AN1219

M68HC08 Integer Math Routines

By Mark Johnson
CSIC Applications Engineering
Austin, Texas

Introduction

This application note discusses six integer math subroutines[1] that take advantage of one of the main CPU enhancements in the 68HC08 Family of microcontroller units (MCU). Each of these subroutines uses stack relative addressing, an important CPU enhancement.

Although the 68HC08 MCU is a fully upward-compatible performance extension of the 68HC05 MCU Family, users familiar with the 68HC05 should have little difficulty implementing the 68HC08 architectural enhancements. For instance, storage space for local variables needed by a subroutine can now be allocated on the stack when a routine is entered and released on exit. Since this greatly reduces the need to assign variables to global RAM space, these integer math routines are implemented using only 10 bytes of global RAM space. Eight bytes of global RAM are reserved for the two 32-bit pseudo-accumulators, INTACC1 and INTACC2. The other 2 bytes assigned to SPVAL are used by the unsigned 32 x 32 multiply routine to store the value of the stack pointer.

INTACC1 and INTACC2 are defined as two continuous 4-byte global RAM locations that are used to input hexadecimal numbers to the

1. None of the six subroutines contained in this appication note check for valid or non-zero numbers in the two integer accumulators. The user is responsible for ensuring that proper values are placed in INTACC1 and INTACC2 before the subroutines are invoked.

subroutines[1] and to return the results. For proper operation of the following subroutines, these two storage locations must be allocated together, but may be located anywhere in RAM address space. SPVAL may be allocated anywhere in RAM address space.

Software Description

Unsigned 16 × 16 Multiply (UMULT16)

Entry conditions:

INTACC1 and INTACC2 contain the unsigned 16-bit numbers to be multiplied.

Exit conditions:

INTACC1 contains the unsigned 32-bit product of the two integer accumulators.

Size:

94 bytes

Stack space:

9 bytes

Subroutine calls:

None

Procedure:

This routine multiplies the two leftmost bytes of INTACC1 (INTACC1 = MSB, INTACC1 + 1 = LSB) by the two leftmost bytes of INTACC2 (INTACC2 = MSB, INTACC2 + 1 = LSB). (MSB is the acronym for most significant byte and LSB stands for least significant byte.) Temporary stack storage locations 1,SP–5,SP are used to hold the two intermediate products. These intermediate products are then added together and the final 32-bit result is stored in INTACC1 (INTACC1 = MSB, INTACC1 + 3 = LSB). This process is illustrated in **Figure 1** and in **Figure 2**.

1. The 32 x 16 unsigned divide algorithm was based on the algorithm written for the M6805 by Don Weiss and was modified to return a 32-bit quotient. The table lookup and interpolation routine was written by Kevin Kilbane and was modified to interpolate both positive and negative slope linear functions.

INTACC1 = Multiplier
INTACC2 = Multiplicand

```
                          INTACC1 × INTACC2

                          INTACC1 : INTACC1 + 1
=                      ×  INTACC2 : INTACC2 + 1
                      ---------------------------
           (INTACC1 : INTACC1 + 1) (INTACC2 + 1)
=     (INTACC1 : INTACC1 + 1) (INTACC2)
      ---------------------------------------------------
                          1,SP      2,SP      INTACC1 + 3
+        3,SP             4,SP      5,SP
      ---------------------------------------------------
=     INTACC1 : INTACC + 1 : INTACC1 + 2 : INTACC1 + 3
```

Figure 1. Unsigned Multiply 16 x 16 Equation

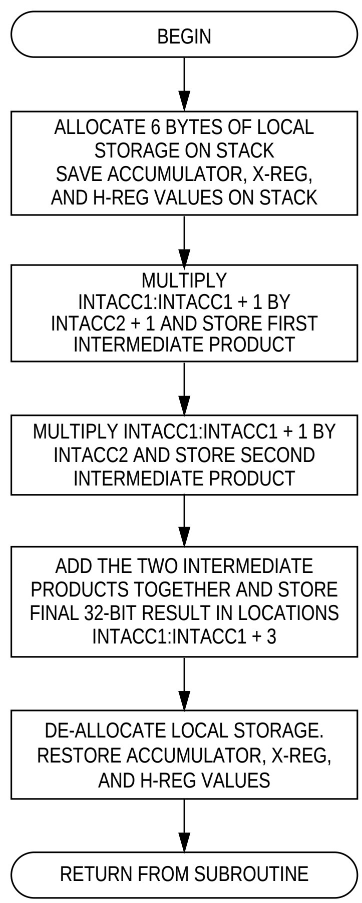

Figure 2. Unsigned 16 × 16 Multiply

Unsigned 32 × 32 Multiply (UMULT32)

Entry conditions:

INTACC1 and INTACC2 contain the unsigned 32-bit numbers to be multiplied.

Exit conditions:

INTACC1 concatenated with INTACC2 contains the unsigned 64-bit result.

Size:

158 bytes

Stack space:

38 bytes

Subroutine calls:

None

Procedure:

This subroutine multiplies the unsigned 32-bit number located in INTACC1 (INTACC1 = MSB, INTACC1 + 3 = LSB) by the unsigned 32-bit number stored in INTACC2 (INTACC2 = MSB, INTACC2 + 3 = LSB). Each byte of INTACC2, starting with the LSB, is multiplied by the 4 bytes of INTACC1 and a 5-byte intermediate product is generated. The four intermediate products are stored in a 32-byte table located on the stack. These products are then added together and the final 8-byte result is placed in INTACC1:INTACC2 + 3 (INTACC1 = MSB, INTACC2 + 3 = LSB). An illustration of this mathematical process is shown in **Figure 3** and **Figure 4**.

INTACC1 = Multiplier
INTACC2 = Multiplicand

	INTACC1 × INTACC2							
	INTACC1:INTACC1 + 1:INTACC1 + 2:INTACC1 + 3							
=	× INTACC2:INTACC2 + 1:INTACC2 + 2:INTACC2 + 3							
	(INTACC1:INTACC1 + 1:INTACC1 + 2:INTACC1 + 3)(INTACC2 + 3)							
	(INTACC1:INTACC1 + 1:INTACC1 + 2:INTACC1 + 3)(INTACC2 + 2)							
	(INTACC1:INTACC1 + 1:INTACC1 + 2:INTACC1 + 3)(INTACC2 + 1)							
=	(INTACC1:INTACC1 + 1:INTACC1 + 2:INTACC1 + 3)(INTACC2)							
	0	0	0	IR03	IR04	IR05	IR06	R07 [1]
	0	0	IR12	IR13	IR14	IR15	IR16	0
	0	IR21	IR22	IR23	IR24	IR25	0	0
+	IR30	IR31	IR32	IR33	IR34	0	0	0
=	INTACC1...INTACC2 + 3							

