Freescale Semiconductor
Application Note

Document Number: AN3348
Rev. 0, 11/2006

Integer Math Routines for RS08

by: Murray Stewart
Applications Engineer
East Kilbride, Scotland

1 Introduction

This application note is based on *AN1219: M68HC08 Integer Math Routines* and is re-targeted for the RS08 family. Some significant code changes were required to implement the integer math routines on the RS08. These changes were required for several reasons. First, the RS08 uses a compact subset of the HCS08 instruction set. In particular, the core does not have an 8-bit multiplier or a negate (two's complement) instruction. Secondly, the RS08 uses a memory-paging scheme in place of extended addressing. Finally, the RS08 does not implement stack-relative addressing. Even with these limitations, the complex math routines were successfully implemented, showing the versatility of the RS08.

These subroutines are implemented in this application note:

- Unsigned and signed 8-bit × 8-bit multiplier
- Unsigned and signed 16-bit × 16-bit multiplier
- Unsigned 32-bit × 32-bit multiplier

Contents

- Unsigned 8-bit × 8-bit divide
- Unsigned 32-bit × 16-bit divide
- Lookup table and interpolate

This application note demonstrates very specific math routines; however, it will also be of great interest to any programer new to the RS08. The subroutines provide useful examples of programing features such as subroutine nesting using CodeWarrior™ macros and paging to access memory.

The first subroutine implemented is the unsigned 8-bit × 8-bit multiplier. Because this is implemented in software rather than hardware (as done in HCS08), you must load/store RAM locations rather than the core registers. The multiplier and multiplicand must be stored at memory locations MP and MA, respectively, and the 16-bit result can be loaded from RESULT. The 8-bit × 8-bit multiplier is then called by all the other multiplier subroutines in this application note. Other than the 8-bit × 8-bit multiplier, all the routines use INTACC1 and INTACC2 which are two continuous 4-byte global RAM locations that are used to input hexadecimal numbers to the subroutines and to return the results[1]. For proper operation of the following subroutines, these two storage locations must be allocated together but may be located anywhere in RAM address space.

As mentioned, the RS08 does not have the NEG (negate = two's complement) instruction. The most efficient way to generate the two's complement on RS08 is to load the accumulator with zero and then subtract from the accumulator the positive number to be converted. For example:

```
;Two's complement on RS08
LDA #00              ;load accumulator with 0
SUB <location>       ;subtract positive value
STA <location>       ;store at positive value location
```

This operation is used in all subroutines where two's complement is required.

2 Software Description

2.1 Unsigned 8 × 8 Multiply (UMULT8)

Entry conditions:

The two 8-bit numbers to be multiplied need to be saved at RAM locations MP (multiplier) and MA (multiplicand).

Exit conditions:

RAM locations RESULT (MSB) and RESULT+1 (LSB) contain the 16-bit multiplication result.

Size:

103 bytes

1. None of these six routines contained in this application note check for valid or non-zero numbers in the two integer accumulators. You must ensure proper values are placed in INTACC1 and INTACC2 before the subroutines are invoked.

RAM space:

7 bytes

Subroutine calls:

None

Procedure:

This routine multiplies the bytes MA and MP. The RAM location INDEX loops through each bit of MP in turn. If the Nth bit of MP is set, MA is multiplied by 2^N. Do this by using a left-shift by N. This intermediate result is stored at PP (MSB) and PP+1 (LSB). The running total of intermediate results are added and stored at RESULT and RESULT+1. This process is illustrated below:

MP = multiplier

MA = multiplicand

MA x MP

MA[0:7]
MP[0:7]

```
                MA[0:7] × MP[0] (= MA << 0 if MP[0] set)
+               MA[0:7] × MP[1] (= MA << 1 if MP[1] set)
+               MA[0:7] × MP[2] (= MA << 2 if MP[2] set)
+               MA[0:7] × MP[3] (= MA << 3 if MP[3] set)
+               MA[0:7] × MP[4] (= MA << 4 if MP[4] set)
+               MA[0:7] × MP[5] (= MA << 5 if MP[5] set)
+               MA[0:7] × MP[6] (= MA << 6 if MP[6] set)
+               MA[0:7] × MP[7] (= MA << 7 if MP[7] set)
```

= RESULT:RESULT+1

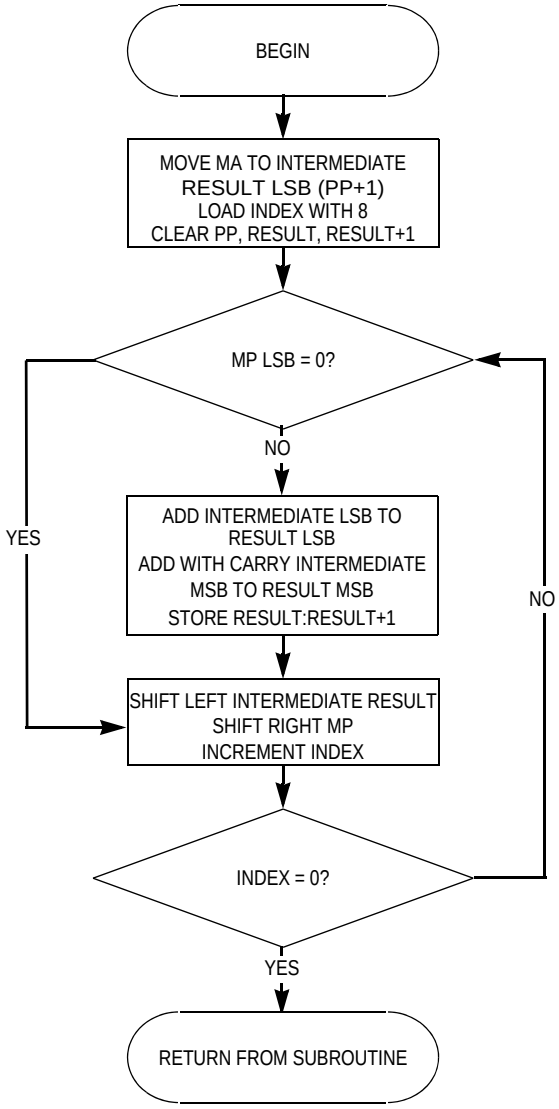

Figure 1. Unsigned 8 × 8 Multiply

2.2 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:

217 bytes

RAM space:

12 bytes (including those required for UMULT8)

Subroutine calls:

UMULT8

Procedure:

This routine multiplies the two left-most bytes of INTACC1 (INTACC1 = MSB, INTACC1 + 1 = LSB) by the two left-most bytes of INTACC2 (INTACC2 = MSB, INTACC2 + 1 = LSB). Temporary stack storage locations hold the two intermediate product. These intermediate products are then added, and the final 32-bit result is stored in INTACC1 (INTACC1 = MSB, INTACC1 + 3 = LSB). This process is illustrated below:

```
                                    INTACC1 x INTACC2
=                                   INTACC1 : INTACC1 +1
×                                   INTACC2 : INTACC2 +1
=   ------------------------------------------------------------
                          (INTACC1 : INTACC1 + 1) (INTACC2 + 1)
+    (INTACC1 : INTACC1 + 1) (INTACC2)
=   ------------------------------------------------------------
                  RES1_MSB   RES1_BYTE2   INTACC1 + 3
+   INTACC1   RES2_BYTE2     RES2_LSB
=   ------------------------------------------------------------
    INTACC1     INTACC +1   INTACC1 + 2    INTACC1 +3
```

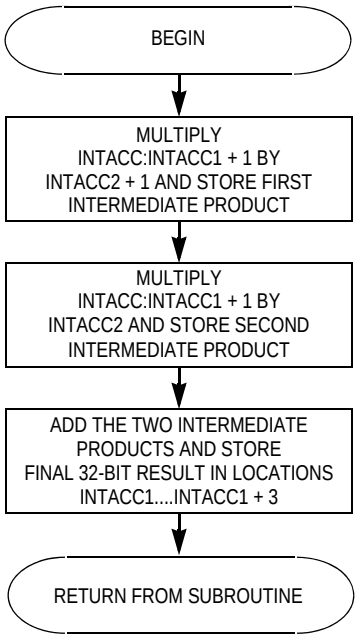

Figure 2. Unsigned 16 × 16 Multiply

2.3 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:

386 bytes

RAM space:

47 bytes (including those required for UMULT8)

Subroutine calls:

UMULT8

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 in RAM. These products are then added, and the final 8-byte result is placed in INTACC1.....INTACC2 + 3 (INTACC1 = MSB, INTACC2 + 3 = LSB). This process is illustrated below.

```
                                                INTACC1 x INTACC2

                                    INTACC1:INTACC1 + 1:INTACC1 + 2INTACC1 + 3
x                                   INTACC2:INTACC2 + 1:INTACC2 + 2INTACC2 + 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     IR15    IR19    IR23    IR27    IR31
+         0     0     IR10  IR14    IR18    IR22    IR26    0
+         0     IR5   IR9   IR13    IR17    IR21    0       0
+         IR0   IR4   IR8   IR12    IR16    0       0       0
  ____________________________________________________________________________
= INTACC1.....INTACC2 + 3
```

