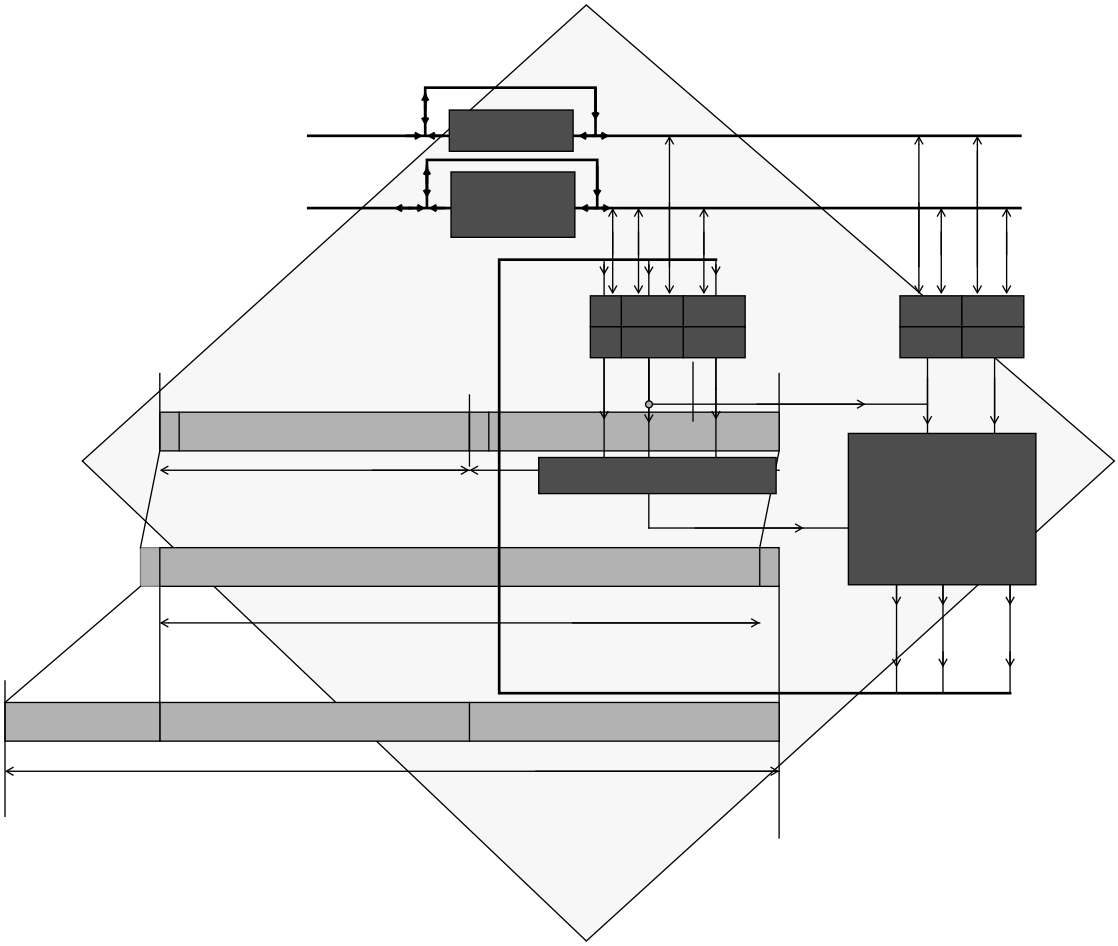
SECTION 3
DATA ARITHMETIC LOGIC UNIT



SECTION CONTENTS

3.1 DATA ARITHMETIC LOGIC UNIT

This section describes the operation of the Data ALU registers and hardware. It discusses data representation, rounding, and saturation arithmetic used within the Data ALU, and concludes with a discussion of the programming model.

3.2 OVERVIEW AND DATA ALU ARCHITECTURE

As described in Section 2, The DSP56K family central processing module is composed of three execution units that operate in parallel. They are the Data ALU, address generation unit (AGU), and the program control unit (PCU) (see Figure 3-1). These three units are register oriented rather than bus oriented and interface over the system buses with memory and memory-mapped I/O devices.

The Data ALU (see Figure 3-2) is the first of these execution units to be presented. It balances speed with the capability to process signals that have a wide dynamic range and performs all arithmetic and logical operations on data operands.

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.

The 24-bit data words provide 144 dB of dynamic range. This range is sufficient for most real-world applications since the majority of data converters are 16 bits or less – and certainly not greater than 24 bits. The 56-bit accumulator inside the Data ALU provides 336 dB of internal dynamic range so that no loss of precision will occur due to intermediate processing. Special circuitry handles data overflows and roundoff errors.

The Data ALU can perform any of the following operations in a single instruction cycle: multiplication, multiply-accumulate with positive or negative accumulation, convergent rounding, multiply-accumulate with positive or negative accumulation and convergent rounding, addition, subtraction, a divide iteration, a normalization iteration, shifting, and logical operations.

The components of the Data ALU are:

- Four 24-bit input registers
- A parallel, single-cycle, nonpipelined multiply-accumulator/logic unit (MAC)
- Two 48-bit accumulator registers
- Two 8-bit accumulator extension registers
- An accumulator shifter
- Two data bus shifter/limiter circuits

