Application Report
SPRA515

Understanding TMS320C62xx DSP Single-Precision Floating-Point Functions

Syd Poland *Technical Training Organization*

Abstract

This application report describes the 32-bit single-precision (SP) floating-point (FP) functions of the Texas Instruments (TI™) TMS320C62xx digital signal processor (DSP). The SP FP functions provide fast assembly language subroutines that are C callable. This document is organized around the the following topics:

- Descriptions
- Run-time library functions
- Programs size and timing estimates
- Assembly (ASM) listings
- Linear assembly (SA) listings
- Performance

Contents

Notice

Customer Workshop Team
Semiconductor Training Technical Organization
Texas Instruments Incorporated
7834 Churchill Way, MS 3984
Dallas, TX 75251-1903
(972) 917-3894

Revision History

August, 1998 1st release

Description of Single-Precision Floating-Point Functions

This section describes the following areas of the single-precision floating-point functions :

- Format
- Special cases
- Rounding
- Conventions for passing input arguments
- Convention for returning answers
- Temporary registers
- Validation

Format

The 32-bit IEEE-754 standard FP format is used:

- Bit 31 = sign bit (0 = positive, 1 = negative numbers)
- Bits 30–23 = exponent field as a power of 2 with a bias of 127 (exponent of 1.0)
- Bits 22–0 = fractional absolute mantissa that has an implied 1(bit 23) so that the full mantissa is *1.mantissa* when the exponent is non-zero and represents about 7 decimal digits (16,777,215 is maximum)
- Number = $(-1)^{sign} * (1.mantissa) * 2^{(exp -127)}$ for non-zero exponents
- Smallest FP number = `0x0080_0000` with exponent of 1 is 1.175494×10^{-38} (1.0×2^{-126})
- Largest FP number = `0x7f7f_ffff` with exponent of 254 is $3.402823 \times 10^{+38}$ ($2.0 \times 2^{+127}$)

NOTE:
The prefix `0x` indicates the number is hexadecimal.

Special Cases

FP representation is unique in the following cases:

- Zero—the entire 32-bits are 0 (software never returns –0).
- Underflow—this occurs when the exponent < 1; the answer returned is 0.
- Overflow—this occurs when the exponent > 254; one of two possible answers is returned:
  - Positive infinity = `0x7fff_ffff` (exponent and mantissa = 1s)
  - Negative infinity = `0xffff_ffff` (all fields = 1s)

- Denormal—zero exponent, non-zero mantissa; denormal FP numbers are never generated. If the exponent = 0, the sign and mantissa are returned as zero.
- Signaling Not-a-Number (SNaN) is treated as infinity. SNaN has:
  - Sign—0 or 1
  - Exponent—255
  - Mantissa—bit 22 = 0 and remaining mantissa bits are arbitrary
- Quiet Not-a-Number (QNaN) is treated as infinity. QNaN has:
  - Sign—0 or 1
  - Exponent—255
  - Mantissa—bit 22 = 1 and remaining mantissa bits are arbitrary
- Infinity—the C67xx uses the following format for infinity:
  - Sign—0 or 1
  - Exponent—255
  - Mantissa—all 0s

NOTE:
The infinite value returned by these C62xx functions is an exponent and mantissa of all 1s.

Rounding

The rounding mode adopted here is *round-to-zero*. No provisions are made for:

- Round-to-nearest (half adjust)
- Round-to-positive-infinity
- Round-to-negative-infinity

Conventions for Passing Input Arguments

Conventions for passing arguments are described next.

- 32-bit input integers—signed or unsigned values are in register **A4**
- 40-bit input integers—signed or unsigned values are in registers **A5:A4**
- 32-bit SP FP input numbers are passed as follows:
- Argument 1 is in register **A4**.
- Argument 2 is in register **B4** (if needed).

Convention for Returning Answers

The following conventions are used for returning answers:

- 32-bit output integers—signed or unsigned values are in register **A4**
- 40-bit output integers—signed or unsigned values are in registers **A5:A4**
- 32-bit SP FP output numbers are returned in register **A4**.

Temporary Registers

In accordance with the C software convention, registers **A0-A9** and **B0-B9** (except **B3** with the return address) are considered to be temporary registers and thus are not saved or restored. Any registers **A10-A15** and **B10-B15** that are used are saved and restored before returning from the subroutine.

Validation

These routines were validated by testing them with the Tools version 2.00 *rts6201.lib* routines and also when possible using the equivalent C6701 floating-point hardware instructions.

Run-Time Library Functions

The C62xx *rts6201.lib* functions that have been coded here include the following (for more information on these functions, see the *TMS320C6x Optimizing C Compiler User's Guide*, literature number SPRU187C):

NOTE:
C67xx hardware instructions are shown in [square brackets].

- `_addf` = SP FP addition [ADDSP on C67xx]
- _divf = SP FP division

  [C67xx RCPSP performs an 8-bit table look-up of the reciprocal of the divisor. Two software Newton-Rhapson iterations are performed to obtain the SP FP quotient.]
- `_fixfi` = convert SP FP to 32-bit signed integer [SPINT on C67xx]
- `_fixfli` = convert SP FP to 40-bit signed long integer
- `_fixfu` = convert SP FP to 32-bit unsigned integer
- `_fixful` = convert SP FP to 40-bit unsigned long integer
- `_fltif` = convert 32-bit signed integer to SP FP [INTSP on C67xx]
- `_fltlif` = convert 40-bit signed long integer to SP FP
- `_fltuf` = convert 32-bit unsigned integer to SP FP [INTSPU on C67xx]
- `_fltulf` = convert 40-bit unsigned long integer to SP FP
- `_mpyf` = SP floating-point multiplication [MPYSP on C67xx]
- `_subf` = SP floating-point subtraction [SUBSP on C67xx]

Program Size and Timing Estimates

Name	C67xx H/W	ASM Fetch Packets	ASM Instructions	ASM cycles = Execution Packets	Million/s at 200 MHz (ASM)
_addf	ADDSP	7	46	15	13.3
_divf	RCPSP	7	47	42	4.8
_fixfi	SPINT	4	20	7	28.6
_fixfli	---	3	21	8	25.0
_fixfu	---	2	16	7	28.6
_fixful	---	3	20	7	28.6
_fltif	INTSP	2	12	6	33.3
_fltlif	---	2	15	7	28.6
_fltuf	INTSPU	2	10	6	33.3
_fltulf	---	2	12	6	33.3
_mpyf	MPYSP	6	41	13	15.4
_subf	SUBSP	7	46	15	13.3
Totals		46	306	138	
Average		3.8	25.5	11.5	21.1

Times represent function executions only; no call overhead is included. Instructions are on-chip (no cache misses). Times assume no memory contentions, no interrupt occurrences (no ISRs), no DMA activities, or host port action that would stall the CPU. PUSH and POP use on-chip SRAM for their stack (if needed).

MAC = MPY and ADD = 15.4 + 13.3 = 28.7/2 = 14.4 million/s. MPY – ADD average at 200 MHz.

SP FP Addition = _addf [ADDSP on C67xx]

The sum of two-input 32-bit FP numbers is generated. Underflow returns zero, overflow returns + or – infinity.

`_addf` returns the SP FP sum: ans = arg1 + arg2 (all SP FP).

SP FP Division = _divf

The quotient of two-input 32-bit FP numbers is generated. Underflow returns zero. Overflow returns + or – infinity. Zero over zero returns zero. Non-zero over zero returns infinity.

`_divf` returns the SP FP quotient: ans = arg1/arg2 (all SP FP).

Convert SP FP to 32-Bit Signed Integer = _fixfi [SPINT on C67xx]

An input 32-bit FP number is converted to a 32-bit signed integer. Numbers with magnitude of less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values:

- `0x7fff_ffff` (for positive numbers)
- `0x8000_0000` (for negative numbers)

`_fixfi` returns a 32-bit signed integer: ans = SP FP arg1.

Convert SP FP to 40-Bit Signed Long Integer = _fixfli

An input 32-bit FP number is converted to a 40-bit signed long integer. Numbers with magnitude of less than 1.0 return zero. Numbers greater than 40 bits return one of the following saturation values:

- `0x7f_ffff_ffff` (for positive numbers)
- `0x80_0000_0000` (for negative numbers)

`_fixfli` returns a 40-bit signed long integer: ans = SP FP arg1.

Convert SP FP to 32-Bit Unsigned Integer = _fixfu

An input 32-bit FP number is converted to a 32-bit unsigned integer. Numbers less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values:

- `0xffff_ffff` (for positive numbers)
- `0x0000_0000` (for negative numbers or numbers < 1.0)

`_fixfu` returns a 32-bit unsigned integer: ans = SP FP arg1.

Convert SP FP to 40-Bit Unsigned Long Integer = _fixful

An input 32-bit FP number is converted to a 40-bit unsigned long integer. Numbers less than 1.0 return zero. Numbers greater than 40 bits return one of the following saturation values:

- `0xff_ffff_ffff` (for positive numbers)
- `0x00_0000_0000` (for negative numbers or numbers < 1.0)

`_fixful` returns a 40-bit unsigned long integer: ans = SP FP arg1.

Convert 32-Bit Signed Integer to SP FP = _fltif [INTSP on C67xx]

An input 32-bit signed integer is converted to a 32-bit SP FP number.

`_fltif` returns a 32-bit SP FP number: ans = INT arg1.

Convert 40-Bit Signed Long Integer to SP FP = _fltlif

An input 40-bit signed long integer is converted to a 32-bit SP FP number.

`_fltlif` returns a 32-bit SP FP number: ans = LONG arg1.

Convert 32-Bit Unsigned Integer to SP FP = _fltuf [INTSPU on C67xx]

An input 32-bit unsigned integer is converted to a 32-bit SP FP number.

`_fltuf` returns a 32-bit SP FP number: ans = Unsigned INT arg1.

Convert 40-Bit Unsigned Long Integer to SP FP = _fltulf

An input 40-bit unsigned long integer is converted to a 32-bit SP FP number.

_fltulf returns a 32-bit SP FP number: ans = Unsigned LONG arg1.

SP FP Multiplication = _mpyf [MPYSP on C67xx]

The product of two input 32-bit FP numbers is generated. Underflow or exponents < 2 (2^{-125} = 2.35 x 10^{-38}) returns zero. Overflow returns + or – infinity.

`_mpyf` returns the SP FP product: ans = arg1 * arg2 (all SP FP).

SP FP Subtraction = _subf [SUBSP on C67xx]

The difference of two input 32-bit FP numbers is generated. Underflow returns zero. Overflow returns + or – infinity.

`_subf` returns the SP FP difference: ans = arg1 – arg2 (all SP FP).

Assembly (ASM) Listings

This section contains the hand-coded ASM listings and register allocations used for each function.

NOTE:
Equivalent C67xx hardware instructions are listed in [square brackets].

- `_addf` = SP FP addition [ADDSP on C67xx]
- `_divf` = SP FP division

  [C67xx RCPSP performs an 8-bit table look-up of the reciprocal of the divisor. Two software Newton-Rhapson iterations are performed to obtain the SP FP quotient.]
- `_fixfi` = convert SP FP to 32-bit signed integer [SPINT on C67xx]
- `_fixfli` = convert SP FP to 40-bit signed long integer

- `_fixfu` = convert SP FP to 32-bit unsigned integer
- `_fixful` = convert SP FP to 40-bit unsigned long integer
- `_fltif` = convert 32-bit signed integer to SP FP [`INTSP` on C67xx]
- `_fltlif` = convert 40-bit signed long integer to SP FP
- `_fltuf` = convert 32-bit unsigned integer to SP FP [`INTSPU` on C67xx]
- `_fltulf` = convert 40-bit unsigned long integer to SP FP
- `_mpyf` = SP FP multiplication [`MPYSP` on C67xx]
- `_subf` = SP FP subtraction [`SUBSP` on C67xx]

NOTE:

All labels used in C are generated with an underscore preceding the symbol. For example, label “Name” in C becomes label “_Name” in assembly. Likewise, if the label already has an underscore, a second one will be added. For example, the label “_subf ” in C becomes “_ _subf ” in assembly.