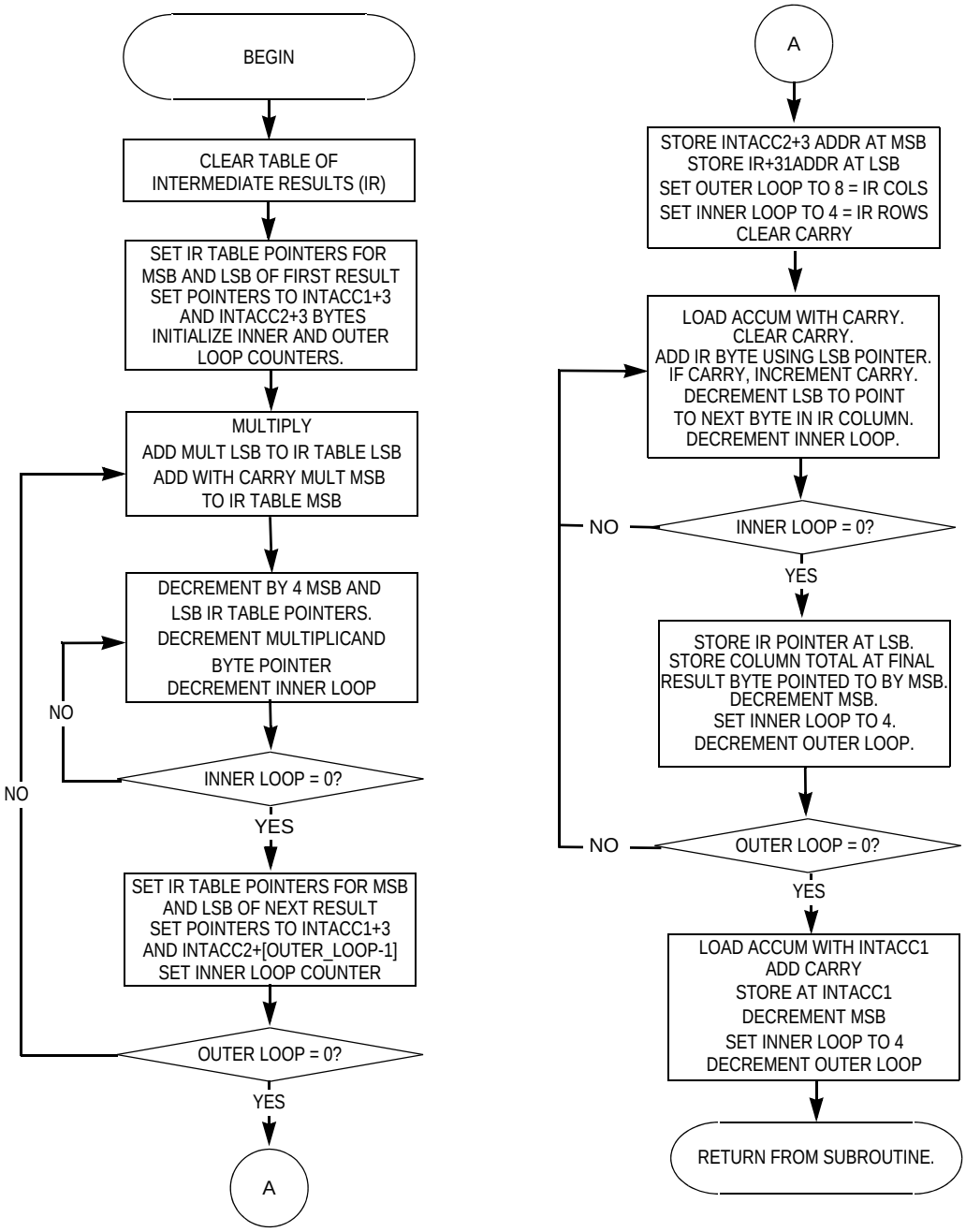

Figure 3. Unsigned 32 × 32 Multiply (Sheet 1 of 2)

2.4 Signed 8 × 8 Multiply (SMULT8)

Entry conditions:

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

Exit conditions:

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

Size:

184 bytes

RAM space:

8 bytes (including those required for UMULT8)

Subroutine calls:

UMULT8

Procedure:

This routine performs a signed multiply of INTACC1 (MSB) and INTACC2 (MSB). Before multiplying the two numbers, 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 RAM storage holds the result sign. If both numbers to be multiplied are negative or positive, the result sign LSB is cleared or set to indicate a negative result. Both numbers are then multiplied, 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.

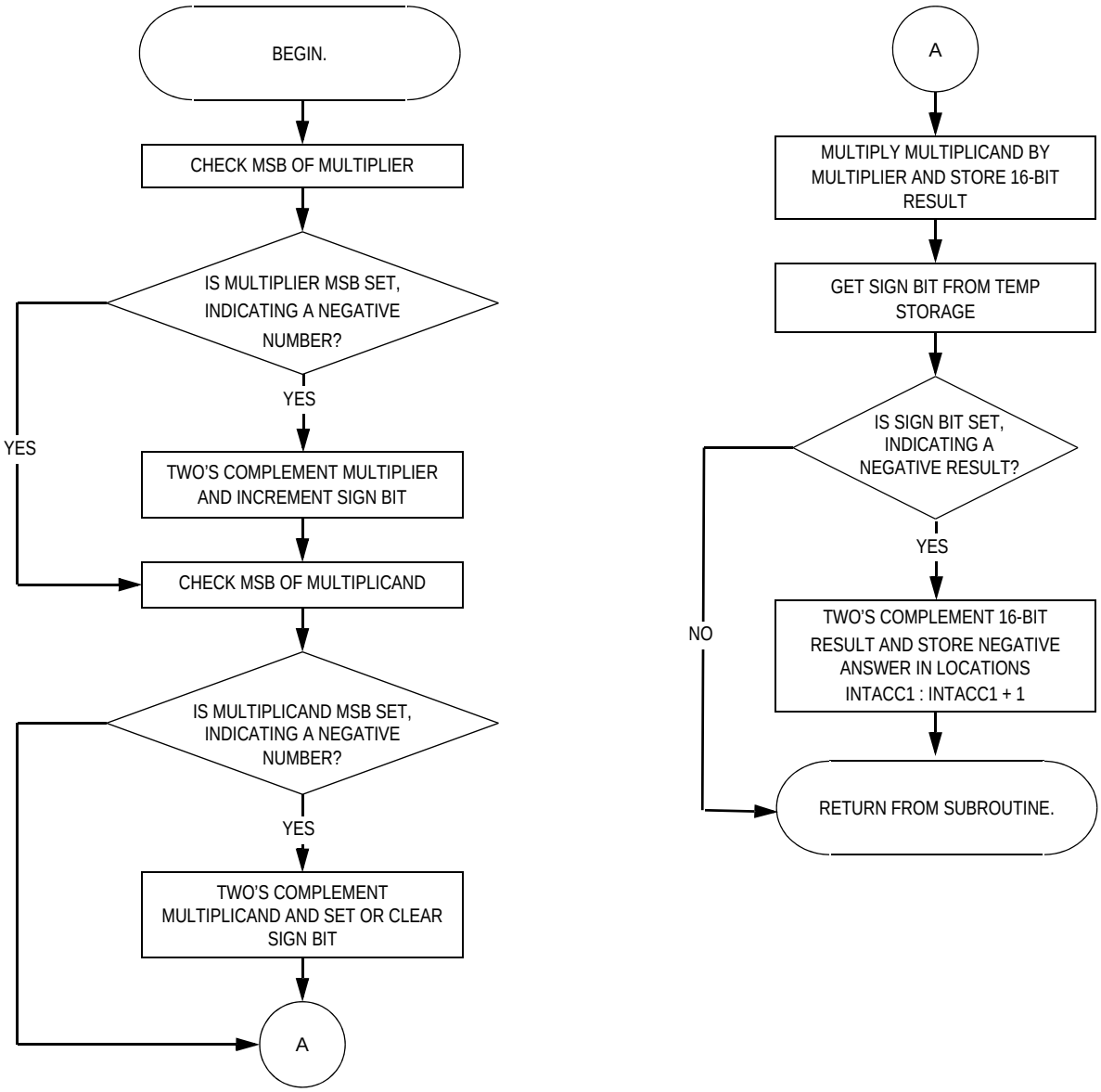

Figure 4. Signed 8 × 8 Multiply

2.5 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:

316 bytes

RAM space:

13 bytes (including those required for UMULT8)

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, 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 storage space is allocated for the result sign. If both 16-bit numbers to be multiplied are positive or negative, the result sign bit is cleared, indicating a positive result, otherwise the result sign bit is set. Subroutine UMULT16 is called to multiply the two unsigned 16-bit numbers, and the result is stored in locations INTACC1.....INTACC1 + 3 (INTACC1 = MSB, INTACC1+3 = LSB). After multiplication the subroutine is exited if the result sign bit is cleared. Otherwise, the two's complement of the result is taken to generate the correct sign, and INTACC1 is updated with the 32-bit negative result.