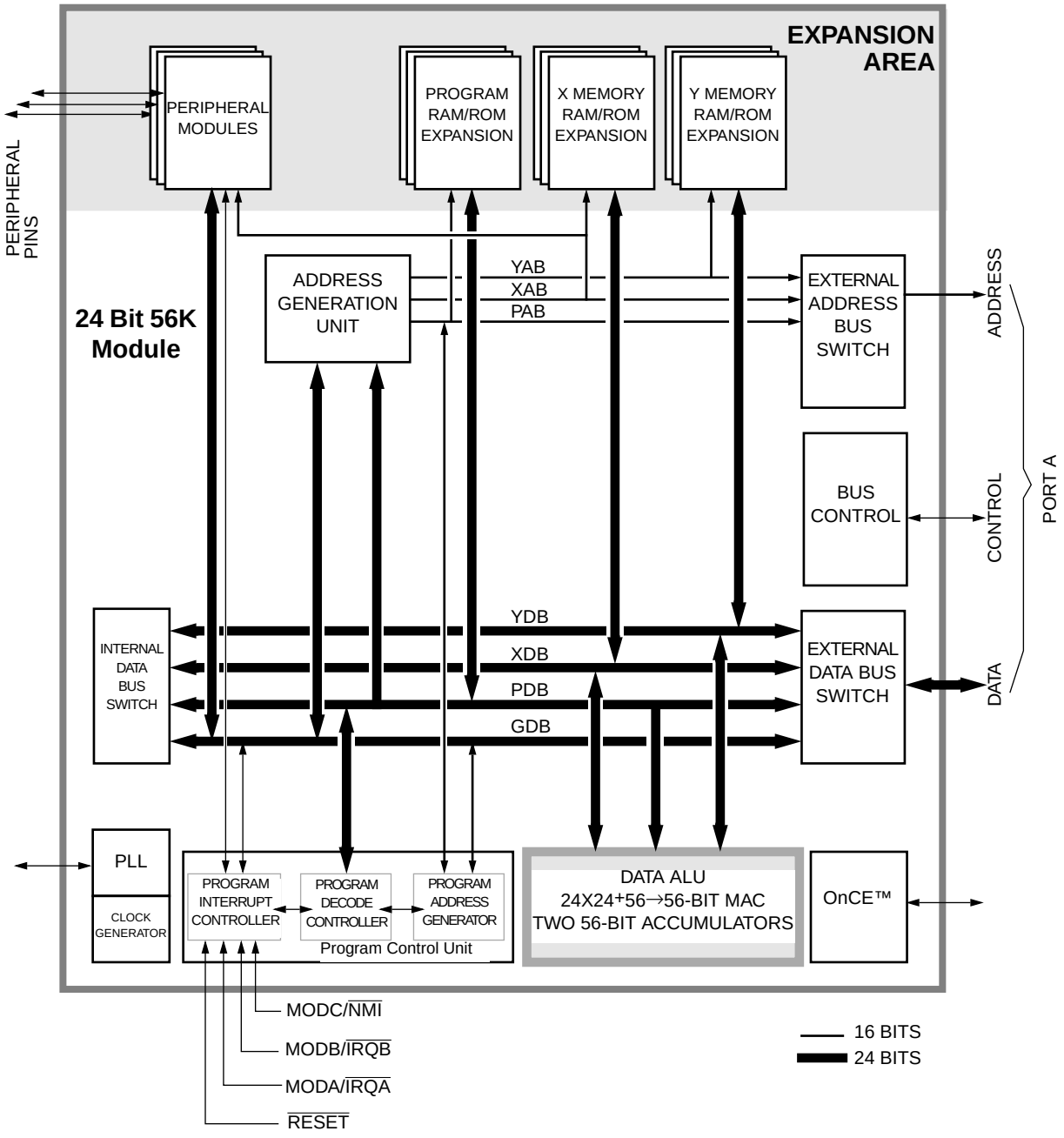


Figure 3-1 DSP56K Block Diagram

The following paragraphs describe each of these components and provide a description of data representation, rounding, and saturation arithmetic.

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, developed by concatenating 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. They act as Data ALU source operands and allow new operands to be loaded for the next instruction while the current instruction uses the

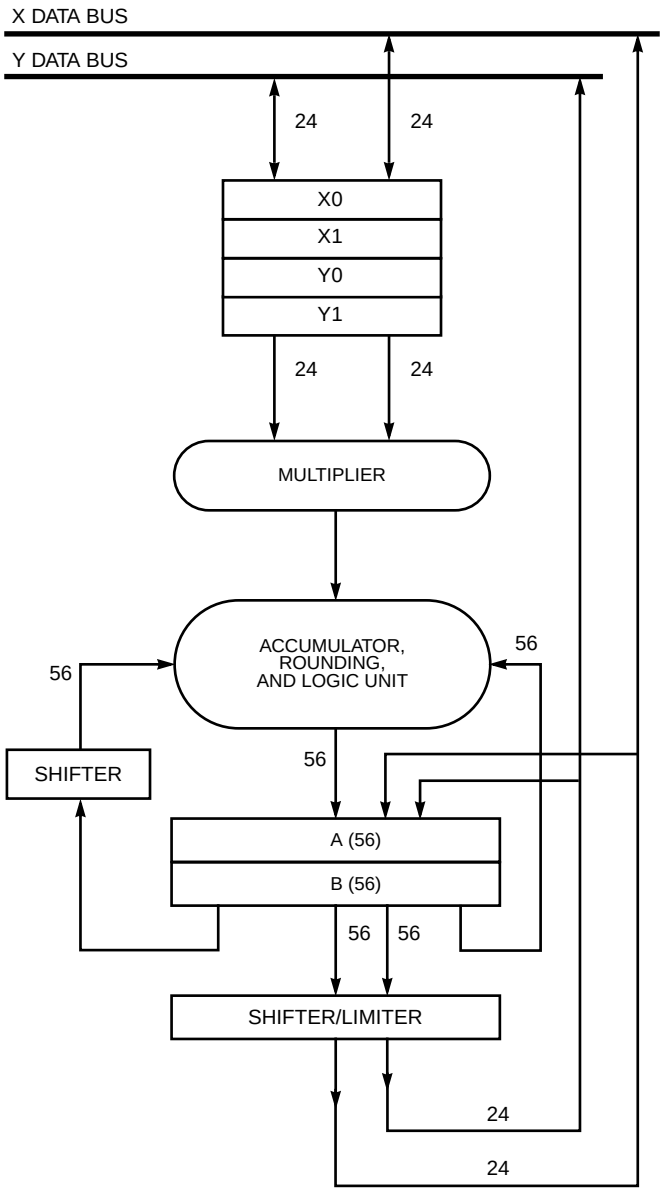


Figure 3-2 Data ALU

register contents. The registers may also be read back out to the appropriate data bus to implement memory-delay operations and save/restore operations for interrupt service routines.