SP FP Addition ASM Listing for _addf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPADD1.asm      Syd Poland    04-09-98        (for TMS320C62xx DSPs)
;                                a4 + b4 ---> a4
;       32-bit SP Floating Point Addition ---> 32-bit SP Floating Point
;                 or Subtraction  a4 - b4 ---> a4
;
;       using IEEE-754-1985 format

        .text                                ; program memory
        .global _SPADD,_SPSUB,__addf,__subf ; entry labels for ADD or SUB
        .align  32                           ; Fetch Packet boundary

arg1    .set    a4      ; IEEE-754 32-bit SP Floating Point input argument 1
arg2    .set    b4      ;     "      "    "     "       "     "       "    2
ans     .set    a4      ;     "      "    "     "       "   output answer
exp1    .set    a1      ; arg1 exp
exp2    .set    b1      ; arg2 exp
sign1   .set    a2      ; input sign1
sign2   .set    b2      ; input sign2
mant1   .set    a5      ; mant1
mant2   .set    b5      ; mant2
V       .set    b0      ; Overflow sw.
U       .set    a2      ; Underflow switch
N       .set    b6      ; exp. difference
T       .set    b1      ; shift amount
W       .set    a6      ; work reg.
Z       .set    b7      ; Zero
S       .set    a7      ; output sign
AZ      .set    b2      ; answer = 0 switch

;__subf:                                        ; entry in rts6201 library

;_SPSUB:                                        ; ans = arg1 - arg2 entry
;        zero    .D2     mant2                 ; zero mant2
;||      cmpeq   .L2     arg2,0,sign2          ; is arg2 = 0 ?
;
;        set     .S2     mant2,31,31,mant2     ; set sign bit = 1
;
;  [!sign2] xor  .L2     arg2,mant2,arg2       ; invert non-0 arg2's sign

__addf:                                  ; entry in rts6201 library

_SPADD:                                  ; ans = arg1 + arg2 entry
        extu    .S1     arg1,1,24,exp1  ; get arg1's exp.field
||      extu    .S2     arg2,1,24,exp2  ; get arg2's exp.field
||      cmplt   .L1     arg1,0,sign1    ; input sign1 < 0 ?
||      cmplt   .L2     arg2,0,sign2    ; input sign2 < 0 ?

  [exp1]  extu  .S1     arg1,9,9,mant1  ; get arg1's mant field if exp1 > 0
||[exp2]  extu  .S2     arg2,9,9,mant2  ; get arg2's mant field if exp2 > 0
||[!exp1] zero  .D1     mant1           ; zero mant1 field if exp1=0
||[!exp2] zero  .D2     mant2           ; zero mant2 field if exp2=0
||        mpy   .M2     Z,0,Z           ; force a zero

  [exp1]  set   .S1     mant1,23,23,mant1       ; add hidden 1 if exp1 > 0
||[exp2]  set   .S2     mant2,23,23,mant2       ; add hidden 1 if exp2 > 0
||[!exp1] zero  .D1     sign1                   ; zero sign1 if exp1=0
||[!exp2] zero  .D2     sign2                   ; zero sign2 if exp2=0
||        sub   .L1x    exp1,exp2,W             ; dif. of 2 input exponents
||        cmplt .L2x    exp1,exp2,V             ; exp1 < exp2 ?

  [sign1] sub   .L1     0,mant1,mant1           ; set mant1 = -mant1 ?
||[sign2] sub   .D2     Z,mant2,mant2           ; set mant2 = -mant2 ?
||[V]     mv    .S1x    exp2,exp1               ; move bigger exp2 to exp1
||        abs   .L2     W,N                     ; get absolute N
```

```
||        mvk   .S2     25,T                    ; T = max. shift amount

        cmpltu  .L2     N,T,T                   ; is N < T (25) ?
||[V]   mv      .S2x    mant1,mant2             ; swap mant1 and mant2
||[V]   mv      .S1x    mant2,mant1             ; if exp2 > exp1

  [T]   shr     .S2     mant2,N,mant2           ; shift mant2 right (N is OK)
||[!T]  zero    .D2     mant2                   ; zero mant2 (N is too big)

        add     .L1x    mant1,mant2,ans         ; add 2 mantissas

        extu    .S1     ans,0,31,sign1          ; output sign
||      abs     .L1     ans,ans                 ; get |mantissa|
||      addab   .D1     exp1,7,exp1             ; exp1 adjust pre-norm
||      mvk     .S2     254,mant2               ; max. exp
||      cmpeq   .L2x    ans,0,AZ                ; is ans = 0 ?

        norm    .L1     ans,W                   ; normalize count
||      shl     .S1     sign1,31,S              ; shift to sign field

        shl     .S1     ans,W,ans               ; normalize mantissa
||      subab   .D1     exp1,W,exp1             ; output exp w/norm N
||      b       .S2     b3                      ; normal return

        shru    .S1     ans,7,ans               ; shift to mantissa field
||      cmplt   .L1     exp1,1,U                ; exp < 1 (underflow) ?
||      cmpgt   .L2x    exp1,mant2,V            ; exp >254 (overflow) ?

        clr     .S1     ans,23,31,ans           ; keep only mant.field
||[AZ]  add     .L1x    AZ,0,U                  ; if ans=0, force U=1

        shl     .S1     exp1,23,exp1            ; shift to exp.field

  [!V]  add     .L1     exp1,ans,ans            ; combine exp/mant fields
||[V]   set     .S1     ans,0,30,ans            ; set max exp/mant=all 1s

  [!U]  add     .L1     ans,S,ans               ; include sign bit in answer
||[U]   zero    .D1     ans                     ; set to 0 on underflow

        .end
```

NOTE:

This listing has the SP FP subtract `_ _subf` instructions commented out. However, if space is primary and speed is secondary, remove the comments and reassemble; do not link in the separate subtract subroutine. This assumes addition dominates subtract.

SP FP Addition ASM Registers for _addf

A0		V	B0
A1	exp1	exp2/T	B1
A2	Sign1/U	sign2/AZ	B2
A3		return address	B3
A4	arg1/ans	arg2	B4
A5	mant1	mant2	B5
A6	W	N	B6
A7	S	Z	B7

SP FP Division ASM Listing for _divf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPDIV1.asm      Syd Poland    05-05-98        (for TMS320C62xx DSPs)
;                                  a4 / b4 ---> a4
;       32-bit SP Floating Point Division ---> 32-bit SP Floating Point
;
;       using IEEE-754-1985 format

        .text                                   ; program memory
        .global _SPDIV,__divf                   ; entry labels
        .align  32                              ; Fetch Packet boundary

arg1    .set    a4      ; IEEE-754 32-bit SP Floating Point input argument 1
arg2    .set    b4      ;     "      "    "     "       "     "       "    2
ans     .set    a4      ;     "      "    "     "       "   output answer
exp1    .set    a1      ; arg1 exp
exp2    .set    b1      ; arg2 exp
sign    .set    a0      ; output sign
mant1   .set    a5      ; mant1
mant2   .set    b5      ; mant2
V       .set    b0      ; Overflow sw.
U       .set    a2      ; Underflow switch
N       .set    b2      ; loop count
W       .set    a6      ; shift amount
Q       .set    a3      ; partial quotient

__divf:                                 ; entry in rts6201 library

_SPDIV:                                 ; ans = arg1 / arg2 entry
        extu    .S1     arg1,1,24,exp1  ; get arg1's exp.field
||      extu    .S2     arg2,1,24,exp2  ; get arg2's exp.field
||      xor     .L1     arg1,arg2,sign  ; output sign

  [exp1]  extu  .S1     arg1,9,2,mant1  ; get arg1's mant field if exp1 > 0
||[exp2]  extu  .S2     arg2,9,2,mant2  ; get arg2's mant field if exp2 > 0
||[!exp1] zero  .D1     mant1           ; zero mant1 field if exp1=0
||[!exp2] zero  .D2     mant2           ; zero mant2 field if exp2=0
||[!exp2] sub   .L1     sign,sign,sign  ; zero sign if exp2=0

  [exp1]  set   .S1     mant1,30,30,mant1       ; add hidden 1 if exp1 > 0
||        set   .S2     mant2,30,30,mant2       ; add hidden 1 always
||[!exp1] zero  .D1     sign                    ; zero sign if exp1=0
||        mpy   .M1     Q,0,Q                   ; zero Q register
||        sub   .L1x    exp1,exp2,exp1          ; dif. of 2 input exponents

        ; 0.5 < quotient < 2.0, hence the 25 iterations

        clr     .S1     sign,0,30,sign          ; sign field
||      mvk     .S2     4,N                     ; cnt N = 4..3..2..1..0

loop:
        mv      .D1     mant1,W                 ; copy Q bits into tmp. W
||      clr     .S1     mant1,0,4,mant1         ; clear low bits
||[N]   b       .S2     loop                    ; branch ?

        subc    .L1x    mant1,mant2,mant1       ; subc #  1..6.11.16.21
||      extu    .S1     W,27,27,W               ; low-bits of quotients

        subc    .L1x    mant1,mant2,mant1       ; subc #  2..7.12.17.22
||      add     .D1     Q,W,Q                   ; next partial quotient
||[!N]  addk    .S1     133,exp1                ; 127 bias + Q=24 (last time)

        subc    .L1x    mant1,mant2,mant1       ; subc #  3..8.13.18.23
||      shl     .S1     Q,5,Q                   ; make room for next bits
```

```
        subc    .L1x    mant1,mant2,mant1       ; subc #  4..9.14.19.24
|||[N]  sub     .L2     N,1,N                   ; N=N-1 ?

        subc    .L1x    mant1,mant2,mant1       ; subc #  5.10.15.20.25
||      mvk     .S2     254,V                   ; max. exp

        norm    .L1     Q,W                     ; shift cnt
||      extu    .S1     mant1,27,27,U           ; get last quotient bits

        add     .D1     Q,U,Q                   ; final quotient bits
||      sub     .L1     exp1,W,exp1             ; adjusted exp.

        shl     .S1     Q,W,Q                   ; normalize mant
||      cmplt   .L1     exp1,1,U                ; exp < 1 (underflow) ?
||      cmpgt   .L2x    exp1,V,V                ; exp > 254 (overflow)?
||      b       .S2     b3                      ; normal return

          shr   .S1     Q,7,ans                 ; shift to mant.field
|||[!exp2] add   .L2     exp2,1,V                ; if exp2=0, force V=True
|||[!U]   cmpeq .L1     Q,0,U                   ; set U=T if Q=0 when U=False

          clr   .S1     ans,23,31,ans           ; keep only mant.field
|||[!exp2] sub   .L1     sign,sign,sign          ; force sign = 0 ?

          shl   .S1     exp1,23,exp1            ; shift to exp.field

  [!V]  add     .L1     exp1,ans,ans            ; combine exp/mant fields
|||[V]   set     .S1     ans,0,30,ans            ; exp/mant=all 1s (overflow)

  [!U]  add     .L1     ans,sign,ans            ; include sign bit in answer
|||[U]   zero    .D1     ans                     ; set to 0 on underflow

        .end
```

