the saturated values change to $007FFF00FFFF00 and $FF800000000000.
4. The carry bit C will be added/subtracted, in ADC/SBC instructions, to the least significant bit of the 16-bit LSP.

5. Logic operations will only affect the 16-bit wide word.
6. Rotation in rotate instructions will be performed on a 16-bit wide word.
7. The possible normalization range changes, thus affecting the CLB instruction.
8. DMAC instruction will perform a 16-bit arithmetic right shift of the accumulator before accumulation.
9. Double Precision Multiplication Algorithm, with the Double Precision Multiply Mode bit is set, will not be supported.
10. The bit parsing instructions (MERGE, EXTRACT, EXTRACTU and INSERT) are affected by the sixteen bit arithmetic mode so as to perform on the appropriate bit positions of the sixteen bit data. In INSERT the user has to update the offset by adding a bias value of 16. For further details refer to the specific instruction details in appendix A.
11. 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, i.e. bit number 0 is the least significant bit and bit number 15 is the most significant bit. Bit numbers over 15 should not be used.

3.5 PIPELINE CONFLICTS

The Data ALU is fully pipelined and every instruction takes two clock cycles to complete. However a new instruction can be started on every clock cycle and a new result is produced on every clock cycle thus yielding an effective execution rate of an instruction per clock cycle. There are no pipeline dependencies when using the result of the Data ALU as source operand for the immediate following Data ALU instruction. Nevertheless, Data ALU operations can produce pipeline conflicts as described in the following paragraphs.

Since every Data ALU instruction takes two clock cycles to complete, an interlock condition occurs when trying to read an accumulator (or parts of an accumulator) while the preceding instruction was a Data ALU instruction that specified that same accumulator as the destination. This interlock condition, named “arithmetic stall”, is detected in hardware and an idle cycle (no op) is inserted, thereby the correctness is guaranteed. The user can optimize his code by inserting a useful instruction before the read instruction. Figure 3-13 describes the cases in which the pipelined nature of the Data ALU generates arithmetic stall cases.

Figure 3-11. Pipeline Conflicts - 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
```

A second interlock condition, named “status stall”, occurs when trying to read the status register (SR) while the preceding or the second preceding instruction was a Data ALU instruction or an accumulator read (which updates the S and L condition codes in the status register). The hardware will insert two or one idle cycles (no op) accordingly, thereby the correctness is guaranteed. Notice that “read status register” implies a MOVE status register, Bit Manipulation Instructions (e.g. BSET) on a status register bit, or Program Control Instructions (e.g. BSCLR) which test for a bit in the status register. Figure 3-12 describes the cases in which the pipelined nature of the Data ALU generates stall interlock cases.

Figure 3-12. Pipeline Conflicts - 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
```

A third interlock condition, named “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, e.g. A1 and A, A2 and A0 etc. The hardware will insert one idle cycle (no op) thereby the correctness is guaranteed.

Figure 3-13. Pipeline Conflicts - Transfer stall

```
;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
```

Note: A special case of interlock occurs when using a 24 bit logic instruction and writing concurrently to the EXT or the LSP of the same accumulator. The hardware will insert one idle cycle (no op), thereby the correctness is guaranteed. For example:

or x1,a y1,a0