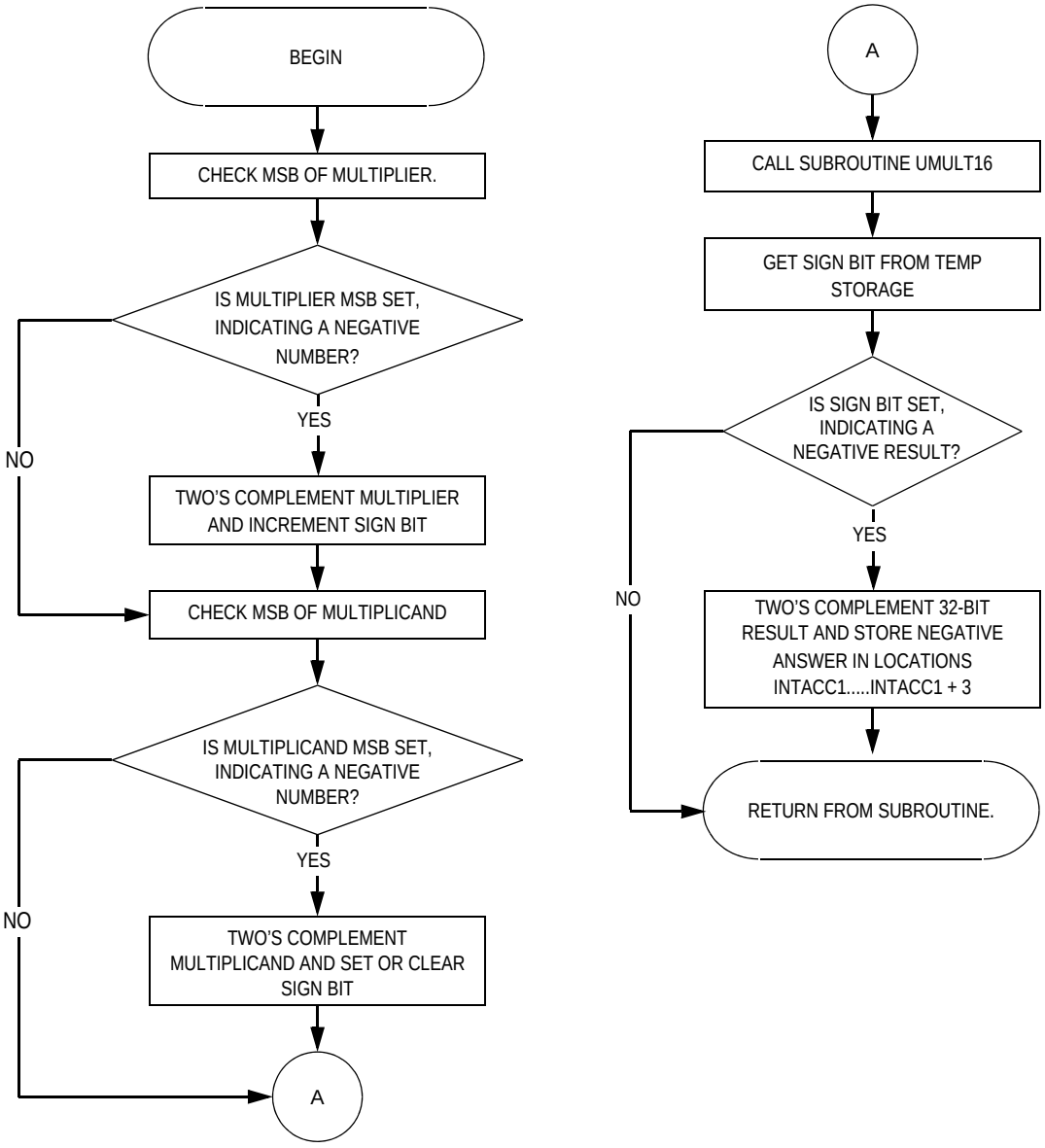

Figure 5. Signed 16 × 16 Multiply

2.6 8 × 8 Unsigned Divide (UDVD8)

Entry conditions:

INTACC1 contains the 8-bit unsigned dividend. INTACC2 contains the 8-bit unsigned divisor.

Exit conditions:

INTACC1 contains the 8-bit quotient. INTACC2 contains the 8-bit remainder.

Size:

69 bytes

RAM space:

4 bytes

Subroutine calls:

None

Procedure:

This routine takes the 8-bit dividend stored in INTACC1 and divides it by the divisor stored in INTACC2 using the standard shift-and-subtract algorithm. This algorithm first clears the 8-bit remainder, then shifts the dividend/quotient to the left one bit at a time until all eight bits of the dividend are shifted through the remainder, and the divisor subtracted from the remainder (illustrated below). Each time a trial subtraction succeeds, a 1 is placed in the LSB of the quotient. The 8-bit quotient is placed in locations INTACC1, and the remainder is returned in locations INTACC2.

Table 1. Before Subroutine Executes

INTACC1	INTACC2	INTACC2 + 1
dividend	divisor	0

Table 2. During Subroutine Execution

INTACC1 ⇐	INTACC2	INTACC2 + 1 ⇐
dividend	divisor	remainder
		– divisor

Table 3. After Returning from Subroutine

INTACC1	INTACC2	INTACC2 + 1
quotient	remainder	X

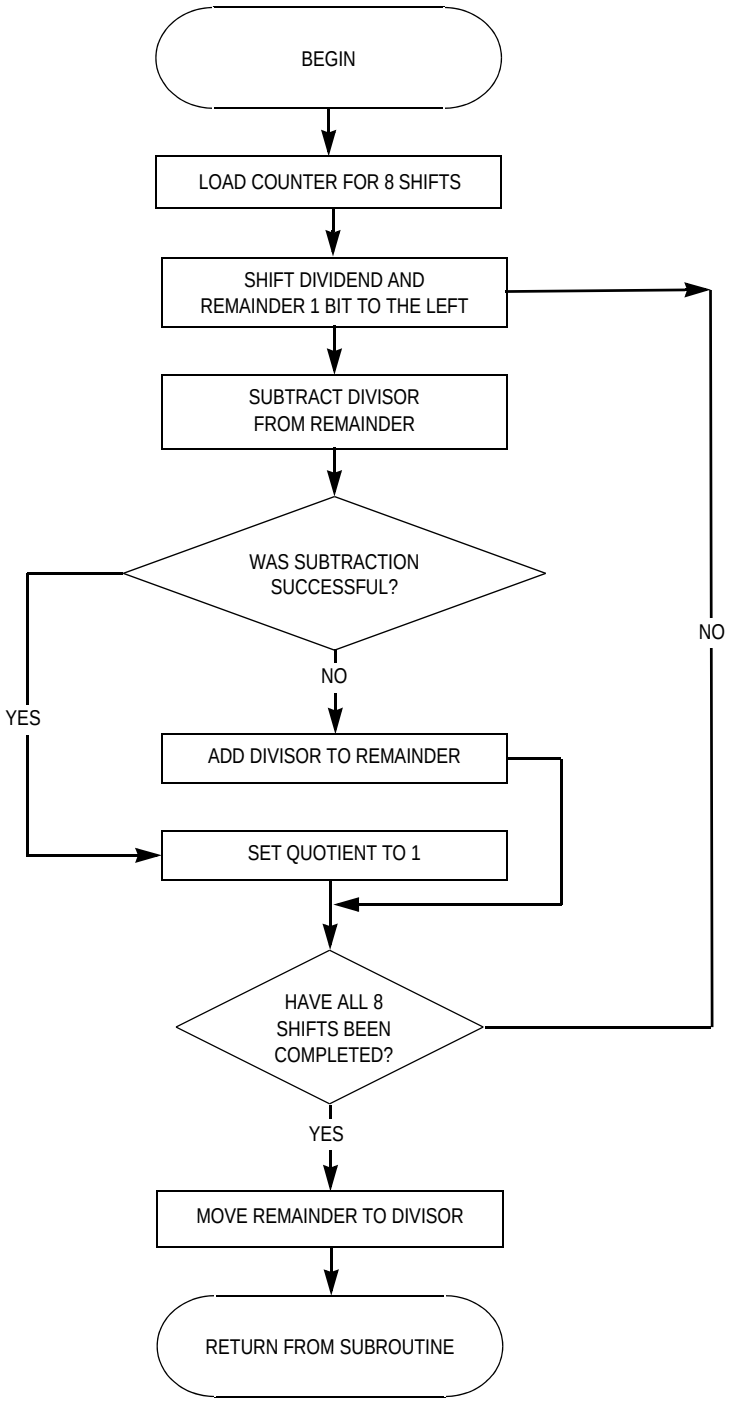

Figure 6. 8 × 8 Unsigned Divide

2.7 32 × 16 Unsigned Divide (UDVD32)

Entry conditions:

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

Exit conditions:

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

Size:

193 bytes

RAM space:

9 bytes

Subroutine calls:

None

Procedure:

This routine takes the 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 are shifted through the remainder, and the divisor subtracted from the remainder (illustrated below). 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.