1. The intermediate result (IR) tags are temporary storage locations on the stack, not hard-coded locations in RAM.

Figure 3. Unsigned 32 x 32 Multiply Equation

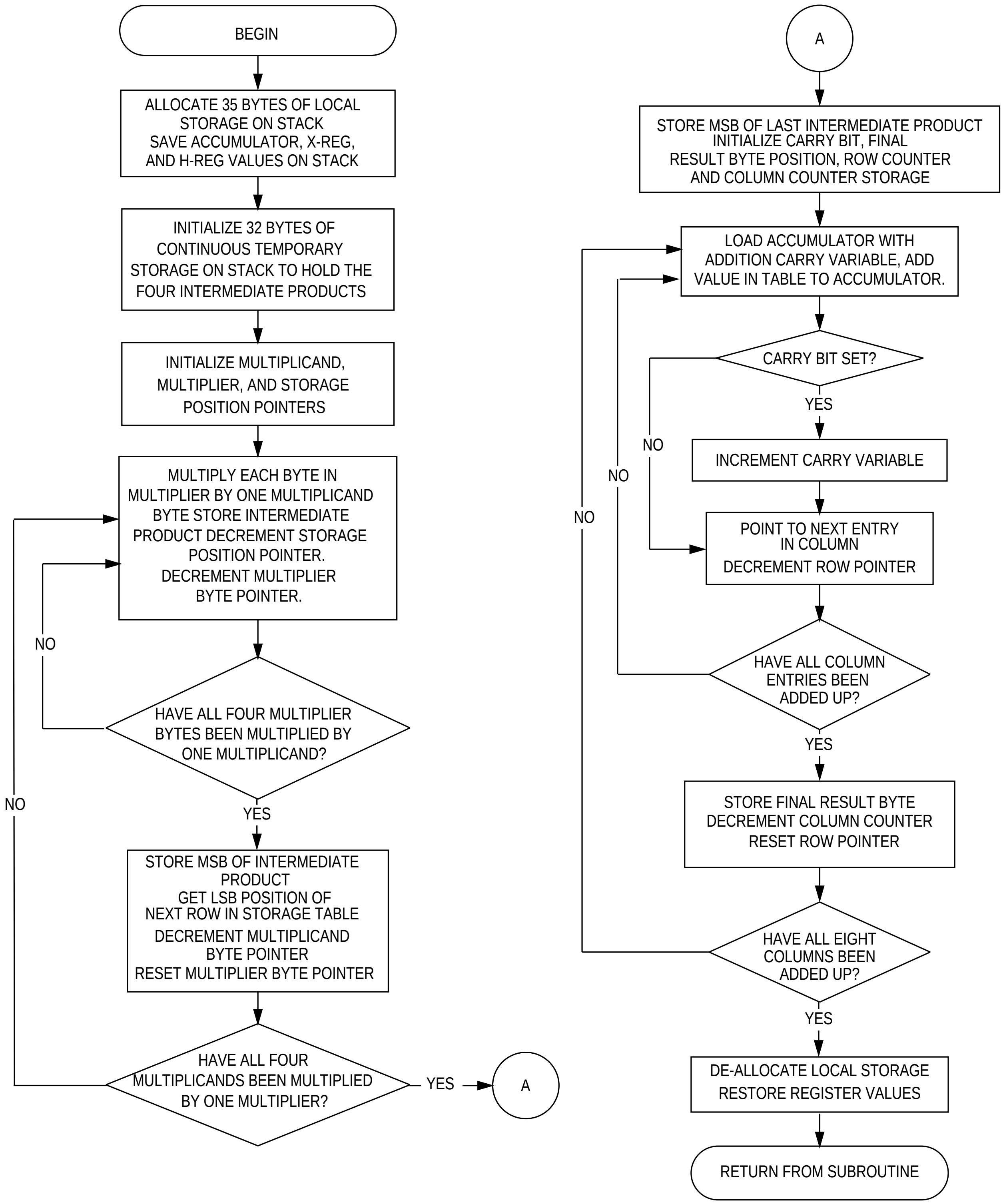

Figure 4. Unsigned 32 $\times$ 32 Multiply

Signed 8 × 8 Multiply (SMULT8)

Entry conditions:

INTACC1 and INTACC2 contain the signed 8-bit numbers to be multiplied.

Exit conditions:

The two leftmost bytes of INTACC1 (INTACC1 = MSB, INTACC1 + 1 = LSB) contain the signed 16-bit product.

Size:

57 bytes

Stack space:

4 bytes

Subroutine calls:

None

Procedure:

This routine performs a signed multiply of INTACC1 (MSB) and INTACC2 (MSB). Before multiplying the two numbers together, the program checks the MSB of each byte and performs a two's complement of that number if the MSB is set. One byte of temporary stack storage is used to hold the result sign. If both of the numbers to be multiplied are either negative or positive, the result sign LSB is cleared or it is set to indicate a negative result. Both numbers are then multiplied together and the results are placed in the two left-most bytes of INTACC1 (INTACC1 = MSB, INTACC1 + 1 = LSB). The routine is exited if the result sign storage location is not equal to one or the result is two's complemented and the negative result is stored in locations INTACC1 and INTACC1 + 1.

INTACC1 = Multiplier
INTACC2 = Multiplicand

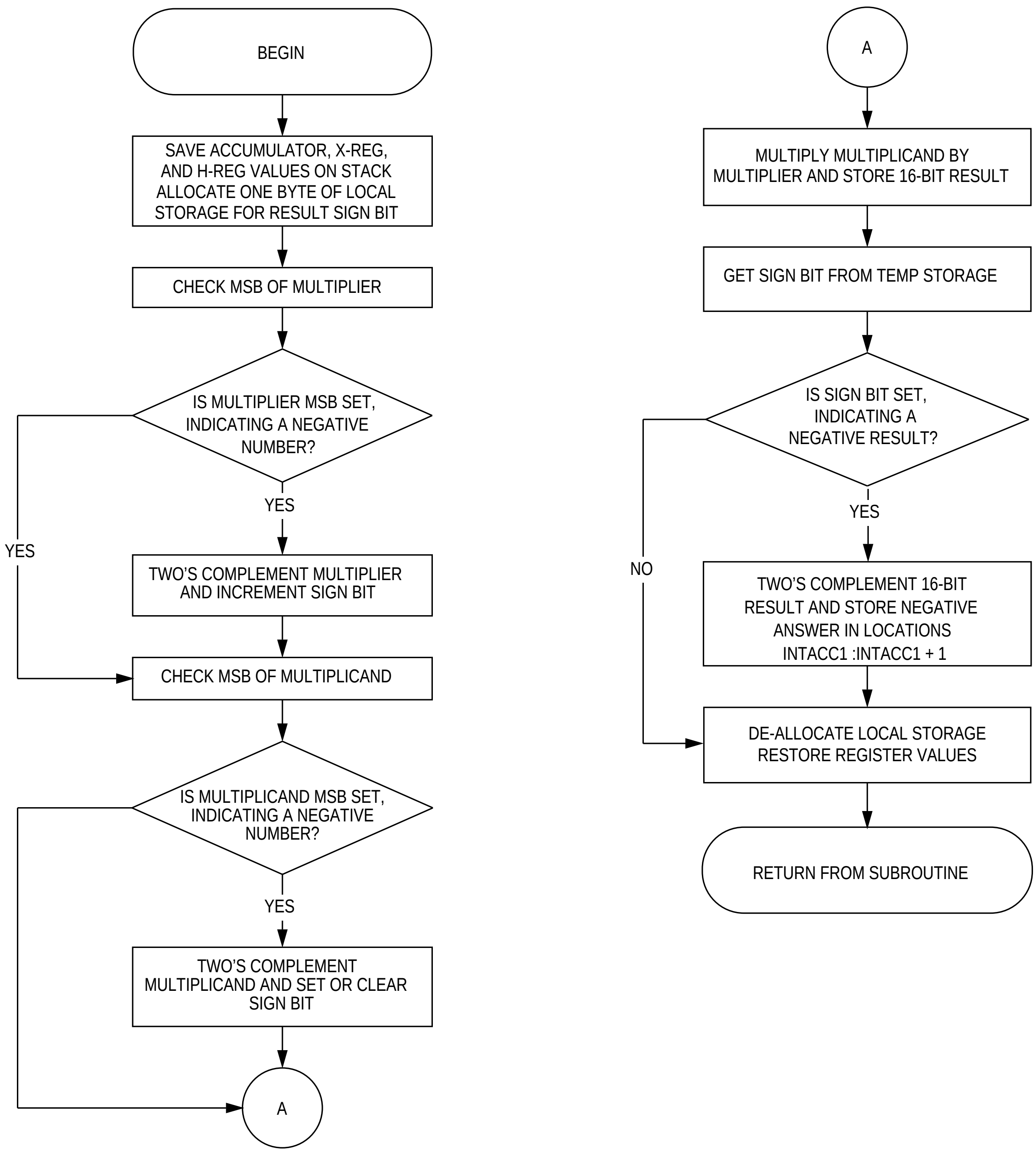

Figure 5. Signed 8 × 8 Multiply

Signed 16 × 16 Multiply (SMULT16)