3.2.2 MAC and Logic Unit

The MAC and logic unit shown in Figure 3-3 conduct the main arithmetic processing and perform all calculations on data operands in the DSP.

For arithmetic instructions, the unit accepts up to three input operands and outputs one 56-bit result in 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 Data ALU inputs and outputs. Latches on the MAC unit input permit writing an input register which is the source for a Data ALU operation in the same instruction.

The arithmetic unit contains a multiplier and two accumulators. The input to the multiplier can only come from the X or Y registers (X1, X0, Y1, Y0). The multiplier executes 24-bit x 24-bit, parallel, twos-complement fractional multiplies. 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 (see Figure 3-3). An 8-bit adder, which acts as an extension accumulator for the MAC array, accommodates overflow of up to 256 and allows the two 56-bit accumulators to be added to and subtracted from each other. The extension adder output is the EXT portion of the MAC unit output. This multiply/accumulate operation is not pipelined, but is a single-cycle operation. If the instruction specifies a multiply without accumulation (MPY), the MAC clears the accumulator and then adds the contents to the product.

In summary, the results of all arithmetic instructions are valid (sign-extended and zero-filled) 56-bit operands in the form of EXT:MSP:LSP (A2:A1:A0 or B2:B1:B0). 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 (using convergent rounding) into the MSP.

Convergent rounding (round-to-nearest) is performed when the instruction (for example, the signed multiply-accumulate and round (MACR) instruction) specifies adding the multiplier’s product to the contents of the accumulator. The scaling mode bits in the status register specify which bit in the accumulator shall be rounded.

The logic unit performs the logical operations AND, OR, EOR, and NOT on Data ALU registers. It is 24 bits wide and operates on data in the MSP portion of the accumulator. The LSP and EXT portions of the accumulator are not affected.

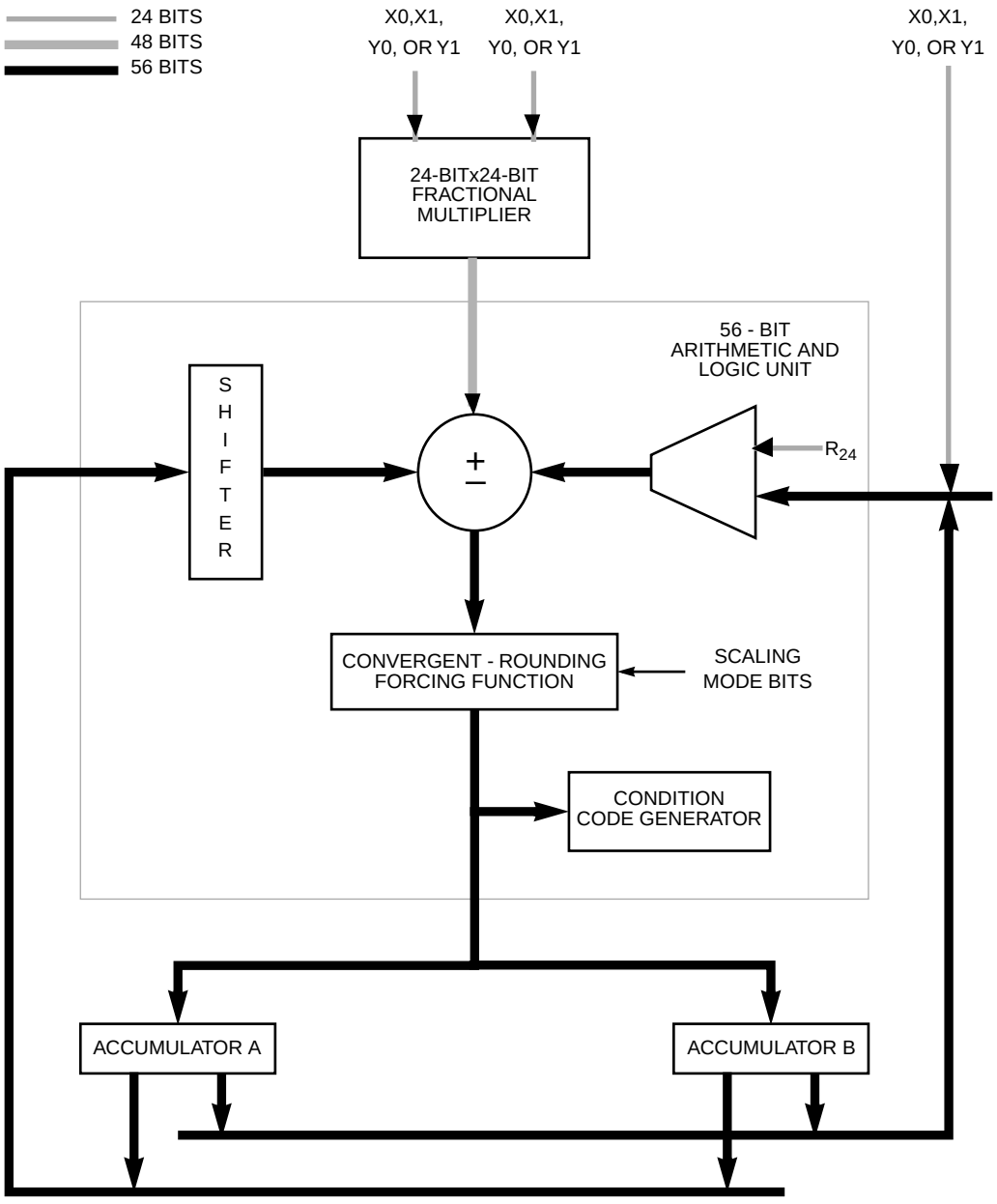
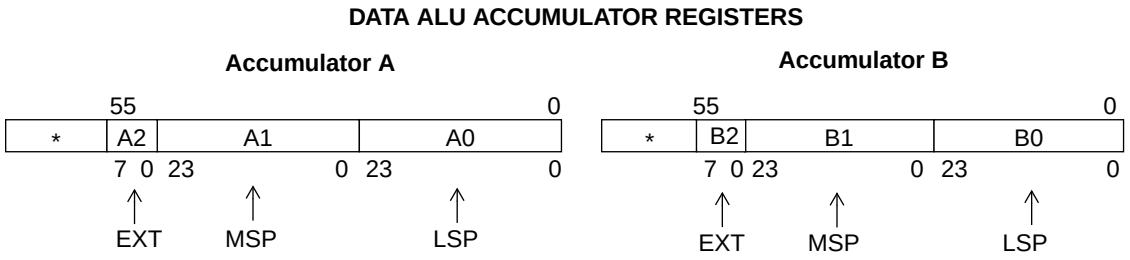