Table 4. Before Subroutine Executes

INTACC1	INTACC1 + 1	INTACC1 + 2	INTACC1 + 3	INTACC2	INTACC2 + 1
dividend [MSB]	dividend	dividend	dividend [LSB]	divisor [MSB]	divisor [LSB]

Table 5. 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]				

Table 6. After Returning from Subroutine

INTACC1	INTACC1 + 1	INTACC1 + 2	INTACC1 + 3	INTACC2	INTACC2 + 1
quotient [MSB]	quotient	quotient	quotient [LSB]	remainder [MSB]	remainder [LSB]

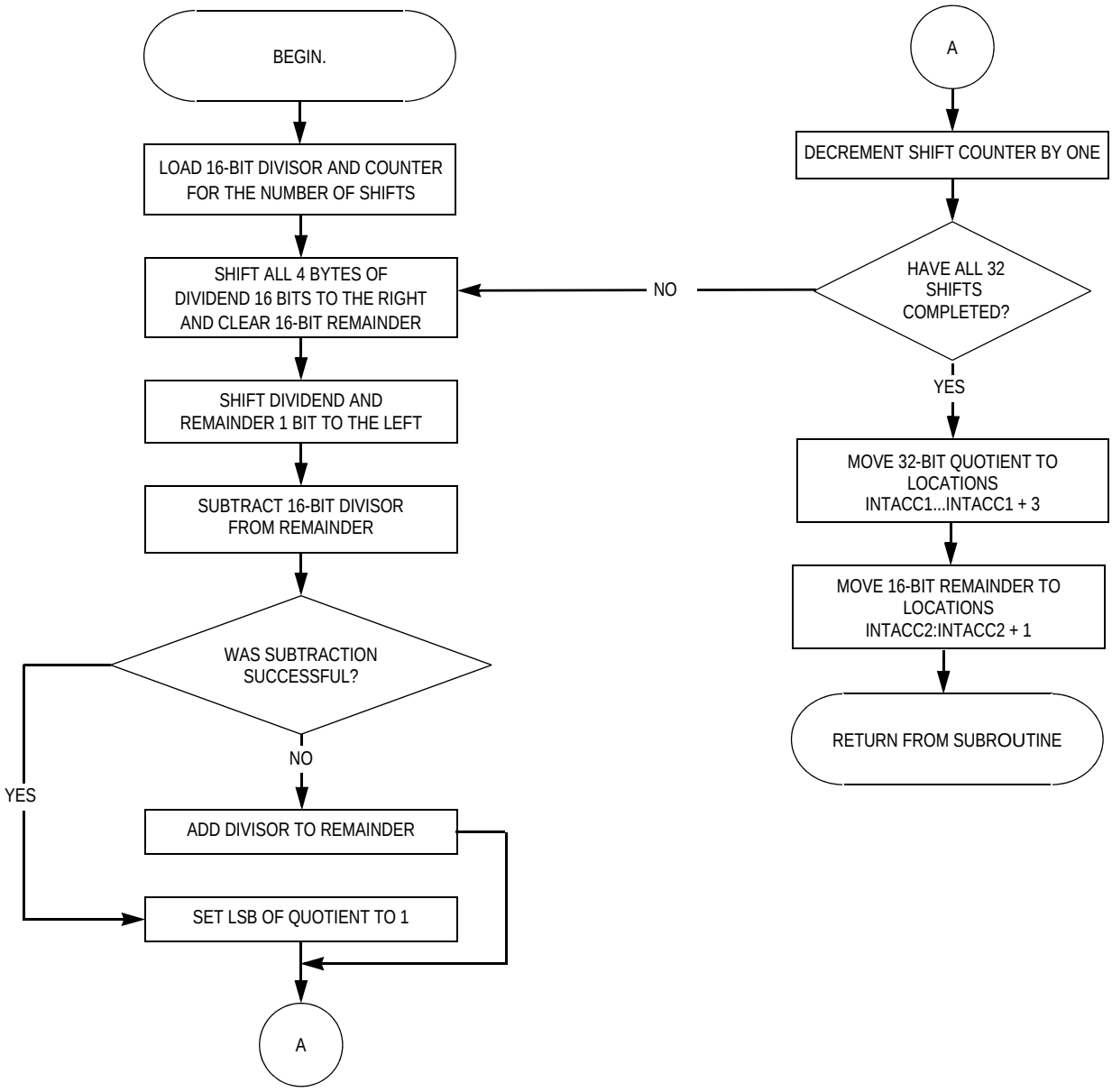

Figure 7. 32 × 16 Unsigned Divide

2.8 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

RAM space:

8 bytes (including those required for UMULT8)

Subroutine calls:

UMULT8

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 64k table of 16-bit entries can be compressed into just 256 16-bit entries). INTACC1 contains the position of table entry 2. INTACC1 + 1 contains the interpolation fraction. The unrounded 16-bit result is placed in INTACC1 + 2 = MSB, INTACC1 + 3 = LSB. INTACC2 holds 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 $0 < Y < 32767$
- INTPFRC = $(1 \leq X \leq 255) / 256$
- ENTRY1 and ENTRY2 can be within the range $0 < ENTRY < 32767$
- Slope of linear function can be positive or negative.
- The table of values can be located anywhere in the memory map.

Table 7. Example

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

— Find the interpolated Y value halfway between entry 146 and 147.

— ENTRY2 = Entry # 147 = 4271

— ENTRY1 = Entry # 146 = 2416

— For a 50% level of interpolation: INTPFRC = 128 / 256 = $80

— So:

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

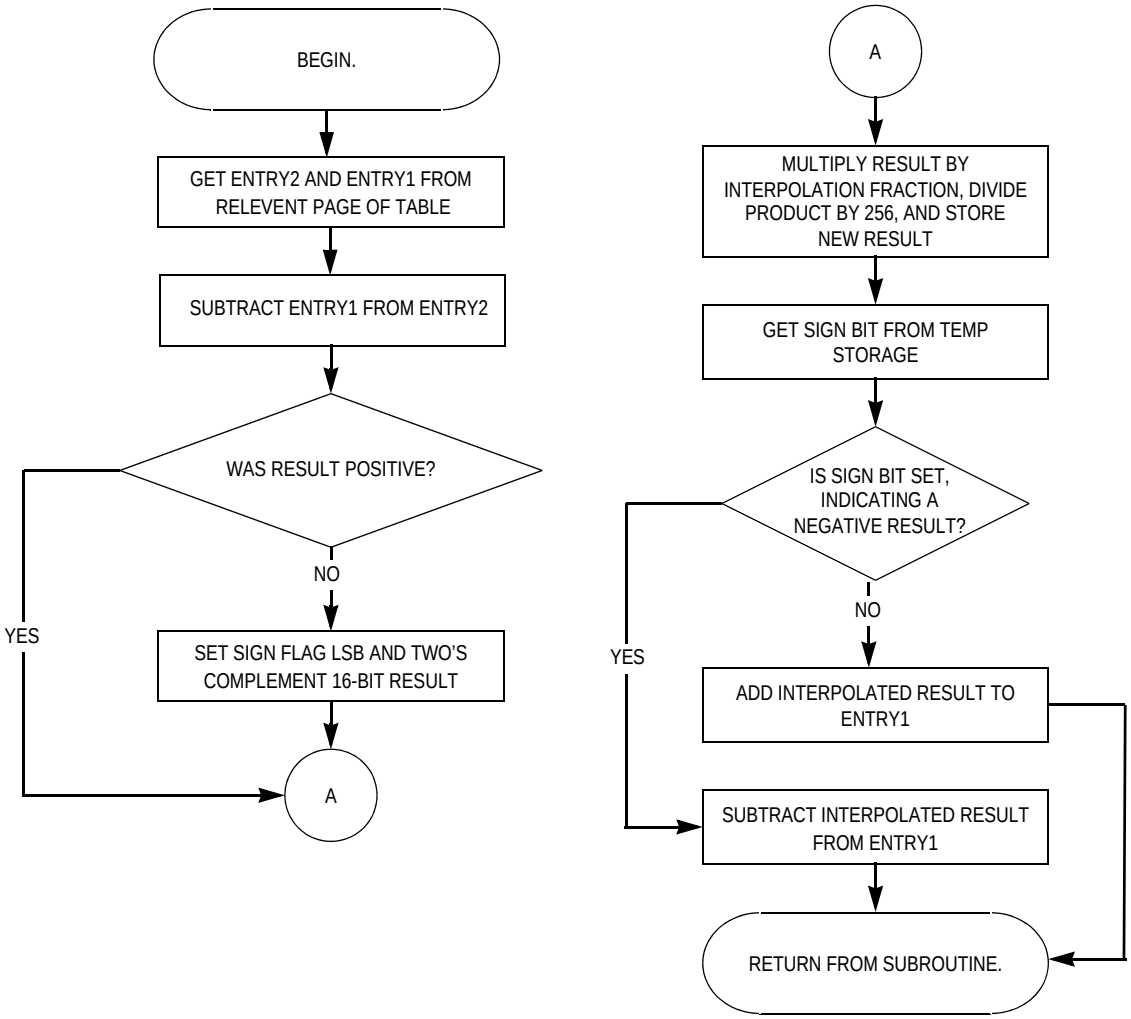

Figure 8. Table Lookup and Interpolation

3 Software Listing

```
;*****************************************************************************
;       Filename: main.asm
;       Revision: 1.00
;       Date: September 19, 2006
;       Updated By: Murray Stewart
;                   Freescale TSPG MCD Applications
;
;       Original By: Mark Johnson
;
;       Assembled Under: Code Warrior HC(R/S)08 V5.1
;
;       Revision 1.00   09/19/06 Original source from AN1219 updated for RS08
;
;*****************************************************************************
;       Program Description:
;*****************************************************************************
;
;       This program contains seven integer math routines for the RS08 family
;       of microcontrollers.
;
;       1. UMULT8   - Unsigned 8 bit x 8 bit multiply
;       2. UMULT16  - Unsigned 16 bit x 16 bit multiply
;       3. UMULT32  - Unsigned 32 bit x 32 bit multiply
;       4. SMULT8   - Signed 8 bit x 8 bit multiply
;       5. SMULT16  - Signed 16 bit x 16 bit multiply
;       6. UDVD8    - Unsigned 8 bit x 8 bit divide
;       7. UDVD32   - Unsigned 32 bit x 16 bit divide
;       8. TBLINT   - Table lookup and interpolation
;
;*****************************************************************************
; Macro to manage nested Subroutine entry code
;*****************************************************************************
ENTRY_CODE: MACRO
             SHA
             STA pcBUFFER+(2*(\1))
             SHA
             SLA
             STA pcBUFFER+(2*(\1))+1
             SLA
             ENDM

;*****************************************************************************
; Macro to manage nested Subroutine exit code
;*****************************************************************************
EXIT_CODE: MACRO
             LDA pcBUFFER+2*(\1)
             SHA
             LDA pcBUFFER+2*(\1)+1
             SLA
             ENDM

;*****************************************************************************
; Include derivative-specific definitions
;*****************************************************************************
             INCLUDE 'derivative.inc'
```

```
;*****************************************************************************
; export symbols
;*****************************************************************************
            XDEF _Startup
            ABSENTRY _Startup

;*****************************************************************************
; constants
;*****************************************************************************
MAXlevel     EQU   3            ;Max nesting depth for subroutine macro
HPAWS        EQU   $C0          ;Start address of High Page Access Window
LUT_START_PG EQU   $F7          ;LUT start page

;*****************************************************************************
; variable/data section
;*****************************************************************************
            ORG   TINY_RAMStart;Insert your TINY data definition here

pcBUFFER     DS.W   MAXlevel     ;Buffer for return address of nested subroutine macro
;                   =6 bytes
MA           DS.B   1           ;Multiplier for 8x8 bit Multiplier
MP           DS.B   1           ;Multiplicand for 8x8 bit Multiplier
PP           DS.B   2           ;16-bit PP for 8x8 bit Multiplier
RESULT       DS.B   2           ;16-bit Result for 8x8 bit Multiplier
INDEX:       DS.B   1           ;Bit index for 8x8 bit Multiplier
SIGNED:      DS.B   1           ;Storage for sign for signed multipliers
;                   =14 out of 14 bytes

            ORG    RAMStart    ;Insert your DIRECT data definition here

CARRY:       DS.B   1           ;Storage for carry for 16x16 bit multiplier
LSB:         DS.B   1           ;Pointer to intermediate result table LSB
MSB:         DS.B   1           ;Pointer to intermediate result table MSB
OUTER_LOOP:  DS.B   1           ;Outer loop counter
INNER_LOOP:  DS.B   1           ;Inner loop counter
LSB_64       DS.B   1           ;LSB of 64-bit result
MP_POINTER:  DS.B   1           ;Pointer to multiplier byte
MA_POINTER:  DS.B   1           ;Pointer to multiplicand byte
IR:          DS.B   32          ;Array of intermediate results (IR from now on)

INTACC1      DS.B   4           ;Multiplier input & Multiplier Output MSB
INTACC2      DS.B   4           ;Multiplicand input & Multiplier Output LSB
                    ;---
                    ;=48 out of 48 bytes
```

```
;*******************************************************************************
; code section
;*******************************************************************************
            ORG    ROMStart

_Startup

mainLoop
             ;Load subroutine inputs
             MOV   #$FF,INTACC1
             MOV   #$FF,INTACC1+1
             MOV   #$FF,INTACC2
             MOV   #$FF,INTACC2+1
             MOV   #$FF,INTACC1
             MOV   #$FF,INTACC1+1
             MOV   #$FF,INTACC2
             MOV   #$FF,INTACC2+1

             ;Jump to subroutine
             JSR   UMULT8
             JSR   SMULT8
             JSR   UMULT8
             JSR   SMULT16
             JSR   UMULT32
             JSR   UDVD32
             JSR   TBLINT
```

```
;*******************************************************************************
; 8 bit Unsigned Multiplier
;*******************************************************************************
;
; This routine multiplies the unsigned 8-bit number stored in
; location MP by the signed 8-bit number stored in location MA
; and places the signed 16-bit result in RESULT:RESULT+1.
;
UMULT8       ENTRY_CODE 2               ;call macro for nested subroutines

             MOV   MA,PP+1              ;store multiplicand in partial product LSB
             MOV   #8,INDEX             ;set number of loops
             CLR   PP                   ;zero partial product MSB
             CLR   RESULT+1             ;zero result LSB
             CLR   RESULT               ;zero partial product MSB

MULT         BRCLR 0,MP,NO_MULT         ;branch if MP[x] is zero
             LDA   PP+1                 ;load pp LSB
             ADD   RESULT+1             ;add pp LSB to result LSB
             STA   RESULT+1             ;store result LSB
             LDA   PP                   ;load pp MSB
             ADC   RESULT               ;add w/carry pp MSB to result MSB
             STA   RESULT               ;store result MSB

NO_MULT      LDA   PP+1                 ;load pp LSB
             LSLA                       ;shift left (x2)
             STA   PP+1                 ;store pp LSB
             LDA   PP                   ;load pp MSB
             ROLA                       ;rotate left with carry
             STA   PP                   ;store pp MSB
             LDA   MP                   ;load multiplier
             RORA                       ;rotate right
             STA   MP                   ;store multiplier
             DBNZ  INDEX,MULT           ;branch to mult if INDEX not zero

             EXIT_CODE 2                ;call macro for nested subroutines
             RTS
```

```
;*************************************************************************
; 16 Bit Unsigned Multiplier
;*************************************************************************
;
; 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.
;
;Need to re-use variables used in UMULT32
;
RES1_BYTE2  EQU IR     ;Intermediate results for unsigned 16x16 bit multiplier
RES1_MSB    EQU IR+1   ;      "         "      "     "       "    "      "
RES2_LSB    EQU IR+2   ;      "         "      "     "       "    "      "
RES2_BYTE2  EQU IR+3   ;      "         "      "     "       "    "      "

UMULT16
            ENTRY_CODE 1                ;call macro for nested subroutines

                                        ;clear local variables
            CLR   RES1_BYTE2
            CLR   RES1_MSB
            CLR   RES2_LSB
            CLR   RES2_BYTE2

                                        ;Multiply (INTACC1:INTACC1+1) by INTACC2+1

            MOV   INTACC1+1,MP          ;load w/multiplier LSB
            MOV   INTACC2+1,MA          ;load w/multiplicand LSB
            JSR   UMULT8                ;multiply
            MOV   RESULT,CARRY          ;save carry from multiply
            MOV   RESULT+1,INTACC1+3    ;store LSB of final result
            MOV   INTACC1,MP            ;load w/multiplier MSB
            MOV   INTACC2+1,MA          ;load w/multiplicand LSB
            JSR   UMULT8                ;multiply
            LDA   RESULT+1              ;load result LSB
            ADD   CARRY                 ;add carry from previous multiply
            STA   RES1_BYTE2            ;store 2nd byte of interm. result 1.
            LDA   RESULT                ;load mult msb
            BCC   NOINCA                ;check for carry from addition
            INCA                        ;
NOINCA      STA   RES1_MSB              ;store MSB of interm. result 1.
            CLR   CARRY                 ;clear storage for carry

                                        ;Multiply (INTACC1:INTACC1+1) by INTACC2

            MOV   INTACC1+1,MP          ;load w/multiplier LSB
            MOV   INTACC2,MA            ;load w/multiplicand MSB
            JSR   UMULT8                ;multiply
            MOV   RESULT,CARRY          ;save carry from multiply