Entry conditions:

INTACC1 and INTACC2 contain the signed 16-bit numbers to be multiplied.

Exit conditions:

INTACC1 contains the signed 32-bit result.

Size:

83 bytes

Stack space:

4 bytes

Subroutine calls:

UMULT16

Procedure:

This routine multiplies the signed 16-bit number in INTACC1 and INTACC1 + 1 by the signed 16-bit number in INTACC2 and INTACC2 + 1. Before multiplying the two 16-bit numbers together, the sign bit (MSB) of each 16-bit number is checked and a two's complement of that number is performed if the MSB is set. One byte of temporary stack storage space is allocated for the result sign. If both 16-bit numbers to be multiplied are either positive or negative, the sign bit LSB is cleared, indicating a positive result, but otherwise the sign bit LSB is set. Subroutine UMULT16 is called to multiply the two 16-bit numbers together and store the 32-bit result in locations INTACC:INTACC1 + 3 (INTACC1 = MSB, INTACC2 = LSB). The routine is exited if the result sign LSB is cleared or the result is two's complemented by first one's complementing each byte of the product and then adding one to that result to complete the two's complement. The 32-bit negative result is then placed in INTACC1.

INTACC1 = Multiplier
INTACC2 = Multiplicand

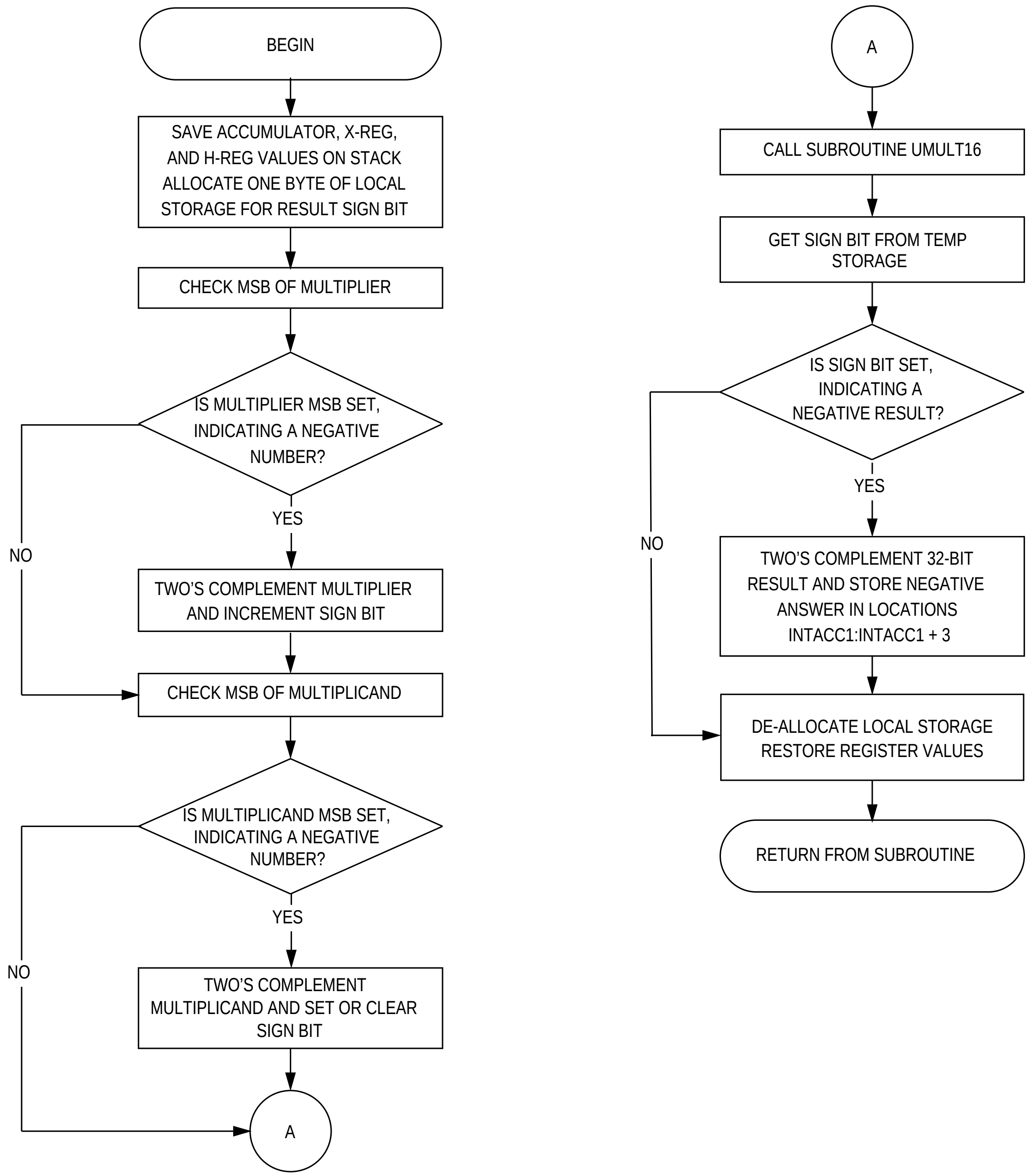

Figure 6. Signed 16 $\times$ 16 Multiply

32 × 16 Unsigned Divide (UDVD32)

Entry conditions:

INTACC1 contains the 32-bit unsigned dividend and INTACC2 contains the 16-bit unsigned divisor.

Exit conditions:

INTACC1 contains the 32-bit quotient and INTACC2 contains the 16-bit remainder.

Size:

136 bytes

Stack space:

6 bytes

Subroutine calls:

None

Procedure:

This routine takes a 32-bit dividend stored in INTACC1:INTACC1 + 3 and divides it by the divisor stored in INTACC2:INTACC2 + 1 using the standard shift-and-subtract algorithm. This algorithm first clears the 16-bit remainder, then shifts the dividend/quotient to the left one bit at a time until all 32 bits of the dividend have been shifted through the remainder and the divisor is subtracted from the remainder. (See illustration.) Each time a trial subtraction succeeds, a 1 is placed in the LSB of the quotient. The 32-bit quotient is placed in locations INTACC1 = MSB:INTACC1 + 3 = LSB and the remainder is returned in locations INTACC2 = MSB, INTACC2 + 1 = LSB.