Figure 3-3 MAC Unit

3.2.3 Data ALU A and B Accumulators

The Data ALU features two general-purpose, 56-bit accumulators, A and B. Each consists of three concatenated registers (A2:A1:A0 and B2:B1:B0, respectively). The 8-bit sign extension (EXT) is stored in A2 or B2 and is used when more than 48-bit accuracy is needed; the 24-bit most significant product (MSP) is stored in A1 or B1; the 24-bit least



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

Figure 3-4 DATA ALU Accumulator Registers

significant product (LSP) is stored in A0 or B0 as shown in Figure 3-4.

Overflow occurs when a source operand requires more bits for accurate representation than are available in the destination. The 8-bit extension registers offer protection against overflow. In the DSP56K chip family, the extreme values that a word operand can assume are - 1 and + 0.9999998. If the sum of two numbers is less than - 1 or greater than + 0.9999998, the result (which cannot be represented in a 24 bit word operand) has underflowed or overflowed. The 8-bit extension registers can accurately represent the result of 255 overflows or 255 underflows. Whenever the accumulator extension registers are in use, the V bit in the status register is set.

Automatic sign extension occurs when the 56-bit accumulator is written with a smaller operand of 48 or 24 bits. A 24-bit operand is written to the MSP (A1 or B1) portion of the accumulator, the LSP (A0 or B0) portion is zero filled, and the EXT (A2 or B2) portion is sign extended from MSP. A 48-bit operand is written into the MSP:LSP portion (A1:A0 or B1:B0) of the accumulator, and the EXT portion is sign extended from MSP. No sign extension occurs if an individual 24-bit register is written (A1, A0, B1, or B0).When either A or B is read, it may be optionally scaled one bit left or one bit right for block floating-point arithmetic. Sign extension can also 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).

Overflow protection occurs when the contents of A or B are transferred over the XDB and YDB by substituting a limiting constant for the data. Limiting does not affect the content of A or B – only the value transferred over the XDB or YDB is limited. This overflow protection occurs after the contents of the accumulator has been shifted according to the scaling mode. Shifting and limiting occur only when the entire 56-bit A or B accumulator is specified as the source for a parallel data move over the XDB or YDB. When individual registers A0, A1, A2, B0, B1, or B2 are specified as the source for a parallel data move,

shifting and limiting are not performed.

3.2.4 Accumulator Shifter

The accumulator shifter (see Figure 3-3) 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)
- 1-Bit Left Shift (Arithmetic or Logical) ASL, LSL, ROL
- 1-Bit Right Shift (Arithmetic or Logical) ASR, LSR, ROR
- Force to zero

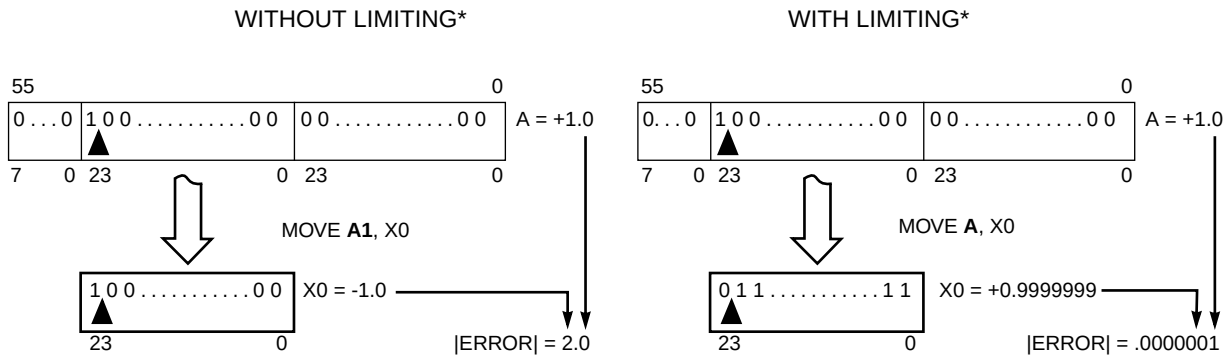
3.2.5 Data Shifter/Limiter

The data shifter/limiter circuits (see Figure 3-3) provide special post-processing on data read from the Data ALU A and B accumulators 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.

3.2.5.1 Limiting (Saturation Arithmetic)

The A and B accumulators serve as buffer registers between the MAC unit and the XDB and/or YDB. They act both as Data ALU source and destination operands.Test logic exists in each accumulator register to support the 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. This process is called saturation arithmetic

The Data ALU A and B accumulators 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. If the contents of the selected source accumulator can be represented without overflow in the destination operand size (i.e., accumulator extension register not in use), the data limiter is disabled, and the operand is not modified. If 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 with maximum magnitude (saturated) and with 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 saturation arithmetic. The value in the accumulator register is not shifted and can be reused within the Data ALU. When limiting does occur, a flag is set and latched in the status register.Two limiters allow two-word operands to be limited independently in the same instruction cycle. The two data limiters can also be com-



* Limiting automatically occurs when the 56 - bit operands A or B (not A2, A1, A0, B2, B1, or B0) are read. The contents of A or B are **NOT** changed.