SP FP Division ASM Registers for _divf

A0	sign	V	B0
A1	exp1	exp2	B1
A2	U	N	B2
A3	Q	return address	B3
A4	arg1/ans	arg2	B4
A5	Mant1	mant2	B5
A6	W		B6

Convert SP FP to 32-Bit Signed Integer ASM Listing for _fixfi

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPINT1.asm       Syd Poland    03-23-98      (for TMS320C62xx DSPs)
;                                            a4 ---> a4
;       IEEE-754 short 32-bit SP Floating Point ---> 32-bit Signed Integer

        .text                           ; program section
        .global _SPINT,__fixfi          ; entry labels
        .align  32                      ; start on fetch packet boundary

arg1    .set    a4              ; input IEEE-754 32-bit SP floating-point #
sign    .set    a2              ; input sign bit
exp     .set    a1              ; input exponent
Base    .set    b4              ; base exponent
N       .set    b0              ; test switch
M       .set    a0              ; net shift amount
ans     .set    a4              ; output 32-bit Signed INT (q=0)
big     .set    b1              ; used for OVF values

__fixfi:                                        ; entry in rts6201 library

_SPINT:    extu    .S1     arg1,1,24,exp        ; exp. of input number
||         mvk     .S2     127,Base             ; exp of |numbers| < 1.0
||         zero    .D2     big                  ; set big=0

           shru    .S1     arg1,31,sign         ; input sign bit
||         cmpltu  .L2X    exp,Base,N           ; is input |#| < 1.0
||         b       .S2     b3                   ; normal return

   [!N]    extu    .S1     arg1,9,2,ans         ; get mant. field (|#| => 1.0)
||         addk    .S2     30,Base              ; max. base exp. = 127 + 30
|| [N]     zero    .L1     arg1                 ; force a 0 if |arg1| < 1.0
|| [N]     sub     .D1     exp,exp,exp          ; force exp=0 when |#| < 1.0
|| [N]     mpy     .M1     sign,0,sign          ; set sign=0 when |#| < 1.0

   [exp]   set     .S1     ans,30,30,ans        ; add hidden 1 if exp is non-0
||         sub     .L1X    Base,exp,M           ; net shift amount
||         cmpltu  .L2X    Base,exp,N           ; is Base < exp ?
||         set     .S2     big,31,31,big        ; set big=0x8000 0000 Neg.MAX

   [!N]    shru    .S1     ans,M,ans            ; no, shift answer right (q=0)
|| [N]     not     .L1x    big,ans              ; yes, set to +OVF=0x7fff ffff
|| [N]     mpy     .M2x    sign,N,N             ; -sign & OVF ?

  [sign]   neg     .L1     ans,ans              ; return negative INT (q=0)

   [N]     mv      .L1     big,ans              ; or -OVF = 0x8000 0000.

           .end
```

Convert SP FP to 32-Bit Signed Integer ASM Registers for _fixfi

A0	M	N	B0
A1	exp	big	B1
A2	sign		B2
A3		return address	B3
A4	arg1/ans	Base	B4

Convert SP FP to 40-Bit Signed Long Integer ASM Listing for _fixfli

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPLONG1.asm       Syd Poland    03-23-98      (for TMS320C62xx DSPs)
;                                            a4 ---> a5:a4
;       IEEE-754 short 32-bit SP Floating Point ---> 40-bit Signed Integer

        .text                           ; program section
        .global _SPLONG,__fixfli        ; entry labels
        .align  32                      ; start on fetch packet boundary

arg1    .set    a4              ; input IEEE-754 32-bit SP floating-point #
sign    .set    a2              ; input sign bit
exp     .set    a1              ; input exponent
Base    .set    b4              ; base exponent
N       .set    b0              ; test switch
M       .set    a0              ; net shift amount
ansH    .set    a5              ; output 8-bits of 40-bit Signed LONG INT
ansL    .set    a4              ; output 32-bit of 40-bit Signed LONG INT q=0

__fixfli:                                       ; entry in rts6201 library

_SPLONG:   extu    .S1     arg1,1,24,exp        ; exp. of input number
||         mvk     .S2     127,Base             ; exp of |numbers| < 1.0

           shru    .S1     arg1,31,sign         ; input sign bit
||         cmpltu  .L2X    exp,Base,N           ; is input |#| < 1.0
||         addah   .D2     Base,19,Base         ; max. base exp = 127 + 38

   [!N]    extu    .S1     arg1,9,2,arg1        ; get mant. field (|#| => 1.0)
|| [N]     zero    .L1     arg1                 ; force a 0 if |arg1| < 1.0
|| [N]     sub     .D1     exp,exp,exp          ; force exp=0 when |#| < 1.0
|| [N]     mpy     .M1     sign,0,sign          ; set sign=0 when |#| < 1.0
||         b       .S2     b3                   ; normal return

   [exp]   set     .S1     arg1,30,30,arg1      ; add hidden 1 if exp is non-0
||         sub     .L1X    Base,exp,M           ; net shift amount
||         cmpltu  .L2X    Base,exp,N           ; is Base < exp ?

           shl     .S1     arg1,8,ansH:ansL     ; left justify mantissa

   [!N]    shru    .S1     ansH:ansL,M,ansH:ansL ; no, shift answer right (q=0)
|| [N]     or      .L1     -1,ansL,ansL         ; yes, set to +OVF=0xffff ffff
|| [N]     mpy     .M2x    sign,N,N             ; -sign & OVF ?
|| [N]     sub     .D1     sign,sign,sign       ; zero sign to appear +

   [sign]  neg     .L1     ansH:ansL,ansH:ansL  ; return negative INT (q=0)
|| [N]     mvk     .S1     0x7f,ansH            ; yes, set to +OVF=0x7f as 8-bit

  [N]      add     .L1     1,ansH:ansL,ansH:ansL ; or -OVF = 0x80 0000 0000.

           .end
```

Convert SP FP to 40-Bit Signed Long Integer ASM Registers for _fixfli

A0	M	N	B0
A1	exp		B1
A2	sign		B2
A3		return address	B3
A4	arg1/ansL	Base	B4

Convert SP FP to 32-Bit Unsigned Integer ASM Listing for _fixfu

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPUINT1.asm       Syd Poland    03-10-98      (for TMS320C62xx DSPs)
;                         a4 ---> a4
;       IEEE-754 short 32-bit SP Floating Point ---> 32-bit Unsigned Integer

        .text                           ; program section
        .global _SPUINT,__fixfu         ; entry labels
        .align  32                      ; start on fetch packet boundary

arg1    .set    a4              ; input IEEE-754 32-bit SP floating-point #
sign    .set    a2              ; input sign bit
exp     .set    a1              ; input exponent
Base    .set    b4              ; base exponent
N       .set    b0              ; test switch
M       .set    a0              ; net shift amount
ans     .set    a4              ; output 32-bit Unsigned INT (q=0)

__fixfu:                                        ; entry in rts6201 library

_SPUINT:   shru    .S1     arg1,31,sign         ; input sign bit
||         zero    .L1     exp                  ; initialize exp=0
||         mvk     .S2     127,Base             ; exp of numbers < 1.0

   [sign]  zero    .L1     arg1                 ; force to zero if arg1 < 0
||[!sign]  extu    .S1     arg1,1,24,exp        ; exp. of input number > 0
||         b       .S2     b3                   ; normal return

           cmpltu  .L2X    exp,Base,N           ; is input # < 1.0
||         addk    .S2     31,Base              ; max. base exp. = 127 + 31

   [N]     zero    .L1     arg1                 ; force a 0 if arg1 < 1.0
|| [N]     sub     .D1     exp,exp,exp          ; force exp=0 when # < 1.0
|| [!N]    extu    .S1     arg1,9,1,ans         ; get mant. field (# => 1.0)

   [exp]   set     .S1     ans,31,31,ans        ; add hidden 1 if exp is non-0
||         sub     .L1X    Base,exp,M           ; net shift amount
||         cmpltu  .L2X    Base,exp,N           ; is Base < exp ?

   [!N]    shru    .S1     ans,M,ans            ; no, shift answer right (q=0)

   [N]     set     .S1     ans,0,31,ans         ; yes, set to all 1s (OVF)

           .end
```

Convert SP FP to 32-Bit Unsigned Integer ASM Registers for _fixfu

A0	M	N	B0
A1	exp		B1
A2	sign		B2
A3		return address	B3
A4	arg1/ans	Base	B4

Convert SP FP to 40-Bit Unsigned Long Integer ASM Listing for _fixful

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPULONG1.asm       Syd Poland    03-11-98      (for TMS320C62xx DSPs)
;                                            a4 ---> a5:a4
;       IEEE-754 short 32-bit SP Floating Point ---> 40-bit Unsigned Integer

        .text                           ; program section
        .global _SPULONG,__fixful       ; entry labels
        .align  32                      ; start on fetch packet boundary

arg1    .set    a4              ; input IEEE-754 32-bit SP floating-point #
sign    .set    a2              ; input sign bit
exp     .set    a1              ; input exponent
Base    .set    b4              ; base exponent
N       .set    b0              ; test switch
M       .set    a0              ; net shift amount
ansL    .set    a4              ; output LS 32-bit Unsigned INT (q=0)
ansH    .set    a5              ; output MS 8-bit Unsigned INT
ones    .set    a6              ; reg=32-bits of 1s
VIII    .set    b4              ; reg=8-bits of 1s

__fixful:                       ; entry in rts6201 library

_SPULONG:       ; IEEE-754 32-bit SP Floating Point ---> 40-bit Unsigned INT
           shru    .S1     arg1,31,sign         ; input sign bit
||         zero    .L1     exp                  ; initialize exp=0
||         mvk     .S2     127,Base             ; exp of numbers < 1.0

   [sign]  zero    .L1     arg1                 ; force to zero if arg1 < 0
||[!sign]  extu    .S1     arg1,1,24,exp        ; exp. of input number > 0
||         b       .S2     b3                   ; normal return

           cmpltu  .L2X    exp,Base,N           ; is input # < 1.0
||         addk    .S2     39,Base              ; max. base exp. = 127 + 39
||         mvk     .S1     -1,ones              ; generate reg of all 1s

   [N]     zero    .L1     arg1                 ; force a 0 if arg1 < 1.0
|| [N]     sub     .D1     exp,exp,exp          ; force exp=0 when # < 1.0
|| [!N]    extu    .S1     arg1,9,1,ansL        ; get mant. field (# => 1.0)

   [exp]   set     .S1     ansL,31,31,ansL      ; add hidden 1 if exp is non-0
||         sub     .L1X    Base,exp,M           ; net shift amount
||         cmpltu  .L2X    Base,exp,N           ; is Base < exp ?

           shl     .S1     ansH:ansL,8,ansH:ansL ; left justify #
||         mvk     .S2     255,VIII              ; 8 MS-bits of 1s in 40-bits

   [!N]    shru    .S1     ansH:ansL,M,ansH:ansL; no, shift answer right (q=0)
|| [N]     add     .D1     ones,0,ansL          ; yes, set to all 1s (OVF)
|| [N]     add     .L1X    VIII,0,ansH          ; yes,  "   " 8-bits=1s

           .end
```

Convert SP FP to 40-Bit Unsigned Long Integer ASM Registers for _fixful

A0	M	N	B0
A1	exp		B1
A2	sign		B2
A3		return address	B3
A4	arg1/ansL	Base/VIII	B4
A5	ansH		B5
A6	ones		B6