Before subroutine is executed:

INTACC1	INTACC1 + 1	INTACC1 + 2	INTACC1 + 3	INTACC2	INTACC2 + 1
Dividend MSB	Dividend	Dividend	Dividend LSB	Divisor MSB	Divisor LSB

During subroutine execution:

INTACC1	INTACC1 + 1	INTACC1 + 2	INTACC1 + 3	INTACC2	INTACC2 + 1
Remainder MSB	Remainder LSB	Dividend MSB	Dividend	Dividend	Dividend LSB/ Quotient MSB
– Divisor MSB	– Divisor LSB				

After return from subroutine:

INTACC1	INTACC1 + 1	INTACC1 + 2	INTACC1 + 3	INTACC2	INTACC2 + 1
Quotient MSB	Quotient	Quotient	Quotient LSB	Remainder MSB	Remainder LSB

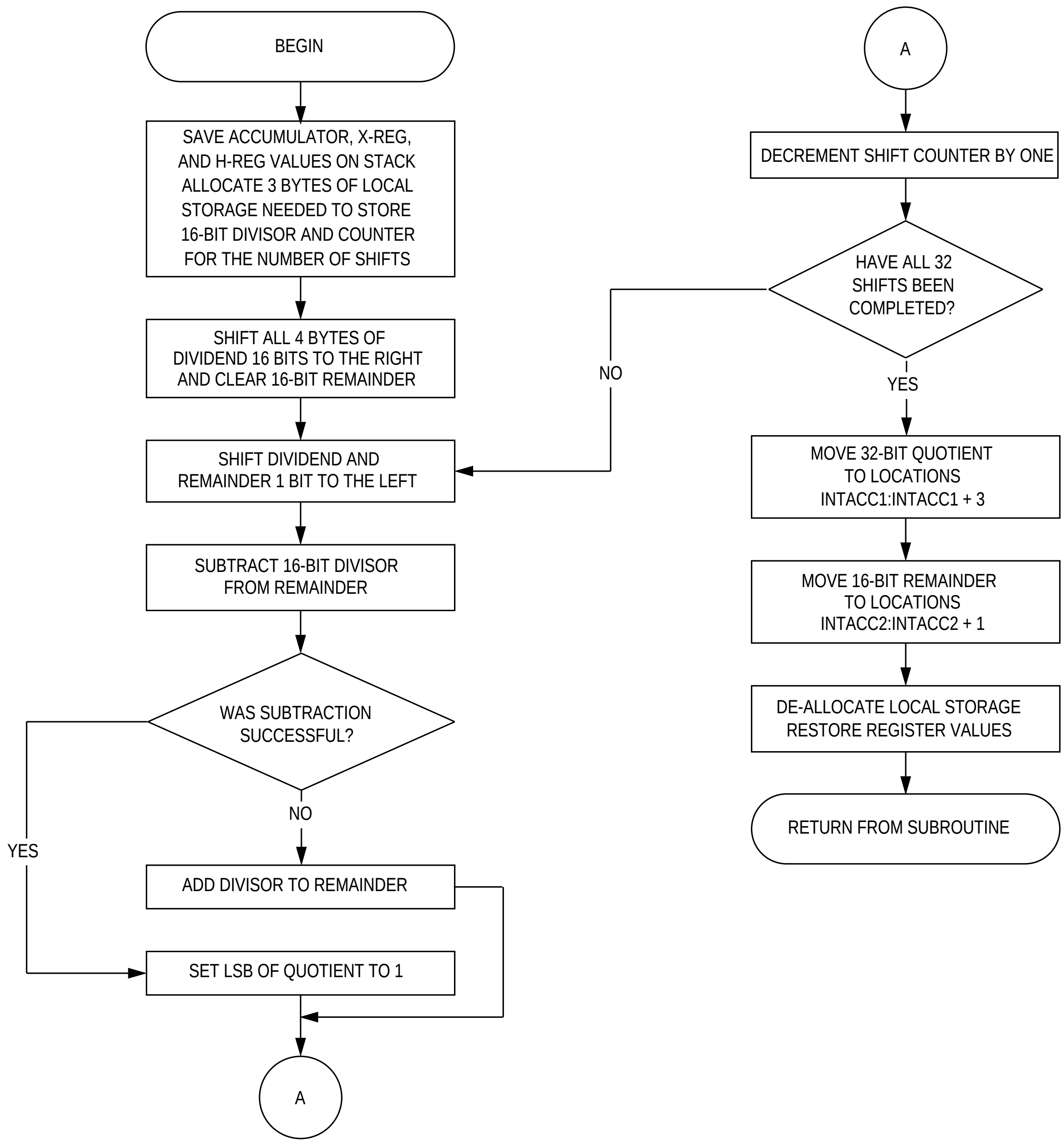

Figure 7. 32 × 16 Unsigned Divide

Table Lookup and Interpolation (TBLINT)

Entry conditions:

INTACC1 contains the position of table ENTRY 2. INTACC1 + 1 contains the interpolation fraction.

Exit conditions:

INTACC1 + 2 : INTACC1 + 3 contains the 16-bit interpolated value (INTACC1 + 2 = MSB, INTACC1 + 3 = LSB).

Size:

125 bytes

Stack space:

4 bytes

Subroutine calls:

None

Procedure:

This routine performs table lookup and linear interpolation between two 16-bit dependent variables (Y) from a table of up to 256 entries and allowing up to 256 interpolation levels between entries. (By allowing up to 256 levels of interpolation between two entries, a 64-k table of 16-bit entries can be compressed into just 256 16-bit entries.) INTACC1 contains the position of table entry 2 and INTACC1 + 1 contains the interpolation fraction. The unrounded 16-bit result is placed in INTACC1 + 2 = MSB, INTACC1 + 3 = LSB. INTACC2 is used to hold the two 16-bit table entries during subroutine execution.

The interpolated result is of the form:

$$Y = ENTRY1 + (INTPFRC(ENTRY2 - ENTRY1)) / 256$$

where:

- Y can be within the range $0 < Y < 32,767$.
- $INTPFRC = (1 \leq X \leq 255) / 256$
- ENTRY1 and ENTRY2 can be within the range $0 < ENTRY < 32767$.
- Slope of linear function can be either positive or negative.
- The table of values can be located anywhere in the memory map.

Example**:**

Table 1. Lookup and Interpolation

	Entry Number	Y Value
	0	0
	:	:
	145	1688
ENTRY 1 →	146	2416
ENTRY 2 →	147	4271
	:	:
	255	0

- Find the interpolated Y value half way between entry 146 and 147.
- ENTRY2 = Entry # 147 = 4271
- ENTRY1 = Entry # 146 = 2416
- For a 50% level of interpolation: INTPFRC = 128 / 256 = $80
- So:

  Y = 2416 + (128(4271 – 2416))/256
  = 2416 + (128(1855))/256
  = 2416 + 927
  Y = 3343_{10} or $D0F