Figure 3-5 Saturation Arithmetic

bined to form one 48-bit data limiter for long-word operands.

For example, if the source operand were 01.100 (+ 1.5 decimal) and the destination register were only four bits, the destination register would contain 1.100 (- 1.5 decimal) after the transfer, assuming signed fractional arithmetic. This is clearly in error as overflow has occurred. To minimize the error due to overflow, it is preferable to write the maximum ("limited") value the destination can assume. In the example, the limited value would be 0.111 (+ 0.875 decimal), which is clearly closer to + 1.5 than - 1.5 and therefore introduces less error.

Figure 3-5 shows the effects of saturation arithmetic on a move from register A1 to register X0. The instruction "MOVE A1,X0" causes a move without limiting, and the instruction "MOVE A,X0" causes a move of the same 24 bits with limiting. The error without limiting is 2.0; whereas, it is 0.0000001 with limiting. Table 3-1 shows a more complete set of limiting situations.

3.2.5.2 Scaling

The data shifters can shift data one bit to the left or one bit to the right, or pass the data unshifted. Each data shifter has a 24-bit output with overflow indication and is controlled by the scaling mode bits in the status register. 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.

3.3 DATA REPRESENTATION AND ROUNDING

The DSP56K uses a fractional data representation for all Data ALU operations. Figure 3-

Table 3-1 Limited Data Values

Destination Memory Reference	Source Operand	Accumulator Sign	Limited Value (Hexadecimal)		Type of Access
			XDB	YDB	
X	X:A X:B	+ -	7FFFFF 800000	— —	One 24 bit
Y	Y:A Y:B	+ -	— —	7FFFFF 800000	One 24 bit
X and Y	X:A Y:A X:A Y:B X:B Y:A X:B Y:B L:AB L:BA	+ - + - + -	7FFFFF 800000 7FFFFF 800000 7FFFFF 800000	7FFFFF 800000 7FFFFF 800000 7FFFFF 800000	Two 24 bit
L (X:Y)	L:A L:B	+ -	7FFFFF 800000	FFFFFF 000000	One 48 bit

7 shows the bit weighting of words, long words, and accumulator operands for this representation. The decimal points are all aligned and are left justified.

Data must be converted to a fractional number by scaling before being used by the DSP or the user will have to be very careful in how the DSP manipulates the data. Moving $3F to a 24-bit Data ALU register does not result in the contents being $00003F as might be expected. Assuming numbers are fractional, the DSP left justifies rather than right justifies. As a result, storing $3F in a 24-bit register results in the contents being $3F0000. The simplest example of scaling is to convert all integer numbers to fractional numbers by shifting the decimal 24 places to the left (see Figure 3-6). Thus, the data has not changed; only the position of the decimal has moved.

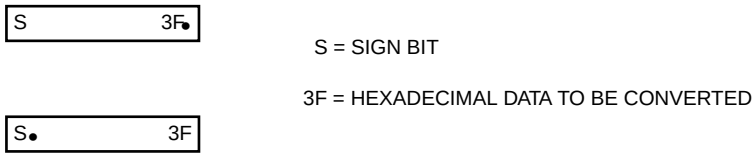


Figure 3-6 Integer-to-Fractional Data Conversion

For words and long words, the most negative number that can be represented is -1 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 $7FFFFFFFFFF

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 approximately -256. When the accumulator extension registers are in use, the data contained in the accumulators cannot be stored exactly in memory or other registers. In these cases, the data must be limited to the most positive or most negative number consistent with the size of the destination and the sign of the accumulator (the most significant bit (MSB) of the extension register).

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. The least significant accumulator register 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 A0 or B0 and the most significant word is written to