            MOV   RESULT+1,RES2_LSB     ;store LSB of interm. result 2.
            MOV   INTACC1,MP            ;load w/multiplier MSB
            MOV   INTACC2,MA            ;load w/multiplicand MSB
            JSR   UMULT8                ;multiply
            LDA   RESULT+1              ;load result LSB
            ADD   CARRY                 ;add carry from previous multiply
            STA   RES2_BYTE2            ;store 2nd byte of interm. result 2.
```

```
            LDA   RESULT                ;multiply
            BCC   NOINCB                ;check for carry from addition
            INCA                        ;increment MSB of interm. result 2.
NOINCB      STA   INTACC1               ;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    RES1_BYTE2           ;load acc with 2nd byte of 1st result
            ADD    RES2_LSB             ;add acc with LSB of 2nd result
            STA    INTACC1+2            ;store 2nd byte of final result
            LDA    RES1_MSB             ;load acc with MSB of 1st result
            ADC    RES2_BYTE2           ;add w/ carry 2nd byte of 2nd result
            STA    INTACC1+1            ;store 3rd byte of final result
            LDA    INTACC1              ;load acc with MSB from 2nd result
            ADC    #0                   ;add any carry from previous addition
            STA    INTACC1              ;store MSB of final result

            EXIT_CODE 1                 ;call macro for nested subroutines
            RTS                         ;return
```

```
;*************************************************************************
; 32 Bit unsigned Multiplier
;*************************************************************************
;
; Each byte of the 32-bit multiplier (INTACC2) multiplies each byte of the
; 32-bit multiplicand to generate an intermediate result (IR).  The IR
; from each multiplication is placed in a the 32-byte IR table.
;
; Intermediate Result 1 = (INTACC1:INTACC1+1:INTACC1+2:INTACC1+3) x (INTACC2+3)
;                       = (IR+3, IR+7, IR+11, IR+15, IR+19, IR+23, IR+27, IR+31)
; Intermediate Result 2 = (INTACC1:INTACC1+1:INTACC1+2:INTACC1+3) x (INTACC2+2)
;                       = (IR+2, IR+6, IR+10, IR+14, IR+18, IR+22, IR+27, IR+31)
; Intermediate Result 3 = (INTACC1:INTACC1+1:INTACC1+2:INTACC1+3) x (INTACC2+1)
;                       = (IR+1, IR+5, IR+9,  IR+13, IR+17, IR+21, IR+25, IR+29)
; Intermediate Result 4 = (INTACC1:INTACC1+1:INTACC1+2:INTACC1+3) x (INTACC2)
;                       = (IR+0, IR+4, IR+8,  IR+12, IR+16, IR+19, IR+24, IR+28)
;
; Sum IRs to get final 64-bit result:
;
;             (IR+3, IR+7, IR+11, IR+15, IR+19, IR+23, IR+27, IR+31)
;           + (IR+2, IR+6, IR+10, IR+14, IR+18, IR+22, IR+26, IR+30)
;           + (IR+1, IR+5, IR+9,  IR+13, IR+17, IR+21, IR+25, IR+29)
;           + (IR+0, IR+4, IR+8,  IR+12, IR+16, IR+19, IR+24, IR+28)
;   ------------------------------------------------------------------------
; = (INTACC1:INTACC1+1:INTACC1+2:INTACC1+3:INTACC2:INTACC2+1:INTACC2+2:INTACC2+3)
;
UMULT32      ENTRY_CODE 0               ;call macro for nested subroutines

         ;Clear IR array

             LDA   #IR+31               ;load address of IR LSB
             STA   X                    ;store at X
CLEAR_IR     CLR   ,X                   ;clear address point to by X
             DEC   X                    ;decrement X
             LDA   #IR                  ;load address of IR MSB
             CMP   X                    ;compare with X
             BNE   CLEAR_IR             ;branch if not equal
             CLR   ,X                   ;to clear final IR

             MOV   #4,OUTER_LOOP        ;set OUTER_LOOP
             MOV   #4,INNER_LOOP        ;set INNER_LOOP
             MOV   #INTACC2+3,MP_POINTER ;set byte pointer to multiplier LSB
             MOV   #INTACC1+3,MA_POINTER ;set byte pointer to multiplicand LSB
             MOV   #IR+31,LSB           ;store IR LSB element in LSB
             MOV   #IR+27,MSB           ;store IR MSB element in MSB


         ;Compute 32 IRs

INNER        LDX   MP_POINTER           ;load X with MP_POINTER
             MOV   D[X],MP              ;store X data at MP
             LDX   MA_POINTER           ;load X with MA_POINTER
             MOV   D[X],MA              ;store X data at MA
             JSR   UMULT8               ;jump to 8-bit multiplier
             LDA   RESULT+1             ;load result LSB
             LDX   LSB                  ;load X with LSB address
             ADD   D[X]                 ;add LSB to acc
```

```
            STA   D[X]                ;store acc at LSB
            LDA   RESULT              ;load result MSB
            LDX   MSB                 ;load X with MSB address
            ADC   D[X]                ;add with carry MSB to acc
            STA   ,X                  ;store acc at MSB

            LDA   LSB                 ;load LSB address
            SUB   #4                  ;point 4 bytes down array
            STA   LSB                 ;store LSB address
            LDA   MSB                 ;load MSB address
            SUB   #4                  ;point 4 bytes down array
            STA   MSB                 ;store MSB address

            DEC   MA_POINTER          ;decrement multiplicand byte pointer
            DBNZ  INNER_LOOP,INNER    ;branch if not zero

            LDA   OUTER_LOOP          ;load OUTER_LOOP
            DECA                      ;decrement acc
            STA   OUTER_LOOP          ;store OUTER_LOOP
            CMP   #3                  ;compare
            BEQ   LOOP3               ;branch if equal
            CMP   #2
            BEQ   LOOP2
            CMP   #1
            BEQ   LOOP1
            BRA   EXIT_LOOP           ;branch always

LOOP3       MOV   #4,INNER_LOOP       ;set loop counter
            MOV   #INTACC2+2,MP_POINTER ;set byte pointer to next multiplier byte
            MOV   #INTACC1+3,MA_POINTER ;set byte pointer to multiplicand LSB
            MOV   #IR+26,LSB          ;store IR LSB element in LSB
            MOV   #IR+22,MSB          ;store IR MSB element in MSB
            BRA   INNER               ;branch always

LOOP2       MOV   #4,INNER_LOOP       ;set loop counter
            MOV   #INTACC2+1,MP_POINTER ;set byte pointer to next multiplier byte
            MOV   #INTACC1+3,MA_POINTER ;set byte pointer to multiplicand LSB
            MOV   #IR+21,LSB          ;store IR LSB element in LSB
            MOV   #IR+17,MSB          ;store IR MSB element in MSB
            BRA   INNER               ;branch always

LOOP1       MOV   #4,INNER_LOOP       ;set loop counter
            MOV   #INTACC2,MP_POINTER ;set byte pointer to next multiplier byte
            MOV   #INTACC1+3,MA_POINTER ;set byte pointer to multiplicand LSB
            MOV   #IR+16,LSB          ;store IR LSB element in LSB
            MOV   #IR+12,MSB          ;store IR MSB element in MSB
            BRA   INNER               ;branch always
EXIT_LOOP


;Sum IRs

            ;LSB/MSB variable reused. Names have no significance
            MOV   #INTACC2+3,MSB      ;load address of 2nd byte of 64-bit result
            MOV   #IR+31,LSB          ;load address of IR LSB column
            MOV   #8, OUTER_LOOP      ;load OUTER_LOOP
            MOV   #4, INNER_LOOP      ;load INNER_LOOP
```

```
            CLR   CARRY                 ;clear CARRY

OUTADDLP    LDA   CARRY                 ;load CARRY
            CLR   CARRY                 ;clear CARRY
            LDX   LSB                   ;load X with address of IR LSB column
INADDLP     ADD   D[X]                  ;add X data
            BCC   ADDFIN                ;check for carry
            INC   CARRY                 ;increment CARRY
ADDFIN      DECX                        ;decrement X address
            DBNZ  INNER_LOOP,INADDLP    ;branch if not zero

            STX   LSB                   ;store X address
            LDX   MSB                   ;load X address
            STA   D[X]                  ;store X data (final result for byte)
            DECX                        ;decrement X address
            STX   MSB                   ;store X address
            MOV   #4, INNER_LOOP        ;set INNER_LOOP
            DBNZ  OUTER_LOOP,OUTADDLP   ;branch if not zero

            EXIT_CODE 0                  ;call macro for nested subroutines
            RTS                         ;return
```

```
;*******************************************************************************
; 8 bit Signed Multiplier
;*******************************************************************************
;
; 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       ENTRY_CODE 0                ;call macro for nested subroutines

          ;Two's complement INTACC1 if negative

             CLR   SIGNED
            BRCLR 7,INTACC1,TEST          ;check multiplier sign bit
             LDA   #0                    ;load acc with zero for subtraction
             SUB   INTACC1               ;2's comp by sub from zero
             STA   INTACC1               ;store result
             INC   SIGNED                ;set sign bit for negative number

                                         ;Two's complement INTACC2 if negative

TEST         BRCLR 7,INTACC2,SMULT       ;check multiplicand sign bit
             LDA   #0                    ;load acc with zero for subtraction
             SUB   INTACC2               ;2's comp by sub from zero