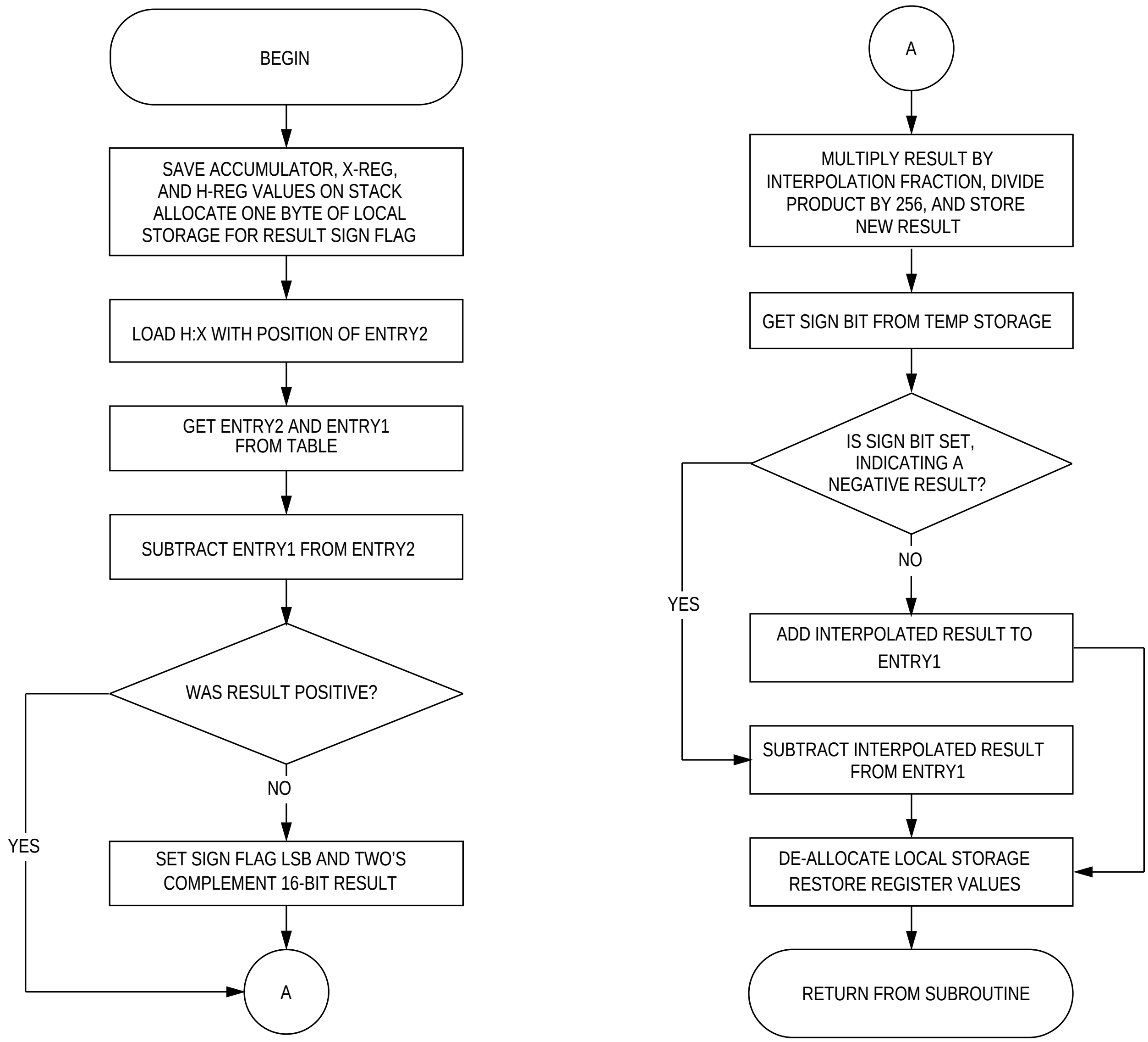

Figure 8. Table Lookup and Interpolation

Software Listing

```
********************************************************************************
*
*       File name: IMTH08.ASM
*       Revision: 1.00
*       Date: February 24, 1993
*
*       Written By: Mark Johnson
*                     CSIC Applications
*
*       Assembled Under: P&E Microcomputer Systems IASM08 (Beta Version)
*
*                          ********************************
*                          *       Revision History       *
*                          ********************************
*
*       Revision 1.00   2/24/93 Original Source
********************************************************************************
*
*       Program Description:
*
*       This program contains six* integer math routines for the 68HC08 Family
*       of microcontrollers.
*
*     *Note: 1)  The 32 x 16 Unsigned divide algorithm was based on
*                the one written for the 6805 by Don Weiss and was
*                modified to return a 32-bit quotient.
*            2)  The Table lookup and interpolation algorithm was
*                based on the one written by Kevin Kilbane and was
*                modified to interpolate both positive and negative
*                slope linear functions.
*
********************************************************************************
*
*     Start of main routine
*
            ORG     $50           ;RAM address space
*
INTACC1     RMB     4             ;32-bit integer accumulator #1
INTACC2     RMB     4             ;32-bit integer accumulator #2
SPVAL       RMB     2             ;storage for stack pointer value
*
            ORG     $6E00         ;ROM/EPROM address space
START       LDHX    #$450         ;load H:X with upper RAM boundary + 1
            TXS                   ;move stack pointer to upper RAM boundary
            CLRH                  ;clear H:X
            JSR     UMULT16       ;call unsigned 16 x 16 multiply routine
            JSR     UMULT32       ;call unsigned 32 x 32 multiply routine
            JSR     UMULT8        ;call signed 8 x 8 multiply routine
            JSR     UMULT16       ;call signed 16 x 16 multiply routine
            JSR     UMULT32       ;call 32 x 16 multiply routine
            JSR     TBLINT        ;call table interpolation routine
            BRA     *             ;end of main routine
```

```
*******************************************************************************
*       Start of subroutine
*       Unsigned 16x16 multiply
*
*       This routine multiplies the 16-bit unsigned number stored in
*       locations INTACC1:INTACC1+1 by the 16-bit unsigned number stored in
*       locations INTACC2:INTACC2+1 and places the 32-bit result in locations
*       INTACC1:INTACC1+3 (INTACC1 = MSB:INTACC1+3 = LSB.
*
*******************************************************************************
UMULT16     EQU     *
            PSHA                        ;save acc
            PSHX                        ;save x-reg
            PSHH                        ;save h-reg
            AIS     #-6                 ;reserve six bytes of temporary
                                        ;storage on stack
            CLR     6,SP                ;zero storage for multiplication carry
*
*       Multiply (INTACC1:INTACC1+1) by INTACC2+1
*
            LDX     INTACC1+1           ;load x-reg w/multiplier LSB
            LDA     INTACC2+1           ;load acc w/multiplicand LSB
            MUL                         ;multiply
            STX     6,SP                ;save carry from multiply
            STA     INTACC1+3           ;store LSB of final result
            LDX     INTACC1             ;load x-reg w/multiplier MSB
            LDA     INTACC2+1           ;load acc w/multiplicand LSB
            MUL                         ;multiply
            ADD     6,SP                ;add carry from previous multiply
            STA     2,SP                ;store 2nd byte of interm. result 1.
            BCC     NOINCA              ;check for carry from addition
            INCX                        ;increment MSB of interm. result 1.
NOINCA      STX     1,SP                ;store MSB of interm. result 1.
            CLR     6,SP                ;clear storage for carry
*
*       Multiply (INTACC1:INTACC1+1) by INTACC2
*
            LDX     INTACC1+1           ;load x-reg w/multiplier LSB
            LDA     INTACC2             ;load acc w/multiplicand MSB
            MUL                         ;multiply
            STX     6,SP                ;save carry from multiply
            STA     5,SP                ;store LSB of interm. result 2.
            LDX     INTACC1             ;load x-reg w/multiplier MSB
            LDA     INTACC2             ;load acc w/multiplicand MSB
            MUL                         ;multiply
            ADD     6,SP                ;add carry from previous multiply
            STA     4,SP                ;store 2nd byte of interm. result 2.
            BCC     NOINCB              ;check for carry from addition
            INCX                        ;increment MSB of interm. result 2.
NOINCB      STX     3,SP                ;store MSB of interm. result 2.
*
```

```
*     Add the intermediate results and store the remaining three bytes of the
*     final value in locations INTACC1:INTACC1+2.
*
            LDA     2,SP                ;load acc with 2nd byte of 1st result
            ADD     5,SP                ;add acc with LSB of 2nd result
            STA     INTACC1+2           ;store 2nd byte of final result
            LDA     1,SP                ;load acc with MSB of 1st result
            ADC     4,SP                ;add w/ carry 2nd byte of 2nd result
            STA     INTACC1+1           ;store 3rd byte of final result
            LDA     3,SP                ;load acc with MSB from 2nd result
            ADC     #0                  ;add any carry from previous addition
            STA     INTACC1             ;store MSB of final result
*
*     Reset stack pointer and recover original register values
*
            AIS     #6                  ;deallocate the six bytes of local
                                        ;storage
            PULH                        ;restore h-reg
            PULX                        ;restore x-reg
            PULA                        ;restore accumulator
            RTS                         ;return
********************************************************************************
********************************************************************************
*
*     Unsigned 32 x 32 Multiply
*
*     This routine multiplies the unsigned 32-bit number stored in locations
*     INTACC1:INTACC1+3 by the unsigned 32-bit number stored in locations
*     INTACC2:INTACC2+3 and places the unsigned 64-bit result in locations
*     INTACC1:INTACC2+3 (INTACCC1 = MSB:INTACC2+3 = LSB).
*
********************************************************************************
UMULT32     EQU     *
            PSHA                        ;save acc
            PSHX                        ;save x-reg
            PSHH                        ;save h-reg
            CLRX                        ;zero x-reg
            CLRA                        ;zero accumulator
            AIS     #-35T               ;reserve 35 bytes of temporary storage
                                        ;on stack
            TSX                         ;transfer stack pointer + 1 to H:X
            AIX     #32T                ;add number of bytes in storage table
            STHX    SPVAL               ;save end of storage table value
            AIX     #-32T               ;reset H:X to stack pointer value
*
*     Clear 32 bytes of storage needed to hold the intermediate results
*
INIT        CLR     ,X                  ;xero a byte of storage
            INCX                        ;point to next location
            CPHX    SPVAL               ;check for end of table