Convert 32-Bit Signed Integer to SP FP ASM Listing for _fltif

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       INTSP1.asm       Syd Poland    03-07-98       (for TMS320C62xx DSPs)
;                          a4 ---> a4
;       32-bit Signed Integer ---> 32-bit IEEE-754 SP Floating Point

        .text                               ; program section
        .global _INTSP,__fltif              ; subroutine entry labels
        .align  32                          ; start on fetch packet boundary

ans     .set    a4      ; output 32-bit Single-Precision Floating-Point answer
arg1    .set    a4      ; input 32-bit signed Integer (q=0)
sign    .set    a1      ; sign bit field
mant    .set    a2      ; IEEE-754 SP FP mantissa field
exp     .set    a0      ; IEEE-754 SP FP exponent field
N       .set    a5      ; normalize counter
Base    .set    a0      ; base exponent for 32-bit signed Integer (q=0)

__fltif:                ; entry in rts6201 library

_INTSP:                 ; Signed 32-bit Integer ---> 32-bit Floating Point
        abs   .L1  arg1,mant                ; absolute value of mantissa
 ||     clr   .S1  arg1,0,30,sign           ; get input sign bit
 ||     b     .S2  b3                       ; normal subroutine return

        lmbd  .L1  1,mant,N                 ; # of leading sign bits
 ||     mvk   .S1  127+31,Base              ; base exponent for INTEGER

        shl   .S1  mant,N,ans               ; normalize absolute mantissa
 ||     sub   .D1  Base,N,exp               ; output exponent for non-zero mant

        shl   .S1  exp,23,exp               ; shift value to exponent field

        extu  .S1  ans,1,9,ans              ; shift to mantissa field
 ||     add   .L1  sign,exp,exp             ; combine sign and exponent fields

  [!mant] zero.D1  ans                      ; return zero answer if arg1 = 0
 ||[mant] add .S1  exp,ans,ans              ; insert sign and exponent fields

          .end
```

Convert 32-Bit Signed Integer to SP FP ASM Registers for _fltif

A0	exp/Base		B0
A1	Sign		B1
A2	Mant		B2
A3		return address	B3
A4	arg1/ans		B4
A5	N		B5

Convert 40-Bit Signed Integer to SP FP ASM Listing for _fltlif

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       LONGSP1.asm       Syd Poland    03-07-98      (for TMS320C62xx DSPs)
;                       a5:a4 ---> a4
;  40-bit Signed LONG integer ---> IEEE-754 32-bit SP Floating-Point

         .global _LONGSP,__fltlif ; entry labels
         .align  32               ; start on fetch packet boundary
         .text                    ; program section

hi1     .set    a5              ; input MS signed LONG INT reg. (q=0)
low1    .set    a4              ; input LS signed LONG INT reg. (q=0)
sign    .set    a1              ; input sign bit
Base    .set    b0              ; signed LONG INT base exponent (q=0)
N       .set    a0              ; normalize amount
exp     .set    a0              ; IEEE-754 SP FP exponent field
mant    .set    a2              ; IEEE-754 SP FP mantissa field
ans     .set    a4              ; output IEEE-754 SP FP answer

__fltlif:                                       ; entry in rts6201 library

_LONGSP:
          extu  .S1   hi1,24,31,sign            ; copy sign bit
||        abs   .L1   hi1:low1,hi1:low1         ; get absolute value of arg1
||        mvk   .S2   127+38,Base               ; base exponent for LONG INT

          b     .S2   b3                        ; normal return
||        shl   .S1   sign,31,sign              ; move sign to sign field
||        norm  .L1   hi1:low1, N               ; # of leading sign bits

          sub   .L1x  Base,N,exp                ; output exp for non-0 mant
||        shl   .S1   hi1:low1,N,hi1:low1       ; normalize mantissa

          shru  .S1   hi1:low1,15,hi1:low1      ; shift to mantissa field

          shl   .S1   exp,23,exp                ; shift value to exp field
||        abs   .L1   low1,mant                 ; absolute value of mantissa

          add   .L1   sign,exp,exp              ; combine sign and exp fields
||        clr   .S1   mant,23,31,ans            ; clear sign and exp fields

  [!mant] zero  .S1   ans                       ; return zero if mantissa = 0
|| [mant] add   .L1   exp,ans,ans              ; insert sign and exp fields
```

Convert 40-Bit Signed Integer to SP FP ASM Registers for _fltlif

A0	Exp/N	Base	B0
A1	Sign		B1
A2	Mant		B2
A3		return address	B3
A4	low1/ans		B4
A5	hi1		B5

Convert 32-Bit Unsigned Integer to SP FP ASM Listing for _fltuf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       UINTSP1.asm       Syd Poland    03-07-98      (for TMS320C62xx DSPs)
;                          arg1 ---> ans
;       Unsigned 32-bit Integer ---> IEEE-754 32-bit SP Floating-Point

        .global _UINTSP,__fltuf      ; subroutine entry labels
        .text                        ; program section
        .align  32                   ; start on fetch packet boundary

arg1    .set    a4                   ; input unsigned 32-bit INT (q=0)
N       .set    a0                   ; # bits to leading 1
ans     .set    a4                   ; IEEE-754 output SP FP answer
Base    .set    b4                   ; base exponent for UINT (q=0)
mant    .set    a1                   ; IEEE-754 SP FP mantissa field
exp     .set    b4                   ; IEEE-754 SP FP exponent field

__fltuf:                             ; entry in rts6201 library

_UINTSP:b       .S2   b3             ; subroutine return

        lmbd    .L1   1,arg1,N       ; find leading bit = 1

        shl     .S1   arg1,N,ans     ; normalize mantissa
 ||     mvk     .S2   127+31,Base    ; base exponent for Unsigned INTEGER

        shru    .S1   ans,8,mant     ; shift to mantissa field
 ||     sub     .L2X  Base,N,exp     ; output exponent for non-zero mant

        clr     .S1   mant,23,31,ans ; clear sign and exponent fields
 ||     shl     .S2   exp,23,exp     ; shift value to exponent field

   [mant]  add  .L1X  exp,ans,ans    ; insert +sign and exponent fields
 ||[!mant] zero .S1   ans            ; return zero answer if arg1 = 0

           .end
```

Convert 32-Bit Unsigned Integer to SP FP ASM Registers for _fltuf

A0	N		B0
A1	mant		B1
A2			B2
A3		return address	B3
A4	arg1/ans	Base/exp	B4

Convert 40-Bit Unsigned Integer to SP FP ASM Listing for _fltulf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       ULONGSP1.asm       Syd Poland    02-27-98     (for TMS320C62xx DSPs)
;                        a5:a4 ---> a4
;  40-bit Unsigned LONG integer ---> 32-bit IEEE-754 SP Floating-Point

        .global _ULONGSP,__fltulf       ; entry labels
        .align  32                      ; start on fetch packet boundary
        .text                           ; program section

hi1     .set    a5      ; MS 8-bits of Unsigned LONG INT
lo1     .set    a4      ; LS 32-bits of Unsigned LONG INT (q=0)
hi2     .set    a7      ; MS 8-bits of TEMP long register
lo2     .set    a6      ; LS 32-bits of TEMP long register
N       .set    a3      ; normalize amount
Base    .set    b4      ; base exponent for Unsigned LONG INT (q=0)
exp     .set    b4      ; IEEE-754 exponent field
mant    .set    a1      ; IEEE-754 mantissa field
ans     .set    a4      ; IEEE-754 SP FP answer

__fltulf:                                       ; entry in rts6201 library

_ULONGSP:  shru    .S1     hi1:lo1,1,hi2:lo2    ; force + sign bit
||         b       .S2     B3                   ; normal return

           norm    .L1     hi2:lo2,N            ; find 1st one

           shl     .S1     hi1:lo1,N,hi1:lo1    ; normalize mantissa
||         mvk     .S2     127+39,Base          ; base exp. for Unsigned LONG

           shru    .S1     hi1:lo1,16,hi1:lo1   ; shift to mantissa field
||         sub     .L2X    Base,N,exp           ; output exp for non-0 mant

           clr     .S1     lo1,23,31,lo1        ; clear sign and exp fields
||         mv      .L1     lo1,mant             ; save mantissa for test
||         shl     .S2     exp,23,exp           ; shift value to exp field

  [mant]   add     .L1X    exp,lo1,ans          ; insert +sign and exp fields
|||[!mant]  zero    .S1     ans                  ; return 0 ans if mant = 0

           .end
```

Convert 40-Bit Unsigned Integer to SP FP ASM Registers for _fltulf

A0			B0
A1	mant		B1
A2			B2
A3	N	return address	B3
A4	lo1/ans	Base/exp	B4
A5	hi1		B5
A6	lo2		B6
A7	hi2		B7

SP FP Multiplication ASM Listing for _mpyf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPMPY1.asm      Syd Poland    03-25-98        (for TMS320C62xx DSPs)
;                                a4 * b4 ---> a4
;       32-bit SP Floating Point Multiply ---> 32-bit SP Floating Point
;       using IEEE-754-1985 format

        .text                   ; program memory
        .global _SPMPY,__mpyf   ; entry labels
        .align  32              ; Fetch Packet boundary

arg1    .set    a4      ; IEEE-754 32-bit SP Floating Point input argument 1
arg2    .set    b4      ;     "      "    "     "       "     "       "    2
ans     .set    a4      ;     "      "    "     "       "   output answer
exp1    .set    a1      ; arg1 exp
exp2    .set    b1      ; arg2 exp
sign    .set    b0      ; output sign
mant1   .set    a5      ; mant1
mant2   .set    b7      ; mant2
Base    .set    a6      ; base exp of 1.0 = 127
U       .set    a2      ; Underflow switch
HH      .set    a7      ; H*H
HL      .set    a8      ; H*L
LH      .set    b7      ; L*H
LL      .set    b8      ; L*L
Big     .set    b9      ; Biggest exp. = 255
Hi      .set    b5      ; Hi register in 40-bits
Lo      .set    b4      ; Lo register in 40-bits
OVF     .set    b2      ; Overflow sw.
N       .set    a7      ; exp adjust of 1

__mpyf:                                 ; entry in rts6201 library

_SPMPY: extu    .S1     arg1,1,24,exp1  ; get arg1's exp.field
||      extu    .S2     arg2,1,24,exp2  ; get arg2's exp.field
||      xor     .L2x    arg1,arg2,sign  ; form output sign

  [exp1]  extu  .S1     arg1,9,9,mant1  ; get arg1's mant field if exp1 > 0
|||[exp2]  extu  .S2     arg2,9,9,mant2  ; get arg2's mant field if exp2 > 0
|||[!exp1] zero  .D1     mant1           ; zero mant1 field if exp1=0
|||[!exp2] zero  .D2     mant2           ; zero mant2 field if exp2=0
|||[!exp2] zero  .L2     sign,0,sign     ; zero output sign if exp2=0