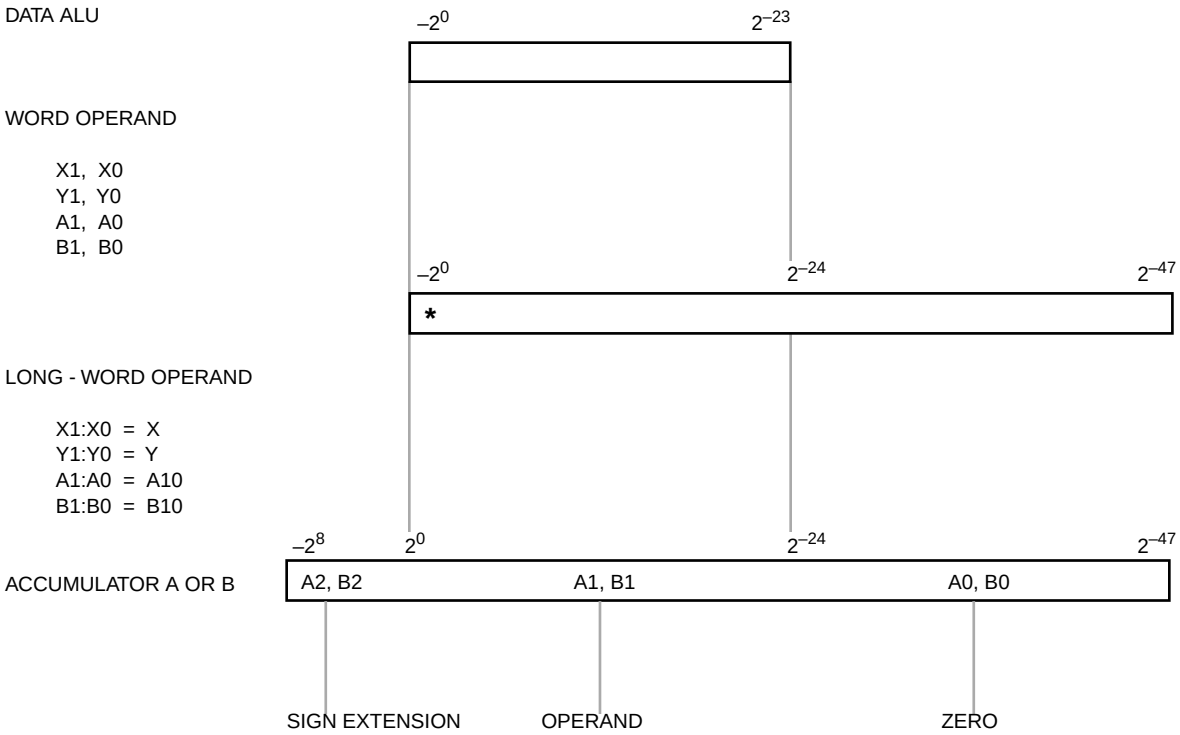


Figure 3-7 Bit Weighting and Alignment of Operands

A1 or B1(see Figure 3-8).

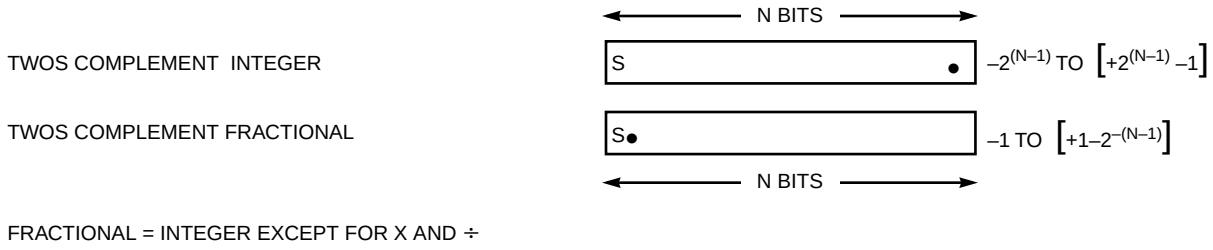


Figure 3-8 Integer/Fractional Number Comparison

A comparison between integer and fractional number representation is shown in Figure 3-8. 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 3-9. The key difference is that the extra bit in the integer multiplication is used as a duplicate sign bit and as the least significant bit (LSB) in the fractional multiplication. The advantages of fractional data representation are as follows:

- The MSP (left half) has the same format as the input data.
- The LSP (right half) can be rounded into the MSP without shifting or updating the exponent.
- A significant bit is not lost through sign extension.
- Conversion to floating-point representation is easier because the industry-standard floating-point formats use fractional mantissas.
- Coefficients for most digital filters are derived as fractions by the high-level language programs used in digital-filter design packages, which implies that the results can be used without the extensive data conversions that other formats require.

Should integer arithmetic be required in an application, shifting a one or zero, depending on the sign, into the MSB converts a fraction to an integer.

The Data ALU MAC performs rounding of the accumulator register to single precision if requested in the instruction (the A1 or B1 register is rounded according to the contents of the A0 or B0 register). The rounding method is called round-to-nearest (even) number, or convergent rounding. The usual rounding method rounds up any value above one-half

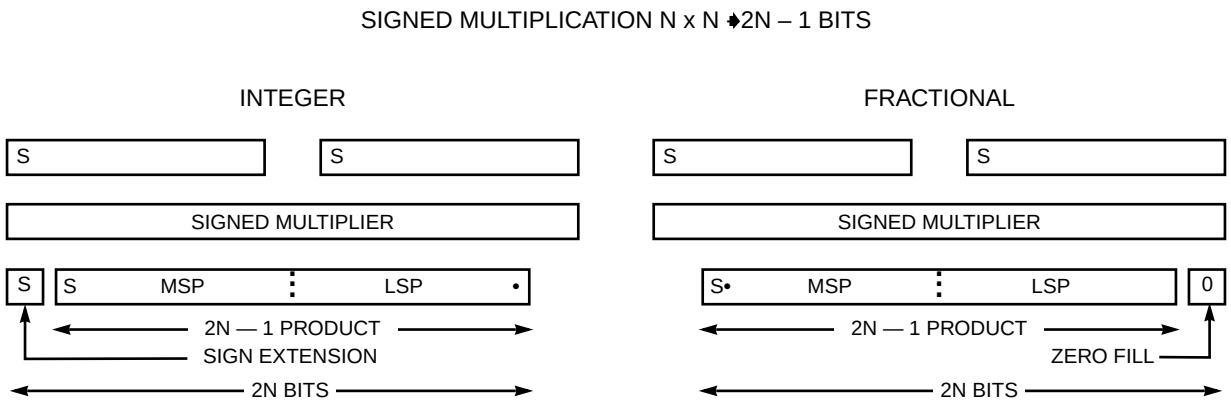
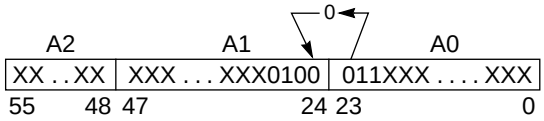


Figure 3-9 Integer/Fractional Multiplication Comparison

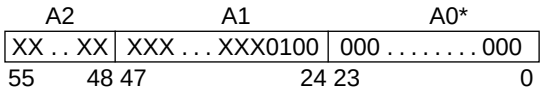
and rounds down any value below one-half. The question arises as to which way one-half should be rounded. If it is always rounded one way, the results will eventually be biased in that direction. Convergent rounding solves the problem by rounding down if the number is odd (LSB=0) and rounding up if the number is even (LSB=1). Figure 3-10 shows the four cases for rounding a number in the A1 (or B1) register. If scaling is set in the status register, the resulting number will be rounded as 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)