            BNE     INIT                ;
*
```

```
*       Initialize multiplicand and multiplier byte position pointers,
*       temporary storage for carry from the multiplication process, and
*       intermediate storage location pointer
*
            STA     35T,SP                  ;zero storage for multiplication carry
            LDA     #3                      ;load acc w/ 1st byte position
            STA     33T,SP                  ;pointer for multiplicand byte
            STA     34T,SP                  ;pointer for multiplier byte
            TSX                             ;transfer stack pointer + 1 to H:X
            AIX     #7                      ;position of 1st column in storage
            STHX    SPVAL                   ;pointer to interm. storage position
            CLRH                            ;clear h-reg
*
*       Multiply each byte of the multiplicand by each byte of the multiplier
*       and store the intermediate results
*
MULTLP      LDX     33T,SP                  ;load x-reg w/multiplicand byte pointer
            LDA     INTACC2,X               ;load acc with multiplicand
            LDX     34T,SP                  ;load x-reg w/ multiplier byte pointer
            LDX     INTACC1,X               ;load x-reg w/ multiplier
            MUL                             ;multiply
            ADD     35T,SP                  ;add carry from previous multiply
            BCC     NOINC32                 ;check for carry from addition
            INCX                            ;increment result MSB
NOINC32     STX     35T,SP                  ;move result MSB to carry
            LDHX    SPVAL                   ;load x-reg w/ storage position pointer
            STA     ,X                      ;store intermediate value
            AIX     #-1                     ;decrement storage pointer
            STHX    SPVAL                   ;store new pointer value
            CLRH                            ;clear h-reg
            DEC     34T,SP                  ;decrement multiplier pointer
            BPL     MULTLP                  ;multiply all four bytes of multiplier
                                            ;by one byte of the multiplicand
            LDHX    SPVAL                   ;load x-reg w/ storage position pointer
            LDA     35T,SP                  ;load acc w/ carry (MSB from last mult)
            STA     ,X                      ;store MSB of intermediate result
            AIX     #!11                    ;add offset for next intermediate
                                            ;result starting position
            STHX    SPVAL                   ;store new value
            CLRH                            ;clear h-reg
            CLR     35T,SP                  ;clear carry storage
            LDX     #3                      ;
            STX     34T,SP                  ;reset multiplier pointer
            DEC     33T,SP                  ;point to next multiplicand
            BPL     MULTLP                  ;loop until each multiplicand has been
                                            ;multiplied by each multiplier
*
```

```
*       Initialize temporary stack variables used in the addition process
*
                TSX                             ;transfer stack pointer to H:X
                AIX     #7                      ;add offset for LSB of result
                STHX    SPVAL                   ;store position of LSB
                CLR     35T,SP                  ;clear addition carry storage
                LDA     #7                      ;
                STA     33T,SP                  ;store LSB position of final result
                LDA     #3                      ;
                STA     34T,SP                  ;store counter for number of rows
*
*       add all four of the entries in each column together and store the
*       final 64-bit value in locations INTACC1:INTACC2+3.
*
OUTADDLP        LDA     35T,SP                  ;load acc with carry
                CLR     35T,SP                  ;clear carry
INADDLP         ADD     ,X                      ;add entry in table to accumulator
                BCC     ADDFIN                  ;check for carry
                INC     35T,SP                  ;increment carry
ADDFIN          AIX     #8                      ;load H:X with position of next entry
                                                ;column
                DEC     34T,SP                  ;decrement row counter
                BPL     INADDLP                 ;loop until all four entries in column
                                                ;have been added together
                CLRH                            ;clear h-reg
                LDX     #3                      ;
                STX     34T,SP                  ;reset row pointer
                LDX     33T,SP                  ;load final result byte pointer
                STA     INTACC1,X               ;store one byte of final result
                LDHX    SPVAL                   ;load original column pointer
                AIX     #-1                     ;decrement column pointer
                STHX    SPVAL                   ;store new pointer value
                DEC     33T,SP                  ;decrement final result byte pointer
                BPL     OUTADDLP                ;loop until all eight columns have
                                                ;been added up and the final results
                                                ;stored
*
*       Reset stack pointer and recover original registers values
*
                AIS     #35T                    ;deallocate local storage
                PULH                            ;restore h-reg
                PULX                            ;restore x-reg
                PULA                            ;restore accumulator
                RTS                             ;return
```

```
***************************************************************************************
***************************************************************************************
*
*       Signed 8 x 8 Multiply
*
*       This routine multiplies the signed 8-bit number stored in location
*       INTACC1 by the signed 8-bit number stored in location INTACC2
*       and places the signed 16-bit result in INTACC1:INTACC1+1.
*
*
SMULT8      EQU     *
            PSHX                        ;save x-reg
            PSHA                        ;save accumulator
            PSHH                        ;save h-reg
            AIS     #-1                 ;reserve 2 bytes of temp. storage
            CLR     1,SP                ;clear storage for result sign
            BRCLR   7,INTACC1,TEST2     ;check multiplier sign bit
            NEG     INTACC1             ;two's comp number if negative
            INC     1,SP                ;set sign bit for negative number
TEST2       BRCLR   7,INTACC2,SMULT     ;check multiplicand sign bit
            NEG     INTACC2             ;two's comp number if negative
            INC     1,SP                ;set or clear sign bit
SMULT       LDX     INTACC1             ;load x-reg with multiplier
            LDA     INTACC2             ;load acc with multiplicand
            MUL                         ;multiply
            STA     INTACC1+1           ;store result LSB
            STX     INTACC1             ;store result MSB
            LDA     1,SP                ;load sign bit
            CMP     #1                  ;check for negative
            BNE     RETURN              ;branch to finish if result is positive
            NEG     INTACC1+1           ;two's comp result LSB
            BCC     NOSUB               ;check for borrow from zero
            NEG     INTACC1             ;two's comp result MSB
            DEC     INTACC1             ;decrement result MSB for borrow
            BRA     RETURN              ;finished
NOSUB       NEG     INTACC1             ;two's comp result MSB without decrement