  [exp1]  set   .S1     mant1,23,23,mant1       ; add hidden 1 if exp1 > 0
|||[exp2]  set   .S2     mant2,23,23,mant2       ; add hidden 1 if exp2 > 0
|||[!exp1] zero  .L2     sign                    ; zero output sign if exp1=0
||        add   .L1x    exp1,exp2,exp1          ; sum of 2 input exponents

        mvk     .S1     127,Base        ; exponent bias
||      mpyu    .M2x    mant1,mant2,LL  ; low * low     (u*u)
||      mpyhu   .M1x    mant1,mant2,HH  ; high * high   (u*u)
||      clr     .S2     sign,0,30,sign  ; keep only the output sign bit (31)

         mpyhlu .M1x    mant1,mant2,HL  ; high * low    (u*u)
||       mpyhlu .M2x    mant2,mant1,LH  ; low * high    (u*u)
||[exp1] sub    .D1     exp1,Base,exp1  ; remove extra bias of 127 if exp1>0
||       mvk    .S2     254,Big         ; Max. exp permitted

        shru    .S2     LL,16,LL        ; L*L >>u 16
||      shl     .S1     HH,16,HH        ; HH << 16 (orig.top halfword == 0)
||      cmpgt   .L2x    exp1,Big,OVF    ; check for overflow (exp1 > Big)
||      cmplt   .L1     exp1,1,U        ; check for underflow (exp1 < 1)

        addu    .L2x    LH,HL,Hi:Lo     ; Hi:Lo = H*L + L*H
||      add     .L1x    HH,LL,HH        ; HH = HH<<16 + LL >>u 16
|| [!U] extu    .S1     exp1,24,1,exp1  ; shift output 8-bit exp to EXP field
```

```
        addu    .L2x    HH,Hi:Lo,Hi:Lo  ; final sum of all 4 products
||      zero    .D1     mant1           ; force mant1 to all zeros
||      b       .S2     b3              ; normal return

        shru    .S2     Lo,31,exp2       ; get MS-bit of final sum
||      set     .S1     mant1,0,22,mant1 ; set mant1 = all 1's (OVF)
||      mv      .L1x    Lo,ans           ; move unshifted Lo to ans

   [!U]  extu   .S1     ans,2,9,ans     ; output 1 <= mant < 2 (w/o hidden 1)
|| [!U]  or     .L2x    sign,exp1,sign  ; combine sign and exp fields
||[exp2] shru   .S2     Lo,8,mant2      ; shift right 1 bit (keep hidden 1
                                        ; as an exp inc of 1) 2 <= mant < 4

  [exp2] mv     .L1x    mant2,ans       ; move mant2 to ans if exp2 = 1

  [OVF]  add    .L1     mant1,0,ans     ; set Max mant for OVF
||[OVF]  set    .S2     sign,23,30,sign ; set max. exp for OVF, keep sign bit

  [U]    zero   .D1     ans             ; return 0 on UNDERFLOW
||[!U]   add    .L1x    sign,ans,ans    ; include ouput sign if no UNDERFLOW

         .end
```

SP FP Multiplication ASM Registers for _mpyf

A0		sign	B0
A1	exp1	exp2	B1
A2	U	OVF	B2
A3		return address	B3
A4	arg1/ans	arg2/Lo	B4
A5	Mant1	Hi	B5
A6	Base		B6
A7	HH/N	mant2/LH	B7
A8	HL	LL	B8
A9		Big	B9

SP Floating Point Subtraction ASM Listing for _subf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPSUB1.asm      Syd Poland    04-29-98        (for TMS320C62xx DSPs)
;                                 a4 - b4 ---> a4
;       32-bit SP Floating Point Subtract ---> 32-bit SP Floating Point
;                     or Addition  a4 + b4 ---> a4
;
;       using IEEE-754-1985 format

        .text                                ; program memory
        .global _SPADD,_SPSUB,__addf,__subf ; entry labels for ADD or SUB
        .align  32                           ; Fetch Packet boundary

arg1    .set    a4      ; IEEE-754 32-bit SP Floating Point input argument 1
arg2    .set    b4      ;     "      "    "     "       "     "       "    2
ans     .set    a4      ;     "      "    "     "       "   output answer
exp1    .set    a1      ; arg1 exp
exp2    .set    b1      ; arg2 exp
sign1   .set    a2      ; input sign1
sign2   .set    b2      ; input sign2
mant1   .set    a5      ; mant1
mant2   .set    b5      ; mant2
V       .set    b0      ; Overflow sw.
U       .set    a2      ; Underflow switch
N       .set    b6      ; exp. difference
T       .set    b1      ; shift amount
W       .set    a6      ; work reg.
Z       .set    b7      ; Zero
S       .set    a7      ; output sign
AZ      .set    b2      ; answer = 0 switch

;__addf:                                        ; entry in rts6201 library

;_SPADD:                                        ; ans = arg1 + arg2 entry
;        zero    .D2     mant2                  ; zero mant2
;||      cmpeq   .L2     arg2,0,sign2           ; is arg2 = 0 ?
;
;        set     .S2     mant2,31,31,mant2      ; set sign bit = 1
;
;  [!sign2] xor  .L2     arg2,mant2,arg2        ; invert non-0 arg2's sign

__subf:                                 ; entry in rts6201 library

_SPSUB:                                 ; ans = arg1 - arg2 entry
        extu    .S1     arg1,1,24,exp1  ; get arg1's exp.field
||      extu    .S2     arg2,1,24,exp2  ; get arg2's exp.field
||      cmplt   .L1     arg1,0,sign1    ; input sign1 < 0 ?
||      cmplt   .L2     arg2,0,sign2    ; input sign2 < 0 ?

  [exp1]  extu  .S1     arg1,9,9,mant1  ; get arg1's mant field if exp1 > 0
||[exp2]  extu  .S2     arg2,9,9,mant2  ; get arg2's mant field if exp2 > 0
||[!exp1] zero  .D1     mant1           ; zero mant1 field if exp1=0
||[!exp2] zero  .D2     mant2           ; zero mant2 field if exp2=0
||        mpy   .M2     Z,0,Z                   ; force a zero

  [exp1]  set   .S1     mant1,23,23,mant1       ; add hidden 1 if exp1 > 0
||[exp2]  set   .S2     mant2,23,23,mant2       ; add hidden 1 if exp2 > 0
||[!exp1] zero  .D1     sign1                   ; zero sign1 if exp1=0
||[!exp2] zero  .D2     sign2                   ; zero sign2 if exp2=0
||        sub   .L1x    exp1,exp2,W             ; dif. of 2 input exponents
||        cmplt .L2x    exp1,exp2,V             ; exp1 < exp2 ?

  [sign1]  sub  .L1     0,mant1,mant1           ; set mant1 = -mant1 ?
||[!sign2] sub  .D2     Z,mant2,mant2           ; set mant2 = -mant2 ?
||[V]      mv   .S1x    exp2,exp1               ; move bigger exp2 to exp1
||         abs  .L2     W,N                     ; get absolute N
||         mvk  .S2     25,T                    ; T = max. shift amount

        cmpltu  .L2     N,T,T                   ; is N < T (25) ?
||[V]   mv      .S2x    mant1,mant2             ; swap mant1 and mant2
||[V]   mv      .S1x    mant2,mant1             ; if exp2 > exp1
```

```
  [T]   shr     .S2     mant2,N,mant2           ; shift mant2 right (N is OK)
||[!T]  zero    .D2     mant2                   ; zero mant2 (N is too big)


        add     .L1x    mant1,mant2,ans         ; add 2 mantissas

        extu    .S1     ans,0,31,sign1          ; output sign
||      abs     .L1     ans,ans                 ; get |mantissa|
||      addab   .D1     exp1,7,exp1             ; exp1 adjust pre-norm
||      mvk     .S2     254,mant2               ; max. exp
||      cmpeq   .L2x    ans,0,AZ                ; is ans = 0 ?

        norm    .L1     ans,W                   ; normalize count
||      shl     .S1     sign1,31,S              ; shift to sign field

        shl     .S1     ans,W,ans               ; normalize mantissa
||      subab   .D1     exp1,W,exp1             ; output exp w/norm N
||      b       .S2     b3                      ; normal return

        shru    .S1     ans,7,ans               ; shift to mantissa field
||      cmplt   .L1     exp1,1,U                ; exp < 1 (underflow) ?
||      cmpgt   .L2x    exp1,mant2,V            ; exp >254 (overflow) ?

        clr     .S1     ans,23,31,ans           ; keep only mant.field
||[AZ]  add     .L1x    AZ,0,U                  ; if ans=0, force U=1

        shl     .S1     exp1,23,exp1            ; shift to exp.field

  [!V]  add     .L1     exp1,ans,ans            ; combine exp/mant fields
||[V]   set     .S1     ans,0,30,ans            ; set max exp/mant=all 1s

  [!U]  add     .L1     ans,S,ans               ; include sign bit in answer
||[U]   zero    .D1     ans                     ; set to 0 on underflow

        .end
```

NOTE:

This listing has the SP FP addition `_ _addf` instructions commented out. However, if space is primary and speed is secondary, remove the comments and reassemble; do not link in the separate add subroutine. This assumes subtraction dominates addition.

SP FP Subtraction ASM Registers for _subf

A0		V	B0
A1	exp1	exp2/T	B1
A2	sign1/U	sign2/AZ	B2
A3		return address	B3
A4	arg1/ans	arg2	B4
A5	mant1	mant2	B5
A6	W	N	B6
A7	S	Z	B7

Linear Assembly (SA) Listings

This section contains the SA listings used for each function. These are paraphrased from the previous ASM code.

NOTE:
Equivalent C67xx hardware instructions are listed in [square brackets].

- `_addf` = SP FP addition [ADDSP on C67xx]
- `_divf` = SP FP division

  [C67xx RCPSP performs an 8-bit table look-up of the reciprocal of the divisor. Two software Newton-Rhapson iterations are performed to obtain the SP FP quotient.]
- `_fixfi` = convert SP FP to 32-bit signed integer [SPINT on C67xx]
- `_fixfli` = convert SP FP to 40-bit signed long integer
- `_fixfu` = convert SP FP to 32-bit unsigned integer
- `_fixful` = convert SP FP to 40-bit unsigned long integer
- `_fltif` = convert 32-bit signed integer to SP FP [INTSP on C67xx]
- `_fltlif` = convert 40-bit signed long integer to SP FP
- `_fltuf` = convert 32-bit unsigned integer to SP FP [INTSPU on C67xx]
- `_fltulf` = convert 40-bit unsigned long integer to SP FP
- `_mpyf` = SP FP multiplication [MPYSP on C67xx]
- `_subf` = SP FP subtraction [SUBSP on C67xx]

SP FP Addition SA Listing for _addf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPADD.sa      Syd Poland    04-28-98        (for TMS320C62xx DSPs)
;                                 a4 + b4 ---> a4
;       32-bit SP Floating Point Addition ---> 32-bit SP Floating Point
;                 or Subtraction  a4 - b4 ---> a4
;
;       using IEEE-754-1985 format

        .text                                   ; program memory

        .global _SPADD,_SPSUB,__addf,__subf ; entry labels for ADD or SUB

        .align  32                              ; Fetch Packet boundary

;__subf:                                    ; entry in rts6201 library

;_SPSUB:                                    ; ans = arg1 - arg2 entry

;        zero    .D2     b5              ; zero mant2
;||      cmpeq   .L2     b4,0,b2         ; is arg2 = 0 ?

;        set     .S2     b5,31,31,b5     ; set mant2 sign bit = 1

;  [!b2] xor     .L2     b4,b5,b4        ; invert non-0 arg2's sign bit

__addf:                                    ; entry in rts6201 library

_SPADD: .cproc  arg1, arg2                ; ans = arg1 + arg2 entry

        .reg    ans,exp1,exp2,sign1,sign2,mant1,mant2

        .reg    W,V,N,T,AZ,U,V,S,Z

        extu         arg1,1,24,exp1  ; get arg1's exp.field
        extu         arg2,1,24,exp2  ; get arg2's exp.field
        cmplt        arg1,0,sign1    ; input sign1 < 0 ?
        cmplt        arg2,0,sign2    ; input sign2 < 0 ?