BEFORE ROUNDING

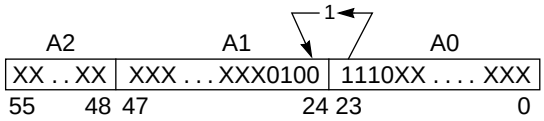


AFTER ROUNDING

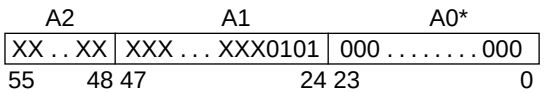


CASE II: IF A0 > $800000 (1/2), THEN ROUND UP (ADD 1 TO A1)

BEFORE ROUNDING

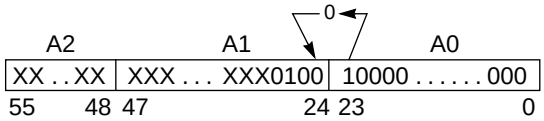


AFTER ROUNDING

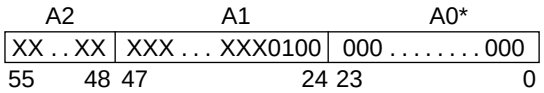


CASE III: IF A0 = $800000 (1/2), AND THE LSB OF A1 = 0,THEN ROUND DOWN (ADD NOTHING)

BEFORE ROUNDING

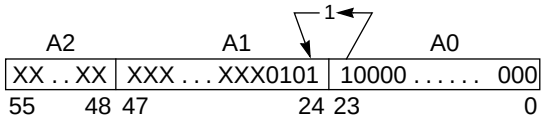


AFTER ROUNDING

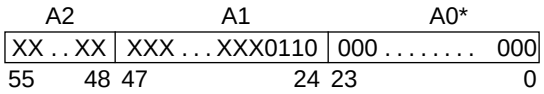


CASE IV: IF A0 = $800000 (1/2), AND THE LSB = 1, THEN ROUND UP (ADD 1 TO A1)

BEFORE ROUNDING



AFTER ROUNDING



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

Figure 3-10 Convergent Rounding

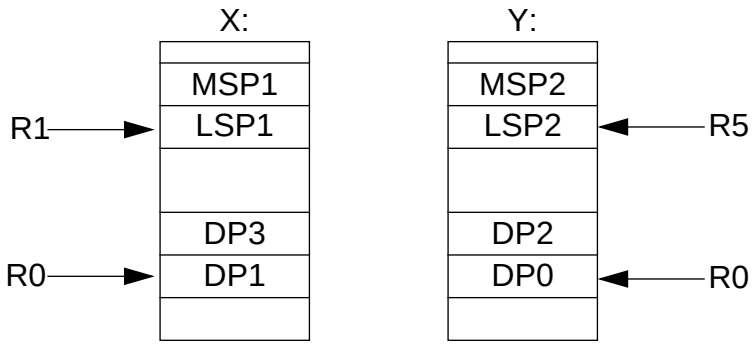
3.4 DOUBLE PRECISION MULTIPLY MODE

The Data ALU double precision multiply operation multiplies two 48-bit operands with a 96-bit result. The processor enters the dedicated Double Precision Multiply Mode when the user sets bit 14 (DM) of the Status Register (bit 6 of the MR register). The mode is disabled by clearing the DM bit. For information on the DM bit, see Section 5.4.2.13 - Double Precision Multiply Mode (Bit 14).

CAUTION:

While in the Double Precision Multiply Mode, only the double precision multiply algorithms shown in Figure 3-11, Figure 3-12, and Figure 3-13 may be executed by the Data ALU; any other Data ALU operation will give indeterminate results.

Figure 3-11 shows the full double precision multiply algorithm. To allow for pipeline delay, the ANDI instruction should not be immediately followed by a Data ALU instruction. For example, the ORI instruction sets the DM mode bit, but, due to the instruction execution pipeline, the Data ALU enters the Double Precision Multiply mode only after



DP3_DP2_DP1_DP0 = MSP1_LSP1 x MSP2_LSP2

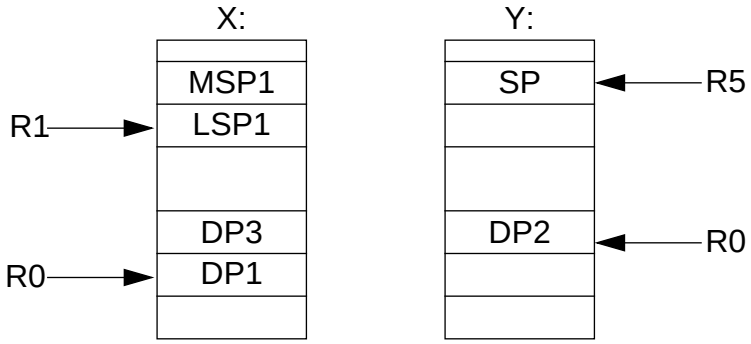
```
ori     #$40,mr                                     ;enter mode
move            x:(r1)+,x0      y:(r5)+,y0          ;load operands
mpy     y0,x0,a x:(r1)+,x1      y:(r5)+,y1          ;LSP*LSP→a
mac     x1,y0,a                 a0,y:(r0)           ;shifted(a)+
                                     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

Figure 3-11 Full Double Precision Multiply Algorithm

one instruction cycle. The ANDI instruction clears the DM mode bit, but, due to the instruction execution pipeline, the Data ALU leaves the mode after one instruction cycle.

The double precision multiply algorithm uses the Y0 register at all stages. 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.

If just single precision times double precision multiply is desired, two of the multiply operations may be deleted and replaced by suitable initialization and clearing of the accumulator and Y0. Figure 3-12 shows the single precision times double precision algorithm.



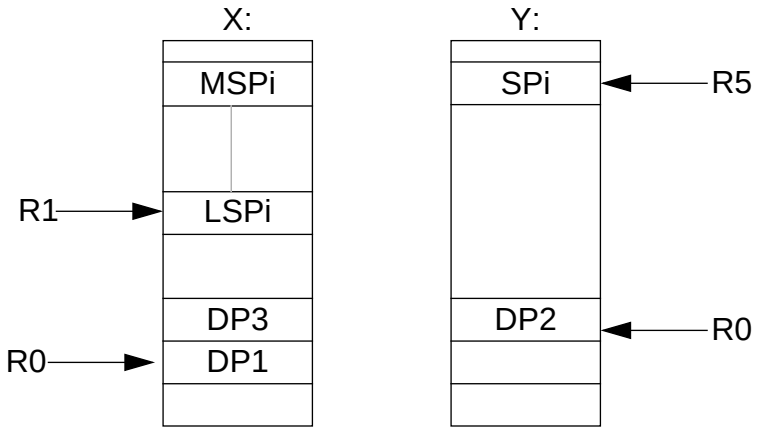
DP3_DP2_DP1 = MSP1_LSP1 x SP