RETURN      AIS     #1                  ;deallocate temp storage
            PULH                        ;restore h-reg
            PULA                        ;restore accumulator
            PULX                        ;restore x-reg
            RTS                         ;return
```

```
******************************************************************************
******************************************************************************
*
*      Signed 16 x 16 multiply
*
*      This routine multiplies the signed 16-bit number in INTACC1:INTACC1+1 by
*      the signed 16-bit number in INTACC2:INTACC2+1 and places the signed 32-bit
*      value in locations INTACC1:INTACC1+3 (INTACC1 = MSB:INTACC1+3 = LSB).
*
SMULT16     EQU     *
            PSHX                        ;save x-reg
            PSHA                        ;save accumulator
            PSHH                        ;save h-reg
            AIS     #-1                 ;reserve 1 byte of temp. storage
            CLR     1,SP                ;clear storage for result sign
            BRCLR   7,INTACC1,TST2      ;check multiplier sign bit and negate
                                        ;(two's complement) if set
            NEG     INTACC1+1           ;two's comp multiplier LSB
            BCC     NOSUB1              ;check for borrow from zero
            NEG     INTACC1             ;two's comp multiplier MSB
            DEC     INTACC1             ;decrement MSB for borrow
            BRA     MPRSIGN             ;finished
NOSUB1      NEG     INTACC1             ;two's comp multiplier MSB (no borrow)
MPRSIGN     INC     1,SP                ;set sign bit for negative number
TST2        BRCLR   7,INTACC2,MLTSUB    ;check multiplicand sign bit and negate
                                        ;(two's complement) if set
            NEG     INTACC2+1           ;two's comp multiplicand LSB
            BCC     NOSUB2              ;check for borrow from zero
            NEG     INTACC2             ;two's comp multiplicand MSB
            DEC     INTACC2             ;decrement MSB for borrow
            BRA     MPCSIGN             ;finished
NOSUB2      NEG     INTACC2             ;two's comp multiplicand MSB (no borrow)
MPCSIGN     INC     1,SP                ;set or clear sign bit
MLTSUB      JSR     UMULT16             ;multiply INTACC1 by INTACC2
            LDA     1,SP                ;load sign bit
            CMP     #1                  ;check for negative
            BNE     DONE                ;exit if answer is positive,
                                        ;otherwise two's complement result
            LDX     #3                  ;
COMP        COM     INTACC1,X           ;complement a byte of the result
            DECX                        ;point to next byte to be complemented
            BPL     COMP                ;loop until all four bytes of result
                                        ;have been complemented
            LDA     INTACC1+3           ;get result LSB
            ADD     #1                  ;add a "1" for two's comp
            STA     INTACC1+3           ;store new value
            LDX     #2                  ;
TWSCMP      LDA     INTACC1,X           ;  add any carry from the previous
            ADC     #0                  ;  addition to the next three bytes
            STA     INTACC1,X           ;  of the result and store the new
            DECX                        ;  values
            BPL     TWSCMP              ;
DONE        AIS     #1                  ;deallocate temp storage on stack
            PULH                        ;restore h-reg
            PULA                        ;restore accumulator
            PULX                        ;restore x-reg
            RTS                         ;return
```

```
****************************************************************************
****************************************************************************
*
*     32 x 16 Unsigned Divide
*
*     This routine takes the 32-bit dividend stored in INTACC1:INTACC1+3
*     and divides it by the 16-bit divisor stored in INTACC2:INTACC2+1.
*     The quotient replaces the dividend and the remainder replaces the divisor.
*
UDVD32    EQU     *
*
DIVIDEND  EQU     INTACC1+2
DIVISOR   EQU     INTACC2
QUOTIENT  EQU     INTACC1
REMAINDER EQU     INTACC1
*
        PSHH                            ;save h-reg value
        PSHA                            ;save accumulator
        PSHX                            ;save x-reg value
        AIS     #-3                     ;reserve three bytes of temp storage
        LDA     #!32                    ;
        STA     3,SP                    ;loop counter for number of shifts
        LDA     DIVISOR                 ;get divisor MSB
        STA     1,SP                    ;put divisor MSB in working storage
        LDA     DIVISOR+1               ;get divisor LSB
        STA     2,SP                    ;put divisor LSB in working storage
*
*     Shift all four bytes of dividend 16 bits to the right and clear
*     both bytes of the temporary remainder location
*
        MOV     DIVIDEND+1,DIVIDEND+3   ;shift dividend LSB
        MOV     DIVIDEND,DIVIDEND+2     ;shift 2nd byte of dividend
        MOV     DIVIDEND-1,DIVIDEND+1   ;shift 3rd byte of dividend
        MOV     DIVIDEND-2,DIVIDEND     ;shift dividend MSB
        CLR     REMAINDER               ;zero remainder MSB
        CLR     REMAINDER+1             ;zero remainder LSB
*
*     Shift each byte of dividend and remainder one bit to the left
*
SHFTLP  LDA     REMAINDER               ;get remainder MSB
        ROLA                            ;shift remainder MSB into carry
        ROL     DIVIDEND+3              ;shift dividend LSB
        ROL     DIVIDEND+2              ;shift 2nd byte of dividend
        ROL     DIVIDEND+1              ;shift 3rd byte of dividend
        ROL     DIVIDEND                ;shift dividend MSB
        ROL     REMAINDER+1             ;shift remainder LSB
        ROL     REMAINDER               ;shift remainder MSB
*
```

```
*       Subtract both bytes of the divisor from the remainder
*
          LDA     REMAINDER+1             ;get remainder LSB
          SUB     2,SP                    ;subtract divisor LSB from remainder LSB
          STA     REMAINDER+1             ;store new remainder LSB
          LDA     REMAINDER               ;get remainder MSB
          SBC     1,SP                    ;subtract divisor MSB from remainder MSB
          STA     REMAINDER               ;store new remainder MSB
          LDA     DIVIDEND+3              ;get low byte of dividend/quotient
          SBC     #0                      ;dividend low bit holds subtract carry
          STA     DIVIDEND+3              ;store low byte of dividend/quotient
*
*       Check dividend/quotient LSB. If clear, set LSB of quotient to indicate
*       successful subraction, else add both bytes of divisor back to remainder
*
          BRCLR   0,DIVIDEND+3,SETLSB     ;check for a carry from subtraction
                                          ;and add divisor to remainder if set
          LDA     REMAINDER+1             ;get remainder LSB
          ADD     2,SP                    ;add divisor LSB to remainder LSB
          STA     REMAINDER+1             ;store remainder LSB
          LDA     REMAINDER               ;get remainder MSB
          ADC     1,SP                    ;add divisor MSB to remainder MSB
          STA     REMAINDER               ;store remainder MSB
          LDA     DIVIDEND+3              ;get low byte of dividend
          ADC     #0                      ;add carry to low bit of dividend
          STA     DIVIDEND+3              ;store low byte of dividend
          BRA     DECRMT                  ;do next shift and subtract

SETLSB    BSET    0,DIVIDEND+3            ;set LSB of quotient to indicate
                                          ;successive subtraction
DECRMT    DBNZ    3,SP,SHFTLP             ;decrement loop counter and do next