  [exp1]  extu       arg1,9,9,mant1  ; get arg1's mant field if exp1 > 0
  [exp2]  extu       arg2,9,9,mant2  ; get arg2's mant field if exp2 > 0
  [!exp1] zero       mant1           ; zero mant1 field if exp1=0
  [!exp2] zero       mant2           ; zero mant2 field if exp2=0
          mpy        Z,0,Z                    ; force a zero

  [exp1]  set        mant1,23,23,mant1        ; add hidden 1 if exp1 > 0
  [exp2]  set        mant2,23,23,mant2        ; add hidden 1 if exp2 > 0
  [!exp1] zero       sign1                    ; zero sign1 if exp1=0
  [!exp2] zero       sign2                    ; zero sign2 if exp2=0
          sub        exp1,exp2,W              ; dif. of 2 input exponents
          cmplt      exp1,exp2,V              ; exp1 < exp2 ?

  [sign1] sub        0,mant1,mant1            ; set mant1 = -mant1 ?
  [sign2] sub        Z,mant2,mant2            ; set mant2 = -mant2 ?
  [V]     mv         exp2,exp1                ; move bigger exp2 to exp1
          abs        W,N                      ; get absolute N
          mvk        25,T                     ; T = max. shift amount

        cmpltu       N,T,T                    ; is N < T (25) ?
  [V]   mv           mant1,mant2              ; swap mant1 and mant2
  [V]   mv           mant2,mant1              ; if exp2 > exp1

  [T]   shr          mant2,N,mant2            ; shift mant2 right (N is OK)
  [!T]  zero         mant2                    ; zero mant2 (N is too big)

        add          mant1,mant2,ans          ; add 2 mantissas

        extu         ans,0,31,sign1           ; output sign
        abs          ans,ans                  ; get |mantissa|
        addab        exp1,7,exp1              ; exp1 adjust pre-norm
        mvk          254,mant2                ; max. exp
```

```
       cmpeq        ans,0,AZ                ; is ans = 0 ?

       norm         ans,W                   ; normalize count
       shl          sign1,31,S              ; shift to sign field



       shl          ans,W,ans               ; normalize mantissa
       subab        exp1,W,exp1             ; output exp w/norm N

       shru         ans,7,ans               ; shift to mantissa field
       cmplt        exp1,1,U                ; exp < 1 (underflow) ?
       cmpgt        exp1,mant2,V            ; exp >254 (overflow) ?

       clr          ans,23,31,ans           ; keep only mant.field
[AZ]   add          AZ,0,U                  ; if ans=0, force U=1

       shl          exp1,23,exp1            ; shift to exp.field

[!V]   add          exp1,ans,ans            ; combine exp/mant fields
[V]    set          ans,0,30,ans            ; set max exp/mant=all 1s

[!U]   add          ans,S,ans               ; include sign bit in answer
[U]    zero         ans                     ; set to 0 on underflow

       .return      ans

       .endproc
```

NOTE:

This listing has the SP FP subtract `_ _subf` instructions commented out. However, if space is primary and speed is secondary, remove the comments and reassemble; do not link in the separate subtract subroutine. This assumes addition dominates subtract.

SP FP Division SA Listing for _divf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPDIV.sa      Syd Poland    05-05-98        (for TMS320C62xx DSPs)
;                                 a4 / b4 ---> a4
;       32-bit SP Floating Point Division ---> 32-bit SP Floating Point
;
;       using IEEE-754-1985 format

        .text                                   ; program memory

        .global _SPDIV,__divf                   ; entry labels

        .align  32                              ; Fetch Packet boundary

__divf:                                         ; entry in rts6201 library

_SPDIV: .cproc  arg1,arg2                       ; ans = arg1 / arg2 entry

        .reg    exp1,exp2,sign,mant1,mant2,W,N,Q,U,V,ans

        extu         arg1,1,24,exp1  ; get arg1's exp.field
        extu         arg2,1,24,exp2  ; get arg2's exp.field
        xor          arg1,arg2,sign  ; output sign

  [exp1]  extu       arg1,9,2,mant1  ; get arg1's mant field if exp1 > 0
  [exp2]  extu       arg2,9,2,mant2  ; get arg2's mant field if exp2 > 0
  [!exp1] zero       mant1           ; zero mant1 field if exp1=0
  [!exp2] zero       mant2           ; zero mant2 field if exp2=0
  [!exp2] sub        sign,sign,sign  ; zero sign if exp2=0

  [exp1]  set        mant1,30,30,mant1       ; add hidden 1 if exp1 > 0
          set        mant2,30,30,mant2       ; add hidden 1 always
  [!exp1] zero       sign                    ; zero sign if exp1=0
          mpy        Q,0,Q                   ; zero Q register
          sub        exp1,exp2,exp1          ; dif. of 2 input exponents

        ; 0.5 < quotient < 2.0, hence the 25 iterations

        clr          sign,0,30,sign          ; sign field
        mvk          5,N                     ; cnt N = 4  3  2  1  0

loop:   .trip   5
        mv           mant1,W                 ; copy Q bits into tmp. W
        clr          mant1,0,4,mant1         ; clear low bits

        subc         mant1,mant2,mant1       ; subc #  1  6 11 16 21
        extu         W,27,27,W               ; low-bits of quotients

        subc         mant1,mant2,mant1       ; subc #  2  7 12 17 22
        add          Q,W,Q                   ; next partial quotient
  [!N]  addk         133,exp1                ; 127 bias + Q=24 (last time)

        subc         mant1,mant2,mant1       ; subc #  3  8 13 18 23
        shl          Q,5,Q                   ; make room for next bits

        subc         mant1,mant2,mant1       ; subc #  4  9 14 19 24
  [N]   sub          N,1,N                   ; N=N-1 ?

        subc        mant1,mant2,mant1        ; subc #  5 10 15 20 25
        mvk          254,V                   ; max. exp

  [N]   b            loop                    ; branch ?

        norm         Q,W                     ; shift cnt
        extu         mant1,27,27,U           ; get last quotient bits
```

```
       add         Q,U,Q                   ; final quotient bits
       sub         exp1,W,exp1             ; adjusted exp.

       shl         Q,W,Q                   ; normalize mant
       cmplt       exp1,1,U                ; exp < 1 (underflow) ?
       cmpgt       exp1,V,V                ; exp > 254 (overflow)?

         shr       Q,7,ans                 ; shift to mant.field
[!exp2] add        exp2,1,V                ; if exp2=0, force V=True
[!U]    cmpeq      Q,0,U                   ; set U=T if Q=0 when U=False

         clr       ans,23,31,ans           ; keep only mant.field
[!exp2] sub        sign,sign,sign          ; force sign = 0 ?

         shl       exp1,23,exp1            ; shift to exp.field

[!V]  add          exp1,ans,ans            ; combine exp/mant fields
[V]   set          ans,0,30,ans            ; exp/mant=all 1s (overflow)

[!U]  add          ans,sign,ans            ; include sign bit in answer
[U]   zero         ans                     ; set to 0 on underflow

       .return     ans

       .endproc
```

Convert SP FP to 32-Bit Signed Integer SA Listing for _fixfi

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPINT.sa       Syd Poland    03-23-98      (for TMS320C62xx DSPs)
;                                                  a4 ---> a4
;       IEEE-754 short 32-bit SP Floating Point ---> 32-bit Signed Integer

        .text                           ; program section
        .global _SPINT,__fixfi          ; entry labels
        .align  32                      ; start on fetch packet boundary

__fixfi:                                ; entry in rts6201 library

_SPINT: .cproc  arg1    ; 32-bit SP Floating Point ---> 32-bit Signed INT

        .reg    exp,Base,big,sign,N,M,ans

           extu         arg1,1,24,exp       ; exp. of input number
           mvk          127,Base            ; exp of |numbers| < 1.0
           zero         big                 ; set big=0

           shru         arg1,31,sign        ; input sign bit
           cmpltu       exp,Base,N          ; is input |#| < 1.0

   [!N]    extu         arg1,9,2,ans        ; get mant. field (|#| => 1.0)
           addk         30,Base             ; max. base exp. = 127 + 30
   [N]     zero         arg1                ; force a 0 if |arg1| < 1.0
   [N]     sub          exp,exp,exp         ; force exp=0 when |#| < 1.0
   [N]     mpy          sign,0,sign         ; set sign=0 when |#| < 1.0

   [exp]   set          ans,30,30,ans       ; add hidden 1 if exp is non-0
           sub          Base,exp,M          ; net shift amount
           cmpltu       Base,exp,N          ; is Base < exp ?
           set          big,31,31,big       ; set big=0x8000 0000 Neg.MAX

   [!N]    shru         ans,M,ans           ; no, shift answer right (q=0)
   [N]     not          big,ans             ; yes, set to +OVF=07fff ffff
   [N]     mpy          sign,N,N            ; -sign & OVF ?

  [sign]   neg          ans,ans             ; return negative INT (q=0)

  [N]      mv           big,ans             ; or -OVF = 0x8000 0000.

           .return      ans

           .endproc
```

Convert SP FP to 40-Bit Signed Long Integer SA Listing for _fixfli

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPLONG.sa       Syd Poland    03-23-98      (for TMS320C62xx DSPs)
;                                              a4 ---> a5:a4
;       IEEE-754 short 32-bit SP Floating Point ---> 40-bit Signed Integer

        .text                           ; program section
        .global _SPLONG,__fixfli        ; entry labels
        .align  32                      ; start on fetch packet boundary

__fixfli:                               ; entry in rts6201 library

_SPLONG:  .cproc  arg1    ; 32-bit SP Floating Point ---> 40-bit Signed LONG

          .reg    exp,Base,sign,N,M,ansH:ansL

           extu         arg1,1,24,exp        ; exp. of input number
           mvk          127,Base             ; exp of |numbers| < 1.0

           shru         arg1,31,sign         ; input sign bit
           cmpltu       exp,Base,N           ; is input |#| < 1.0
           addah        Base,19,Base         ; max. base exp = 127 + 38

   [!N]    extu         arg1,9,2,arg1        ; get mant. field (|#| => 1.0)
   [N]     zero         arg1                 ; force a 0 if |arg1| < 1.0
   [N]     sub          exp,exp,exp          ; force exp=0 when |#| < 1.0
   [N]     mpy          sign,0,sign          ; set sign=0 when |#| < 1.0

   [exp]   set          arg1,30,30,arg1      ; add hidden 1 if exp is non-0
           sub          Base,exp,M           ; net shift amount
           cmpltu       Base,exp,N           ; is Base < exp ?

           shl          arg1,8,ansH:ansL     ; left justify mantissa

   [!N]    shru         ansH:ansL,M,ansH:ansL ; no, shift answer right (q=0)
   [N]     or           -1,ansL,ansL         ; yes, set to +OVF=0xffff ffff
   [N]     mpy          sign,N,N             ; -sign & OVF ?
   [N]     sub          sign,sign,sign       ; zero sign to appear +

   [sign]  neg          ansH:ansL,ansH:ansL  ; return negative INT (q=0)
   [N]     mvk          0x7f,ansH            ; yes, set to +OVF=0x7f as 8-bit

   [N]     add          1,ansH:ansL,ansH:ansL ; or -OVF = 0x80 0000 0000.

           .return      ansH:ansL

           .endproc
```