```
clr a                #0,y0                          ;clear a and y0
ori     #$40,mr                                     ;enter DP mode
move                 x:(r1)+,x0     y:(r5)+,y1      ;load LSP1 and SP
mac     x0,y1,a      x:(r1)+,x1                     ;LSP1*SP→a,
                                                    ;load MSP1
mac     y1,x1,a      a0,x:(r0)+                     ;shifted(a)+
                                                    ;  SP*MSP1→a,
                                                    ;save DP1
move    a,l:(r0)+                                   ;save DP3_DP2
andi    #$bf,mr                                     ;exit DP mode
non-Data ALU operation                              ;pipeline delay
```

Figure 3-12 Single × Double Multiply Algorithm

Figure 3-13 shows a single precision times double precision multiply-accumulate algorithm. First, the least significant parts of the double precision values are multiplied by the single precision values and accumulated in the “Double Precision Multiply” mode. Then the DM bit is cleared and the least significant part of the result is saved to memory. The most significant parts of the double precision values are then multiplied by the single pre-

cision values and accumulated using regular MAC instructions. Note that the maximum number of single times double MAC operations in this algorithm are limited to 255 since overflow may occur (the A2 register is just eight bits long). If a longer sequence is required, it should be split into sub-sequences each with no more than 255 MAC operations.



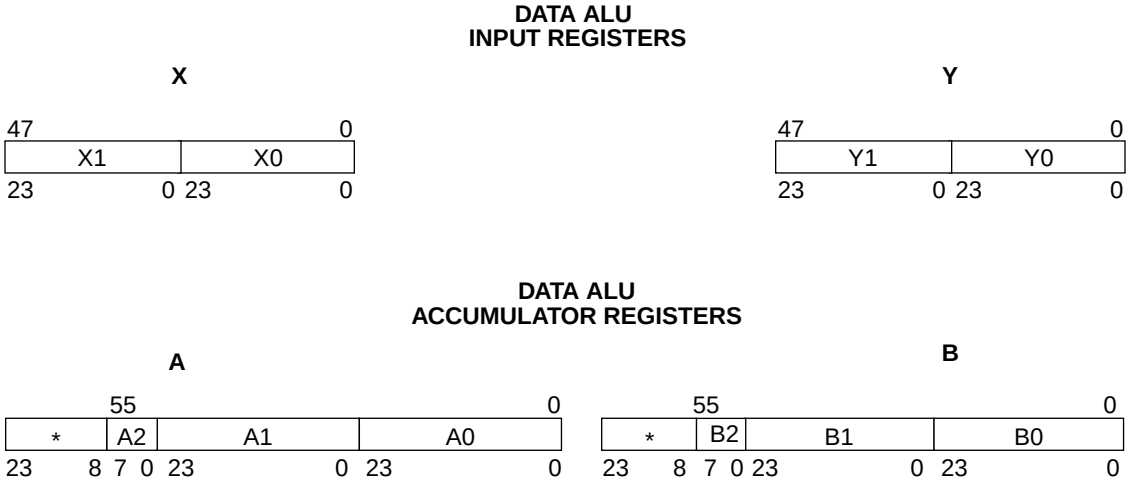
$$DP3\_DP2\_DP1 = \sum MSPi\_LSPi \times SPi$$

```
move              #N-1,m5
clr a             #0,y0                              ;clear a and y0
ori     #$40,mr                                      ;enter DP mode
move              x:(r1)+,x0      y:(r5)+,y1         ;load LSPi and SPi
rep     #N                                           ;0<N<256
mac     x0,y1,a   x:(r1)+,x0      y:(r5)+,y1         ;LSPi*SPi→a
andi    #$bf,mr                                      ;exit DP mode
move              a0,x:(r0)+                         ;save DP1
move              a1,y0
move              a2,a
move              y0,a0                              ;a2:a1→a1:a0
rep     #N
mac     x0,y1,a   x:(r1)+,x0      y:(r5)+,y1         ;load MSPi and SPi
move    a,l:(r0)+                                    ;save DP3_DP2
```

Figure 3-13 Single × Double Multiply-Accumulate Algorithm

3.5 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 3-14 illustrates how the registers in the programming model are grouped.



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

Figure 3-14 DSP56K Programming Model

3.6 DATA ALU SUMMARY

The Data ALU performs arithmetic operations involving multiply and accumulate operations. It executes all instructions in one machine cycle and is not pipelined. The two 24-bit numbers being multiplied can come from the X registers (X0 or X1) or Y registers (Y0 or Y1). After multiplication, they are added (or subtracted) with one of the 56-bit accumulators and can be convergently rounded to 24 bits. The convergent-rounding forcing function detects the $800000 condition in the LSP and makes the correction as necessary. The final result is then stored in one of the accumulators as a valid 56-bit number. The condition code bits are set based on the rounded output of the logic unit.