             STA   INTACC2               ;store result
             INC   SIGNED                ;set or clear sign bit
SMULT        LDA   INTACC1               ;load multiplier
             STA   MP                    ;store multiplier
             LDA   INTACC2               ;load multiplicand
             STA   MA                    ;store multiplicand
             JSR   UMULT8                ;multiply

         ;Two's complement RESULT if negative

             LDA   SIGNED                ;load sign bit
             CMP   #1                    ;check for negative
             BNE   RETURN                ;branch to finish if result is positive
             LDA   #0                    ;load acc with zero for subtraction
             SUB   RESULT+1              ;2's comp by sub from zero
             STA   RESULT+1              ;store result
             BCC   NOSUB                 ;check for borrow from zero
             LDA   #0                    ;load acc with zero for subtraction
             SUB   RESULT                ;2's comp by sub from zero
             DECA                        ;decrement for borrow
             STA   RESULT                ;store result MSB
             BRA   RETURN                ;finished
NOSUB        LDA   #0                    ;load acc with zero for subtraction
             SUB   RESULT                ;2's comp by sub from zero
             STA   RESULT                ;store result MSB
RETURN       MOV   RESULT, INTACC1       ;move RESULT MSB
             MOV   RESULT+1, INTACC1+1   ;move RESULT LSB

             EXIT_CODE 0                 ;call macro for nested subroutines
             RTS                         ;return
```

```
;*************************************************************************
; 16 Bit signed Multiplier
;*************************************************************************
;
; 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      ENTRY_CODE 0               ;call macro for nested subroutines

        ;Two's complement INTACC1 if negative

             CLR   SIGNED               ;clear storage for result sign
             BRCLR 7,INTACC1,TEST2      ;check multiplier sign bit and negate
                                        ;(two's complement) if set
             LDA   #0                   ;load acc with zero for subtraction
             SUB   INTACC1+1            ;2's comp by sub from zero
             STA   INTACC1+1            ;store multiplier LSB
             BCC   NOSUB1               ;check for borrow from zero
             LDA   #0                   ;load acc with zero for subtraction
             SUB   INTACC1              ;2's comp by sub from zero
             DECA                       ;decrement MSB for borrow
             STA   INTACC1              ;store multiplier MSB
             BRA   MPRSIGN              ;finished
NOSUB1       LDA   #0                   ;load acc with zero for subtraction
             SUB   INTACC1              ;2's comp by sub from zero
             STA   INTACC1              ;store multiplicand LSB
MPRSIGN      INC   SIGNED               ;set sign bit for negative number

         ;Two's complement INTACC2 if negative

TEST2        BRCLR 7,INTACC2,MLTSUB     ;check multiplicand sign bit and negate
                                        ;(two's complement) if set
             LDA   #0                   ;load acc with zero for subtraction
             SUB   INTACC2+1            ;2's comp by sub from zero
             STA   INTACC2+1            ;store multiplicand LSB
             BCC   NOSUB2               ;check for borrow from zero
             LDA   #0                   ;load acc with zero for subtraction
             SUB   INTACC2              ;2's comp by sub from zero
             DECA                       ;decrement MSB for borrow
             STA   INTACC2              ;store multiplicand MSB
             BRA   MPCSIGN              ;finished
NOSUB2       LDA   #0                   ;load acc with zero for subtraction
             SUB   INTACC2              ;2's comp by sub from zero
             STA   INTACC2              ;store multiplicand MSB
MPCSIGN      INC   SIGNED               ;set or clear sign bit
MLTSUB       JSR   UMULT16              ;multiply INTACC1 by INTACC2
             LDA   SIGNED               ;load sign bit
             CMP   #1                   ;check for negative
             BNE   DONE                 ;exit if answer is positive,
                                        ;otherwise two's complement result

       ;Two's complement RESULT if negative

             LDA   #0
             SUB   INTACC1+3            ;complement a byte of the result
             STA   INTACC1+3            ;store result
```

```
            LDA   #0
            SBC   INTACC1+2           ;complement a byte of the result
            STA   INTACC1+2           ;store result
            LDA   #0
            SBC   INTACC1+1           ;complement a byte of the result
            STA   INTACC1+1           ;store result
            LDA   #0
            SBC   INTACC1             ;complement a byte of the result
            STA   INTACC1             ;store result

DONE        EXIT_CODE 0               ;call macro for nested subroutines
            RTS                       ;return
```

```
;*************************************************************************
; 8 x 8 Unsigned Divide
;*************************************************************************
;
; This routine takes the 8-bit dividend stored in INTACC1 and divides it by
; the 8-bit divisor stored in INTACC2.
; The quotient replaces the dividend and the remainder replaces the divisor.
;
UDVD8

DIVIDEND8    EQU   INTACC1
DIVISOR8     EQU   INTACC2
REMAINDER8   EQU   INTACC2+1
LOOP         EQU   IR

             MOV   #8,LOOP              ;loop counter for number of shifts
             CLR   REMAINDER8           ;zero remainder
;
;Shift dividend and remainder one bit to the left, ensuring carry from remainder
;goes to dividend and carry from dividend goes to remainder.
;

SHFTLP8      LDA   REMAINDER8           ;get remainder
             ROLA                       ;shift remainder MSB into carry
             LDA   DIVIDEND8            ;shift dividend
             ROLA
             STA   DIVIDEND8            ;store dividend
             LDA   REMAINDER8           ;shift remainder
             ROLA
             STA   REMAINDER8           ;store remainder

;
;Subtract divisor from the remainder
;
             LDA   REMAINDER8           ;get remainder
             SUB   DIVISOR8             ;subtract divisor from remainder
             STA   REMAINDER8           ;store new remainder
             LDA   DIVIDEND8            ;get dividend/quotient
             SBC   #0                   ;dividend holds subtract carry
             STA   DIVIDEND8            ;store dividend/quotient
;
;Check dividend/quotient. If clear, set LSB of quotient to indicate
;successful subraction, else add divisor back to remainder.
;
             BRCLR 0,DIVIDEND8,SETLSB8  ;check for a carry from subtraction
                                        ;and add divisor to remainder if set
             LDA   REMAINDER8           ;get remainder
             ADD   DIVISOR8             ;add divisor to remainder
             STA   REMAINDER8           ;store remainder LSB
             LDA   DIVIDEND8            ;get dividend
             ADC   #0                   ;add carry to dividend
             STA   DIVIDEND8            ;store dividend
             BRA   DECRMT8              ;do next shift and subtract
SETLSB8      BSET  0,DIVIDEND8          ;set LSB of quotient to indicate
                                        ;successive subtraction
DECRMT8      DBNZ LOOP,SHFTLP8          ;decrement loop counter and do next
                                        ;shift
```

```
        MOV  REMAINDER8,DIVISOR8    ;move remainder to divisor
        RTS                         ;return from subroutine
```

```
;*************************************************************************
; 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

DIVIDEND     EQU   INTACC1+2
DIVISOR      EQU   INTACC2
QUOTIENT     EQU   INTACC1
REMAINDER    EQU   INTACC1              ;used as temp variable and written to
                                        ;INTACC2:INTACC2+1 at end of routine
SP1          EQU   IR
SP2          EQU   IR+1
SP3          EQU   IR+2

             MOV   #32,SP3              ;loop counter for number of shifts
             MOV   DIVISOR,SP1          ;put divisor MSB in working storage
             MOV   DIVISOR+1,SP2        ;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
             LDA   DIVIDEND+3            ;shift dividend LSB
             ROLA                        ;shift dividend LSB into carry
             STA   DIVIDEND+3            ;store dividend LSB
             LDA   DIVIDEND+2            ;shift 2nd byte of dividend
             ROLA
             STA   DIVIDEND+2            ;store 2nd byte of dividend
             LDA   DIVIDEND+1            ;shift 3rd byte of dividend
             ROLA
             STA   DIVIDEND+1            ;store 3rd byte of dividend
             LDA   DIVIDEND              ;shift dividend MSB
             ROLA
             STA   DIVIDEND              ;store dividend MSB
             LDA   REMAINDER+1           ;shift remainder LSB
             ROLA
             STA   REMAINDER+1           ;store remainder LSB