                                          ;shift
*
*       Move 32-bit dividend into INTACC1:INTACC1+3 and put 16-bit
*       remainder in INTACC2:INTACC2+1
*
          LDA     REMAINDER               ;get remainder MSB
          STA     1,SP                    ;temporarily store remainder MSB
          LDA     REMAINDER+1             ;get remainder LSB
          STA     2,SP                    ;temporarily store remainder LSB
          MOV     DIVIDEND,QUOTIENT       ;
          MOV     DIVIDEND+1,QUOTIENT+1   ;shift all four bytes of quotient
          MOV     DIVIDEND+2,QUOTIENT+2   ; 16 bits to the left
          MOV     DIVIDEND+3,QUOTIENT+3   ;
          LDA     1,SP                    ;get final remainder MSB
          STA     INTACC2                 ;store final remainder MSB
          LDA     2,SP                    ;get final remainder LSB
          STA     INTACC2+1               ;store final remainder LSB
*
*       Deallocate local storage, restore register values, and return from
*       subroutine
*
          AIS     #3                      ;deallocate temporary storage
          PULX                            ;restore x-reg value
          PULA                            ;restore accumulator value
          PULH                            ;restore h-reg value
          RTS                             ;return
```

```
****************************************************************************************
****************************************************************************************
*
*      Table Lookup and Interpolation
*
*      This subroutine performs table lookup and interpolation between two 16-bit
*      dependent variables (Y) from a table of up to 256 enties (512 bytes) and
*      allowing up to 256 interpolation levels between entries. INTACC1 contains
*      the position of ENTRY2 and INTACC1+1 contains the interpolation fraction.
*      The 16-bit result is placed in INTACC1+2=MSB, INTACC1+3=LSB. INTACC2 is
*      used to hold the two 16-bit entries during the routine.
*
*      Y = ENTRY1 + (INTPFRC(ENTRY2 - ENTRY1))/256
*
TBLINT    EQU       *
*
ENTNUM    EQU       INTACC1                  ;position of entry2 (0-255)
INTPFRC   EQU       INTACC1+1                ;interpolation fraction (1-255)/256
RESULT    EQU       INTACC1+2                ;16-bit interpolated Y value
ENTRY1    EQU       INTACC2                  ;16-bit enrty from table
ENTRY2    EQU       INTACC2+2                ;16-bit entry from table
*
          PSHH                               ;save h-register
          PSHA                               ;save accumulator
          PSHX                               ;save x-reg
          AIS       #-1                      ;allocate one byte of temp storage
          CLRH                               ;zero h-reg
          CLRA                               ;zero accumulator
          CLR       1,SP                     ;clear storage for difference sign
*
*      Load H:X with position of ENTRY2
*
          LDX       ENTNUM                   ;get position of entry2 (0-255)
          LSLX                               ;multiply by 2 (for 16-bit entries)
          BCC       GETENT                   ;if overflow from multiply occured,
                                             ;increment H-reg.
          INCA                               ;accumulator = 1
          PSHA                               ;push accumulator value on stack
          PULH                               ;transfer acc. value to h register
*
*      Get both entries from table, subtract ENTRY1 from ENTRY2 and store the
*      16-bit result.
*
GETENT    LDA       TABLE-2,x                ;get entry1 LSB
          STA       ENTRY1
          LDA       TABLE-1,x                ;get entry1 MSB
          STA       ENTRY1+1
          LDA       TABLE,x                  ;get entry2 MSB
          STA       ENTRY2
          LDA       TABLE+1,x                ;get entry2 LSB
          STA       ENTRY2+1
          SUB       ENTRY1+1                 ;entry2(LSB) - entry1(LSB)
          STA       RESULT+1                 ;store result LSB
          LDA       ENTRY2
          SBC       ENTRY1                   ;entry2(MSB) - entry1(MSB)
          STA       RESULT                   ;store result MSB
*
```

```
*
*     Two's complement 16-bit result if ENTRY1 was greater than ENTRY2, else
*     go do multiply
*
*
         TSTA                            ;test result MSB for negative
         BGE       MLTFRAC               ;go do multiply if postive
         INC       1,SP                  ;set sign flag for negative result
         NEG       RESULT+1              ;two's complement result LSB
         BCC       NODECR                ;check for borrow from zero
         NEG       RESULT                ;two's complement result MSB
         DEC       RESULT                ;decrement result MSB for borrow
         BRA       MLTFRAC               ;go do multiply
NODECR   NEG       RESULT                ;two's comp result MSB (no borrow)
*
*     (INTPFRC(RESULT:RESULT+1))/256 = Interpolated result
*
*     Multiply result by interpolation fraction
*
MLTFRAC  LDA       INTPFRC               ;get interpolation fraction
         LDX       RESULT+1              ;get result LSB
         MUL                             ;multiply
         STX       RESULT+1              ;store upper 8-bits of result and throw
                                         ;away lower 8-bits (divide by 256)
         LDA       INTPFRC               ;get interpolation fraction
         LDX       RESULT                ;get result MSB
         MUL                             ;multiply
         ADD       RESULT+1              ;add result LSB to lower 8-bits of
                                         ;product
         STA       RESULT+1              ;store new result LSB
         TXA                             ;get upper 8-bits of product
         ADC       #0                    ;add carry from last addition
         STA       RESULT                ;store result MSB
*
*     Y = ENTRY1 + Interpolated result
*
*     Check sign flag to determine if interpolated result is to be added to
*     or subtracted from ENTRY1
*
         TST       1,SP                  ;test sign flag for negative
         BLE       ADDVAL                ;if not set, add interpolated result
                                         ;to entry1, else subtract
         LDA       ENTRY1+1              ;get entry1 LSB
         SUB       RESULT+1              ;subtract result LSB
         STA       RESULT+1              ;store new result LSB
         LDA       ENTRY1                ;get entry1 MSB
         SBC       RESULT                ;subtact w/ carry result MSB
         STA       RESULT                ;store new result MSB
         BRA       TBLDONE               ;finished
ADDVAL   LDA       RESULT+1              ;get result LSB
         ADD       ENTRY1+1              ;add entry1 LSB
         STA       RESULT+1              ;store new result LSB
         LDA       ENTRY1                ;get entry1 MSB
         ADC       RESULT                ;add w/ carry result MSB
         STA       RESULT                ;store new result MSB
*
```

Application Note

```
*       Deallocate local storage, restore register values, and return from
*       subroutine.
*
TBLDONE  AIS      #1                  ;deallocate local storage
         PULX                         ;restore x-reg
         PULA                         ;restore accumulator
         PULH                         ;restore h-reg
         RTS                          ;return from subroutine
*
*       Sample of 16-bit table entries
*
TABLE    EQU      *
         FDB      !0000               ;entry 0
         FDB      !32767              ;entry 1
         FDB      !2416               ;entry 2
         FDB      !4271               ;entry 3
********************************************************************************
```


How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™ semiconductor