Convert SP FP to 32-Bit Unsigned Integer SA Listing for _fixfu

```
;      Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;      SPUINT.sa       Syd Poland    03-10-98       (for TMS320C62xx DSPs)
;                                a4 ---> a4
;      IEEE-754 short 32-bit SP Floating Point ---> 32-bit Unsigned Integer

       .text                                    ; program section
       .global _SPUINT,__fixfu                      ; entry labels
       .align  32                               ; start on fetch packet boundary

__fixfu:                                        ; entry in rts6201 library

_SPUINT: .cproc arg1                ; 32-bit Floating Point ---> 32-bit Unsigned Integer

        .reg   sign, Base, exp, N, ans, M

       shru    arg1,31,sign             ; input sign bit
       zero    exp                 ; set exp = 0
       mvk     127,Base                 ; exp of numbers < 1.0

 [sign] zero    arg1                ; force to zero if arg1 < 0
[!sign] extu    arg1,1,24,exp           ; exp. of input number

       cmpltu  exp,Base,N               ; is input # < 1.0
       addk    31,Base                  ; max. base exp. = 127 + 31

 [N]    zero    arg1                    ; force a 0 if arg1 < 1.0
 [N]    sub     exp,exp,exp             ; force exp=0 when # < 1.0
[!N]    extu    arg1,9,1,ans            ; get mant. field

 [exp]  set     ans,31,31,ans           ; add hidden 1 if exp is non-zero
       sub     Base,exp,M              ; net shift amount
       cmpltu  Base,exp,N               ; is Base < exp ?

 [!N]   shru    ans,M,ans               ; no, shift answer right (q=0)

 [N]    set     ans,0,31,ans           ; yes, set to all ones (OVF)

       .return ans

       .endproc
```

Convert SP FP to 40-Bit Unsigned Long Integer SA Listing for _fixful

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPULONG.sa       Syd Poland    03-11-98      (for TMS320C62xx DSPs)
;                                          a4 ---> a5:a4
;       IEEE-754 short 32-bit SP Floating Point ---> 40-bit Unsigned Integer

        .text                           ; program section
        .global _SPULONG,__fixful       ; entry labels
        .align  32                      ; start on fetch packet boundary

__fixful:                               ; entry in rts6201 library

_SPULONG   .cproc arg1     ; 32-bit SP Floating Point ---> 40-bit Unsigned INT

           .reg   sign,exp,Base,N,ones,ansH:ansL,M,VIII

           shru         arg1,31,sign         ; input sign bit
           zero         exp                  ; initialize exp=0
           mvk          127,Base             ; exp of numbers < 1.0

   [sign]  zero         arg1                 ; force to zero if arg1 < 0
  [!sign]  extu         arg1,1,24,exp        ; exp. of input number > 0

           cmpltu       exp,Base,N           ; is input # < 1.0
           addk         39,Base              ; max. base exp. = 127 + 39
           mvk          -1,ones              ; generate reg = all 1s

   [N]     zero         arg1                 ; force a 0 if arg1 < 1.0
   [N]     sub          exp,exp,exp          ; force exp=0 when # < 1.0
   [!N]    extu         arg1,9,1,ansL        ; get mant. field (# => 1.0)

   [exp]   set          ansL,31,31,ansL      ; add hidden 1 if exp is non-0
           sub          Base,exp,M           ; net shift amount
           cmpltu       Base,exp,N           ; is Base < exp ?

           shl          ansH:ansL,8,ansH:ansL ; left justify #
           mvk          255,VIII              ; 8 MS-bits = all 1s in 40-bits

   [!N]    shru         ansH:ansL,M,ansH:ansL ; no, shift answer right (q=0)
   [N]     add          ones,0,ansL           ; yes, set to all 1s (OVF)
   [N]     add          VIII,0,ansH           ; yes,  "   8-bits=1s

           .return      ansH:ansL

           .endproc
```

Convert 32-Bit Signed Integer to SP FP SA Listing for _fltif

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       INTSP.sa       Syd Poland    03-07-98       (for TMS320C62xx DSPs)
;                          a4 ---> a4
;       32-bit Signed Integer ---> 32-bit IEEE-754 SP Floating Point

        .text                               ; program section
        .global _INTSP,__fltif              ; subroutine entry labels
        .align  32                          ; start on fetch packet boundary

__fltif:                ; entry in rts6201 library

_INTSP: .cproc  arg1    ; Signed 32-bit Integer ---> 32-bit Floating Point

        .reg    ans,sign,mant,exp,N,Base

        abs     arg1,mant               ; absolute value of mantissa
        clr     arg1,0,30,sign          ; get input sign bit

        lmbd    1,mant,N                ; # of leading sign bits
        mvk     127+31,Base             ; base exponent for INTEGER

        shl     mant,N,ans              ; normalize absolute mantissa
        sub     Base,N,exp              ; output exponent for non-zero mant

        shl     exp,23,exp              ; shift value to exponent field

        extu    ans,1,9,ans             ; shift to mantissa field
        add     sign,exp,exp            ; combine sign and exponent fields

  [!mant] zero  ans                     ; return zero answer if arg1 = 0
   [mant] add   exp,ans,ans             ; insert sign and exponent fields

        .return ans

        .endproc
```

Convert 40-Bit Signed Integer to SP FP SA Listing for _fltlif

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       LONGSP.sa       Syd Poland    03-07-98      (for TMS320C62xx DSPs)
;                       a5:a4 ---> a4
;  40-bit Signed LONG integer ---> IEEE-754 32-bit SP Floating-Point

         .global _LONGSP,__fltlif ; entry labels
         .align  32               ; start on fetch packet boundary
         .text                    ; program section

__fltlif:                                   ; entry in rts6201 library

_LONGSP:  .cproc hi1:low1

          .reg  sign,Base,N,exp,mant,ans

          extu     hi1,24,31,sign           ; copy sign bit
          abs      hi1:low1,hi1:low1        ; get absolute value of arg1
          mvk      127+38,Base              ; base exponent for LONG INT

          shl      sign,31,sign             ; move sign to sign field
          norm     hi1:low1, N              ; # of leading sign bits

          sub      Base,N,exp               ; output exp for non-0 mant
          shl      hi1:low1,N,hi1:low1      ; normalize mantissa

          shru     hi1:low1,15,hi1:low1     ; shift to mantissa field

          shl      exp,23,exp               ; shift value to exp field
          abs      low1,mant                ; absolute value of mantissa

          add      sign,exp,exp             ; combine sign and exp fields
          clr      mant,23,31,ans           ; clear sign and exp fields

  [!mant] zero     ans                      ; return zero if mantissa = 0
   [mant] add      exp,ans,ans              ; insert sign and exp fields

          .return  ans

          .endproc
```

Convert 32-Bit Unsigned Integer to SP FP SA Listing for _fltuf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       UINTSP.sa       Syd Poland    03-07-98        (for TMS320C62xx DSPs)
;                            a4 ---> a4
;       Unsigned 32-bit Integer ---> IEEE-754 32-bit SP Floating-Point

        .global _UINTSP,__fltuf ; entry labels
        .text                   ; program section
        .align  32              ; start on fetch packet boundary

__fltuf:                        ; entry in rts6201 library

_UINTSP: .cproc  arg1  ; Unsigned 32-bit Integer ---> 32-bit SP Floating-Point

         .reg       N,Base,ans,exp,mant

         lmbd       1,arg1,N             ; find leading bit = 1

         mvk        127+31,Base          ; base exponent for Unsigned INTEGER
         shl        arg1,N,ans           ; normalize mantissa

         sub        Base,N,exp           ; output exponent for non-zero mant
         shru       ans,8,mant           ; shift to IEEE-754 mantissa field

         shl        exp,23,exp           ; shift value to exponent field
         clr        mant,23,31,ans       ; clear sign and exponent fields

 [mant]  add        exp,ans,ans          ; insert sign and exponent fields
 [!mant] zero       ans                  ; return zero answer if arg1 = 0

        .return     ans

        .endproc
```

Convert 40-Bit Unsigned Integer to SP FP SA Listing for _fltulf

```
;       Copyright 1998 by Texas Instruments inc.  All rights reserved.

;       ULONGSP.sa       Syd Poland    03-07-98       (for TMS320C62xx DSPs)
;                         a5:a4 ---> a4
;  40-bit Unsigned LONG integer ---> 32-bit IEEE-754 SP Floating-Point

        .text                           ; program section
        .global _ULONGSP,__fltulf       ; entry labels
        .align  32                      ; start on fetch packet boundary

__fltulf:                               ; entry in rts6201 library

_ULONGSP: .cproc  hi1:arg1  ; Unsigned 40-bit Integer ---> 32-bit Floating Point

          .reg    mant,exp,N,Base,ans,hi2:arg2

          shru    hi1:arg1, 1, hi2:arg2   ; force + sign bit

          norm    hi2:arg2, N             ; find 1st one

          mvk     127+39,Base             ; base exponent for LONG INT (q=0)
          shl     hi1:arg1, N, hi1:arg1   ; normalize mantissa

          sub     Base,N,exp              ; output exponent for non-zero mant
          shru    hi1:arg1, 16, hi1:arg1  ; shift to mantissa field

          mv      arg1,mant               ; save mantissa for test
          shl     exp,23,exp              ; shift value to exponent field
          clr     arg1,23,31,arg1         ; clear sign and exponent fields

 [mant]   add     exp,arg1,ans            ; insert sign and exponent fields
 [!mant]  zero    ans                     ; return zero answer if mantissa = 0

          .return ans

          .endproc
```

SP Floating Point Multiplication SA Listing for _mpyf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPMPY.sa      Syd Poland    04-07-98        (for TMS320C62xx DSPs)
;                                 a4 * b4 ---> a4
;       32-bit SP Floating Point Multiply ---> 32-bit SP Floating Point
;       using IEEE-754-1985 format

        .text                   ; program memory
        .global _SPMPY,__mpyf   ; entry labels
        .align  32              ; Fetch Packet boundary

__mpyf:                         ; entry in rts6201 library

_SPMPY: .cproc  arg1, arg2

        .reg    ans,exp1,exp2,sign,mant1,mant2,Base,U,HH,HL,LH,LL,Big,OVF
        .reg    Hi:Lo

        extu         arg1,1,24,exp1  ; get arg1's exp.field
        extu         arg2,1,24,exp2  ; get arg2's exp.field
        xor          arg1,arg2,sign  ; form output sign

  [exp1]  extu       arg1,9,9,mant1  ; get arg1's mant field if exp1 > 0
  [exp2]  extu       arg2,9,9,mant2  ; get arg2's mant field if exp2 > 0
  [!exp1] zero       mant1           ; zero mant1 field if exp1=0
  [!exp2] zero       mant2           ; zero mant2 field if exp2=0
  [!exp2] zero       sign,0,sign     ; zero output sign if exp2=0