             LDA   REMAINDER             ;shift remainder MSB
             ROLA
             STA   REMAINDER             ;store remainder MSB
```

```
        ;Subtract both bytes of the divisor from the remainder

            LDA   REMAINDER+1           ;get remainder LSB
            SUB   SP2                   ;subtract divisor LSB from remainder LSB
            STA   REMAINDER+1           ;store new remainder LSB
            LDA   REMAINDER             ;get remainder MSB
            SBC   SP1                   ;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 subtraction, 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   SP2                   ;add divisor LSB to remainder LSB
            STA   REMAINDER+1           ;store remainder LSB
            LDA   REMAINDER             ;get remainder MSB
            ADC   SP1                   ;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 SP3,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   SP1                   ;temporarily store remainder MSB
            LDA   REMAINDER+1           ;get remainder LSB
            STA   SP2                   ;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   SP1                   ;get final remainder MSB
            STA   INTACC2               ;store final remainder MSB
            LDA   SP2                   ;get final remainder LSB
            STA   INTACC2+1             ;store final remainder LSB

            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
;
; Lookup Table placed in Flash at page "LUT_START_PG". From
; position of ENTRY2 need to find the page of the memory location.
; Care has to be taken as adjacent data may be at page boundary.
;
TBLINT   ENTRY_CODE 0             ;call macro for nested subroutines

ENTNUM       EQU   INTACC1        ;position of entry2 (0-255)
INTPFRC      EQU   INTACC1+1      ;interpolation fraction (1-255)/256
RES          EQU   INTACC1+2      ;16-bit interpolated Y value
ENTRY1       EQU   INTACC2        ;16-bit entry from table
ENTRY2       EQU   INTACC2+2      ;16-bit entry from table

             CLR   SIGNED         ;clear
;
;Get ENTRY2 Page
;
;256 entries spread over 8 x 64k pages.
;Therefore 3 MSBs of ENTNUM give page number.
;
             LDA   ENTNUM         ;get position of ENTRY2 (0-255)
             AND   #$E0           ;mask 3 MSBs to find page number of ENTRY2
             RORA                 ;rotate right 5 times to get page number (0-7)
             RORA
             RORA
             RORA
             RORA
             ADD   #LUT_START_PG ;add page number of first location of LUT
             STA   PAGESEL
;
;Get ENTRY2 Data
;
;6 LSBs of ENTNUM give address on page of ENTNUM LSB.
;Add 1 to address to get MSB.
;

             LDA    ENTNUM         ;get position of ENTRY1 (0-255)
             LSLA                  ;multiply by 2 (for 16-bit entries)
             AND    #$3F           ;remove 2 MSBs as 64 byte paged memory
             ADD    #HPAWS         ;add start address of High Page Access Window
             STA    X             ;store position at X
             MOV    D[X],ENTRY2    ;store ENTRY2 MSB
             INCX                  ;increment X for ENTRY2 LSB
             MOV    D[X],ENTRY2+1 ;store ENTRY2 LSB
```

```
;
;Get ENTRY1 Page
;
             LDA   ENTNUM        ;get position of ENTRY1 (0-255)
             DECA                ;decrement to find position of ENTRY1
             AND   #$E0          ;mask 3 MSBs to find page number of ENTRY1
             RORA                ;rotate right 5 times to get page number (0-7)
             RORA
             RORA
             RORA
             RORA
             ADD   #LUT_START_PG ;add page number of first location of LUT
             STA   PAGESEL       ;store ENTRY1 page number
;
;Get ENTRY1 Data
;
             LDA   ENTNUM        ;get position of ENTRY2 (0-255)
             DECA                ;decrement for position of ENTRY1
             LSLA                ;multiply by 2 (for 16-bit entries)
             AND   #$3F          ;remove 2 MSBs as 64 byte paged memory
             ADD   #HPAWS        ;add start address of High Page Access Window
             STA   X             ;store position at X
             MOV   D[X],ENTRY1   ;store ENTRY1 MSB
             INCX                ;increment X for ENTRY1 LSB
             MOV   D[X],ENTRY1+1 ;store ENTRY1 LSB
;
;Subtract ENTRY1 from ENTRY2 and store the 16-bit result.
;
             LDA   ENTRY2+1
             SUB   ENTRY1+1      ;entry2(LSB) - entry1(LSB)
             STA   RES+1         ;store result LSB
             LDA   ENTRY2
             SBC   ENTRY1        ;entry2(MSB) - entry1(MSB)
             STA   RES           ;store result MSB
;
;Two's complement 16-bit result if ENTRY1 was greater than ENTRY2, else
;go do multiply
;
             AND   #$80          ;check MSB for sign
             BEQ   MLTFRAC       ;go do multiply if positive
             MOV   #1, SIGNED    ;set sign flag for negative result
             LDA   #0            ;load acc with zero for subtraction
             SUB   RES+1         ;2's comp by sub from zero
             STA   RES+1         ;store in RES LSB
             BCC   NODECR        ;check for borrow from zero
             LDA   #0            ;load acc with zero for subtraction
             SUB   RES           ;2's comp by sub from zero
             STA   RES           ;store in RES MSB
             DEC   RES           ;decrement result MSB for borrow
             BRA   MLTFRAC       ;go do multiply
NODECR       LDA   #0            ;load acc with zero for subtraction
             SUB   RES           ;2's comp by sub from zero
             STA   RES           ;store in RES MSB
;
;(INTPFRC(RES:RES+1))/256 = Interpolated result
;
;Multiply result by interpolation fraction
```

```
;
MLTFRAC      LDA   INTPFRC      ;get interpolation fraction
             STA   MP           ;store at multiplier
             LDA   RES+1        ;get result LSB
             STA   MA           ;store at multiplicand
             JSR   UMULT8       ;branch to multiplier
             MOV   RESULT, RES+1;store upper 8-bits of result and throw
                                ;away lower 8-bits (divide by 256)
             LDA   INTPFRC      ;get interpolation fraction
             STA   MP           ;store at multiplier
             LDA   RES          ;get result MSB
             STA   MA           ;store at multiplicand
             JSR   UMULT8       ;branch to multiplier
             LDA   RESULT+1     ;UMULT RESULT LSB
             ADD   RES+1        ;add result LSB to lower 8-bits of
                                ;product
             STA   RES+1        ;store new result LSB
             LDA   RESULT       ;UMULT RESULT MSB
             ADC   #0           ;add carry from last addition
             STA   RES          ;store result MSB
;
; Y = ENTRY1 + Interpolated result
;
; Check sign flag to determine if interpolated result is to be added to
; or subtracted from ENTRY1
;
             LDA   SIGNED       ;test sign flag for negative
             BEQ   ADDVAL       ;if not set, add interpolated result
                                ;to entry1, else subtract
             LDA   ENTRY1+1     ;get entry1 LSB
             SUB   RES+1        ;subtract result LSB
             STA   RES+1        ;store new result LSB
             LDA   ENTRY1       ;get entry1 MSB
             SBC   RES          ;subtact w/ carry result MSB
             STA   RES          ;store new result MSB
             BRA   TBLDONE      ;finished
ADDVAL       LDA   RES+1        ;get result LSB
             ADD   ENTRY1+1     ;add entry1 LSB
             STA   RES+1        ;store new result LSB
             LDA   ENTRY1       ;get entry1 MSB
             ADC   RES          ;add w/ carry result MSB
             STA   RES          ;store new result MSB
;
;Return from subroutine.
;
TBLDONE      EXIT_CODE 0         ;call macro for nested subroutines
             RTS                ;return from subroutine
```

```
;*************************************************************
;*                  Startup Vector                           *
;*************************************************************
;
            ORG   $3FFD

            JMP _Startup           ; Reset

;*************************************************************
;*                  Lookup Table                             *
;*************************************************************
;256 x 2 byte LUT = 512 byte
;Require 8 x 64 byte pages
;Place LUT at end of flash but remember to leave bottom page for
;Startup Vector
;
;     Data          Page       Start Address
;  ----------------------------------------
;  Startup Vector    FF            3FC0
;  LUT Page 7        FE            3F80
;  LUT Page 6        FD            3F40
;  LUT Page 5        FC            3F00
;  LUT Page 4        FB            3EC0
;  LUT Page 3        FA            3E80
;  LUT Page 2        F9            3E40
;  LUT Page 1        F8            3E00
;  LUT Page 0        F7            3DC0

            ORG   $3DC0

FDATA:      DC.W     0             ;Entry 0
            DC.W     32767         ;Entry 1
            DC.W     2416          ;Entry 2
            DC.W     4271          ;Entry 3
```

THIS PAGE IS INTENTIONALLY BLANK

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

Document Number: AN3348
Rev. 0
11/2006

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

freescale™
semiconductor