  [exp1]  set        mant1,23,23,mant1       ; add hidden 1 if exp1 > 0
  [exp2]  set        mant2,23,23,mant2       ; add hidden 1 if exp2 > 0
  [!exp1] zero       sign                    ; zero output sign if exp1=0
          add        exp1,exp2,exp1          ; sum of 2 input exponents

        mvk          127,Base        ; exponent bias
        mpyu         mant1,mant2,LL  ; low * low     (u*u)
        mpyhu        mant1,mant2,HH  ; high * high   (u*u)
        clr          sign,0,30,sign  ; keep only the output sign bit (31)

         mpyhlu      mant1,mant2,HL  ; high * low    (u*u)
         mpyhlu      mant2,mant1,LH  ; low * high    (u*u)
  [exp1] sub         exp1,Base,exp1  ; remove extra bias of 127 if exp1>0
         mvk         254,Big         ; Max. exp permitted

        shru         LL,16,LL        ; L*L >>u 16
        shl          HH,16,HH        ; HH << 16 (orig.top halfword == 0)
        cmpgt        exp1,Big,OVF    ; check for overflow (exp1 > Big)
        cmplt        exp1,1,U        ; check for underflow (exp1 < 1)

        addu         LH,HL,Hi:Lo     ; Hi:Lo = H*L + L*H
        add          HH,LL,HH        ; HH = HH<<16 + LL >>u 16
   [!U] extu         exp1,24,1,exp1  ; shift output 8-bit exp to EXP field

        addu         HH,Hi:Lo,Hi:Lo  ; final sum of all 4 products
        zero         mant1           ; force mant1 to all zeros

        shru         Lo,31,exp2       ; get MS-bit of final sum
        set          mant1,0,22,mant1 ; set mant1 = all 1's (OVF)
        mv           Lo,ans           ; move unshifted Lo to ans

   [!U]  extu        ans,2,9,ans     ; output 1 <= mant < 2 (w/o hidden 1)
   [!U]  or          sign,exp1,sign  ; combine sign and exp fields
  [exp2] shru        Lo,8,mant2      ; shift right 1 bit (keep hidden 1
                                     ; as an exp inc of 1) 2 <= mant < 4
  [exp2] mv          mant2,ans       ; move mant2 to ans if exp2 = 1
```

```
  [OVF]  add         mant1,0,ans     ; set Max mant for OVF
  [OVF]  set         sign,23,30,sign ; set max. exp for OVF, keep sign bit

  [U]    zero        ans             ; return 0 on UNDERFLOW
  [!U]   add         sign,ans,ans    ; include ouput sign if no UNDERFLOW

         .return     ans

         .endproc
```

SP FP Subtraction SA Listing for _subf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SPSUB.sa      Syd Poland    04-28-98        (for TMS320C62xx DSPs)
;                                 a4 - b4 ---> a4
;       32-bit SP Floating Point Addition ---> 32-bit SP Floating Point
;                 or Subtraction  a4 + b4 ---> a4
;
;       using IEEE-754-1985 format

        .text                                ; program memory

        .global _SPADD,_SPSUB,__addf,__subf ; entry labels for ADD or SUB

        .align  32                           ; Fetch Packet boundary

;__addf:                                  ; entry in rts6201 library

;_SPADD:                                  ; ans = arg1 + arg2 entry

;        zero    .D2     b5              ; zero mant2
;||      cmpeq   .L2     b4,0,b2         ; is arg2 = 0 ?

;        set     .S2     b5,31,31,b5     ; set mant2 sign bit = 1

;  [!b2] xor     .L2     b4,b5,b4        ; invert non-0 arg2's sign bit

__subf:                                  ; entry in rts6201 library

_SPSUB: .cproc  arg1, arg2               ; ans = arg1 - arg2 entry

        .reg    ans,exp1,exp2,sign1,sign2,mant1,mant2

        .reg    W,V,N,T,AZ,U,S,Z

        extu         arg1,1,24,exp1  ; get arg1's exp.field
        extu         arg2,1,24,exp2  ; get arg2's exp.field
        cmplt        arg1,0,sign1    ; input sign1 < 0 ?
        cmplt        arg2,0,sign2    ; input sign2 < 0 ?

  [exp1]  extu       arg1,9,9,mant1  ; get arg1's mant field if exp1 > 0
  [exp2]  extu       arg2,9,9,mant2  ; get arg2's mant field if exp2 > 0
  [!exp1] zero       mant1           ; zero mant1 field if exp1=0
  [!exp2] zero       mant2           ; zero mant2 field if exp2=0
          mpy        Z,0,Z                ; force a zero

  [exp1]  set        mant1,23,23,mant1       ; add hidden 1 if exp1 > 0
  [exp2]  set        mant2,23,23,mant2       ; add hidden 1 if exp2 > 0
  [!exp1] zero       sign1                   ; zero sign1 if exp1=0
  [!exp2] zero       sign2                   ; zero sign2 if exp2=0
          sub        exp1,exp2,W             ; dif. of 2 input exponents
          cmplt      exp1,exp2,V             ; exp1 < exp2 ?
```

```
 [sign1] sub       0,mant1,mant1          ; set mant1 = -mant1 ?
[!sign2] sub       Z,mant2,mant2          ; set mant2 = -mant2 ?
 [V]     mv        exp2,exp1              ; move bigger exp2 to exp1
         abs       W,N                    ; get absolute N
         mvk       25,T                   ; T = max. shift amount

       cmpltu      N,T,T                  ; is N < T (25) ?
 [V]   mv          mant1,mant2            ; swap mant1 and mant2
 [V]   mv          mant2,mant1            ; if exp2 > exp1

 [T]   shr         mant2,N,mant2          ; shift mant2 right (N is OK)
 [!T]  zero        mant2                  ; zero mant2 (N is too big)


       add         mant1,mant2,ans        ; add 2 mantissas

       extu        ans,0,31,sign1         ; output sign
       abs         ans,ans                ; get |mantissa|
       addab       exp1,7,exp1            ; exp1 adjust pre-norm
       mvk         254,mant2              ; max. exp
       cmpeq       ans,0,AZ               ; is ans = 0 ?

       norm        ans,W                  ; normalize count
       shl         sign1,31,S             ; shift to sign field

       shl         ans,W,ans              ; normalize mantissa
       subab       exp1,W,exp1            ; output exp w/norm N

       shru        ans,7,ans              ; shift to mantissa field
       cmplt       exp1,1,U               ; exp < 1 (underflow) ?
       cmpgt       exp1,mant2,V           ; exp >254 (overflow) ?

       clr         ans,23,31,ans          ; keep only mant.field
 [AZ]  add         AZ,0,U                 ; if ans=0, force U=1

       shl         exp1,23,exp1           ; shift to exp.field

 [!V]  add         exp1,ans,ans           ; combine exp/mant fields
 [V]   set         ans,0,30,ans           ; set max exp/mant=all 1s

 [!U]  add         ans,S,ans              ; include sign bit in answer
 [U]   zero        ans                    ; set to 0 on underflow

       .return ans

       .endproc
```

NOTE:

This listing has the SP FP addition `__addf` instructions commented out. However, if space is primary and speed is secondary, remove the comments and reassemble; do not link in the separate add subroutine. This assumes subtraction dominates addition.

Performance

This section presents two classes of measurements using Tools version 2.00:

- 40-point dot product times
- 12 functions sample execution times

40-Point Dot Product Cycle Times

This table demonstrates the run-time-support libraries for the C6201 and C6701 as used in the 40-point dot product routine. The column labeled **SPFP ASM Cycles** uses the new single-precision floating-point routines in place of the run-time-support functions. The two chip times are normalized relative to the ASM times.

C Opt.	SPFP ASM Cycles	C rts6201 Vs 2.00 Cycles	6201/ASM Ratio	C rts6701 Vs 2.00 Cycles	6701/ASM Ratio
-o3	1941	7724	3.98	189	0.10

NOTE:
The C67xx run-time-support library uses the hardware floating-point instructions when possible. These ratios do not consider the different clock rates—C6201 = 200 MHz and C6701 = 167 MHz. This translates into a cycle time of 5 and 6 nanoseconds, respectively, for the two chip families.

40-Point Dot Product C Source Listing

```
/* Copyright 1998 by Texas Instruments Inc.  All rights reserved.  */

/* Prototype for DOTP.C  (Dot Product)   Syd Poland      06-11-98  */

float dotp(float *m, float *n, int count) ;

/* Declarations */

float a[40] = { 40,39,38,37,36,35,34,33,32,31,
                30,29,28,27,26,25,24,23,22,21,
                20,19,18,17,16,15,14,13,12,11,
                10, 9, 8, 7, 6, 5, 4, 3, 2, 1 } ;
float x[40] = {  1, 2, 3, 4, 5, 6, 7, 8, 9,10,
                11,12,13,14,15,16,17,18,19,20,
                21,22,23,24,25,26,27,28,29,30,
                31,32,33,34,35,36,37,38,39,40 } ;
float y = 0 ;

/* Main Code */

main()
{
  y = dotp(a, x, 40) ;
}

/* Dot Product function */

float dotp(float *m, float *n, int count)
{
  int i ;
  float sum = 0 ;

  for (i=0; i < count; i++)
  {
    sum = sum + m[i] * n[i] ;
  }
  return(sum) ;
}
```

12 Functions Sample Execution Times

This table demonstrates the run-time-support libraries for the C6201 Tools v 2.00 as used in the 12 functions test routine. The column labeled **ASM Cycles** uses the new single-precision floating-point routines in place of the run-time-support functions. The chip times are normalized relative to the ASM times.

NOTE:
These times include the function call (6 cycles), the function execution and return, plus the main C program issuing a store answer to memory (2 or 3 cycles for 32 bits or 40 bits).

Test Code	ASM Name	rts Name	ASM Cycles	rts6201 Vs 2.00 Cycles	rts6201/ ASM
a = k	INTSP	_fltif	14	32	2.3
b = uk	UINTSP	_fltuf	14	27	1.9
j = a	SPINT	_fixfi	15	46	3.1
uj = b	SPUINT	_fixfu	15	53	3.5
a = n	LONGSP	_fltlif	15	42	2.8
b = un	ULONGSP	_fltulf	14	35	2.5
m = a	SPLONG	_fixfli	16	46	2.9
um = b	SPULONG	_fltful	16	52	3.3
C = a + b	SPADD	_addf	22	109	5.0
E = c - a	SPSUB	_subf	22	132	6.0
F = a * b	SPMPY	_mpyf	21	89	4.2
g = f / a	SPDIV	_divf	50	168	3.4
Sum			234		
Sum / 12			19.5	69.3	3.4
rts6201/ASM				3.6	

TI Contact Numbers

INTERNET

TI Semiconductor Home Page
www.ti.com/sc

TI Distributors
www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas
Phone +1(972) 644-5580
Fax +1(972) 480-7800
Email sc-infomaster@ti.com

Europe, Middle East, and Africa
Phone
- Deutsch +49-(0) 8161 80 3311
- English +44-(0) 1604 66 3399
- Español +34-(0) 90 23 54 0 28
- Francais +33-(0) 1-30 70 11 64
- Italiano +33-(0) 1-30 70 11 67

Fax +44-(0) 1604 66 33 34
Email epic@ti.com

Japan
Phone
- International +81-3-3457-0972
- Domestic 0120-81-0026

Fax
- International +81-3-3457-1259
- Domestic 0120-81-0036

Email pic-japan@ti.com

Asia
Phone
- International +886-2-23786800
- Domestic
  - Australia 1-800-881-011
    - TI Number -800-800-1450
  - China 10810
    - TI Number -800-800-1450
  - Hong Kong 800-96-1111
    - TI Number -800-800-1450
  - India 000-117
    - TI Number -800-800-1450
  - Indonesia 001-801-10
    - TI Number -800-800-1450
  - Korea 080-551-2804
  - Malaysia 1-800-800-011
    - TI Number -800-800-1450
  - New Zealand 000-911
    - TI Number -800-800-1450
  - Philippines 105-11
    - TI Number -800-800-1450
  - Singapore 800-0111-111
    - TI Number -800-800-1450
  - Taiwan 080-006800
  - Thailand 0019-991-1111
    - TI Number -800-800-1450
  - Fax 886-2-2378-6808
  - Email tiasia@ti.com

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE (“CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty, or endorsement thereof.