Signed Integer Arithmetic on the HPC™

National Semiconductor
Application Note 603
Raj Gopalan
July 1989

This report describes the implementation of signed integer arithmetic operations on the HPC. HPC hardware support for unsigned arithmetic operation. In order to support signed integer arithmetic operations on the HPC, the user can represent negative numbers in two's complement form and perform the signed arithmetic operations explicitly through software.

The following signed integer arithmetic routines are implemented in the package:

Multiplication:

16 by 16 yielding 16-bit result
32 by 32 yielding 32-bit result

Division:

16 by 8 yielding 16-bit quotient and 16-bit remainder
32 by 16 yielding 16-bit quotient and 16-bit remainder
32 by 32 yielding 16-bit quotient and 16-bit remainder

Addition:

16 by 16 yielding 16-bit

Subtraction:

16 by 16 yielding 16-bit

Comparison:

16 by 16 for greater to, less than or equal to.

REPRESENTATION OF NEGATIVE NUMBERS:

For binary numbers, negative numbers are represented in two's complement form. In this system, a number is positive if the MSB is 0, negative if it is 1.

The decimal equivalent of two's complement number is computed the same as for an unsigned number, except that weight of the MSB is $-2^{**}n - 1$ instead of $+2^{**}n - 1$. The range of representable numbers is $-(2^{**}n - 1)$ through $+(2^{**}n - 1 - 1)$.

The two's complement of a binary number is obtained by complementing its individual bits and adding one to it.

The advantage of representing a negative number in two's complement form is that addition and subtraction can be done directly using unsigned hardware.

```
          .title        SIMUSL
          .sect         code,rom8,byte,rel
;Signed multiply (16 by 16)
;         B             Multiplicand
;         A             Multiplier
;         X;A           return
;
          .public signed_mult_16
          .local
signed_mult_16:
          st            a,0.w
          mult          a,b                         ;do unsigned multiplication.
          sc
          ifbit         7,(1).b                     ;if multiplier is negative
          subc          x,b
          sc
          ifbit         7,(B+1).b                   ;if multiplicand is negative
          subc          x,0.w
$exit:
          ret

          .endsect
```

HPC™ is a trademark of National Semiconductor Corporation.

MULTIPLICATION

Method 1:

Signed multiplication can be achieved by taking care of the signs and magnitudes of the multiplicand and multiplier separately.

Perform the multiplication on the magnitudes alone.

The sign of the result can be set based on the signs of the multiplier and the multiplicand.

Method 2:

This method does not require finding the magnitude of the operands. Multiplication can be done using unsigned hardware on the two's complement numbers. The result will be signed based on the signs of the operands.

The algorithm is as follows:

Step 1. Result = op1 * op2

Step 2. If op1 < 0 then subtract op2 from upper half of the result.

Step 3. If op2 < 0 then subtract op1 from upper half of the result.

Now the Result will yield the correct value of the multiplication on two's complement numbers.

Method 3:

By sign extending the multiplier and multiplicand to the size of the result one can always obtain the correct result of signed multiplication using unsigned multiplication.

```
          .title        SIMULL
          .sect         code,rom8,byte,rel
;Multiply (Signed or Unsigned are the same)
;32 bit
;         K:A           Multiplicand
;         -4:6[SP]      Multiplier
;         K:A           return
;
          .public multiply_32
          .local
multiply_32:
          push          x                         ;(Argument now at -6:8[SP])
          st            a,0.w
          ld            a,k                       ;Multiply hi reg* lo stack
          mult          a,-8[sp].w
          x             a,0.w                     ;hold, retrieve lo reg
          push          a                         ;(argument now at -8:10[SP])
          mult          a-8[sp].w                 ;Multiply lo reg* hi stack
          add           0.w,a                     ;add into hi partial
          pop           a                         ;(Argument now at -6:8[SP])
          mult          a,-8[sp].w                ;Multiply lo reg* lo stack
          add           x,0.w                     ;add in hi partial
          ld            k,x                       ;Position
          pop           x                         ;Restore
          ret

          .endsect
```

DIVISION

Similar to multiplication method 1, one can perform the division on the magnitudes of the dividend and divisor.
The sign of the quotient can be set based on the signs of the dividend and the divisor.
The sign of the remainder will be same as the dividend.

```
          .title        SIDVSS
          .sect         code,rom8,byte,rel

;Division & Remainder
;16,8 bit (signed only, unsigned uses inline code)
;         A             Dividend
;         -4[SP]        Divisor
;         A             return
;
          .public signed_divide_8,signed_remainder_8
          .public signed_divide_16,signed_remainder_16
          .local
signed_divide_8:
          jsr           $shared_8                           ;Uses shared routine
          ret
;
signed_remainder_8:
          jsr           $shared_8                           ;Uses shared routine
          ld            a,k                                 ;Return remainder
          ret
;
$shared_8:
          ifgt          a,#0x7f
          or            a,#0xff00
          st            a,k                                 ;Get arguments
          ld            a,-6[sp].w
          ifgt          a,#0x7f
          or            a,#0xff00
          jp            $shared
;
signed_divide_16:
          jsr           $shared_16                          ;Uses shared routine
          ret
;
signed_remainder_16:
          jsr           $shared_16                          ;Uses shared routine
          ld            a,k                                 ;Return remainder
          ret
;
$share_16:
          st            a,k                                 ;Get arguments
          ld            a,-6[sp].w
$shared
          ifeq          a,#0
          ret                                               ;division by zero
          push          x
          ifgt          a,#0x7fff
          jp            $unknown_negative                   ;unknown/negative
          x             a,k
          ifgt          a,#0x7fff
          jp            $negative_positive                  ;negative/positive
          div           a,k                                 ;Positive/positive is plus,plus
          jp            $positive_positive
```

```
$unknown_negative:                                                   ;Unknown/negative
          comp          a
          inc           a
          x             a,k
          ifgt          a,#0x7fff
          jp            $negative_negative                           ; negative/negative
          div           a,k                                          ;Positive/negative is minus,plus
          comp          a
          inc           a
$positive_positive:
          ld            k,x
          jp            $exit
$negative_positive:                                                  ;Negative/positive is minus,minus
          comp          a
          inc           a
          div           a,k
          comp          a
          inc           a
          jp            $negate_remainder
$negative_negative:                                                  ;Negative/negative is plus,minus
          comp          a
          inc           a
          div           a,k
$negate_remainder:
          x             a,x
          comp          a
          inc           a
          st            a,k
          ld            a,x
$exit:
          pop           x
          ret

          .endsect
```

```
        .title        SIDVLS
        .sect         code,rom8,byte,rel


;Division & Remainder
;Signed 32 by 16 divide
;         X;A          Dividend
;         K            Divisor
;         X,A          return (remainder and quotient)
;
        .public signed_div_32
        .local
signed_div_32:
        sc
        ifeq          k,#0
        ret                                       ;Divide by zero, set carry and return
$shared_signed:
        ifbit         7,x+1.b
        jp            $negative_dividend
        jsr           $process_divisor            ;Skipping return
        ret                                       ;+/+=+,+
$negate_quotient:
        comp          a
        inc           a
        ret                                       ;+/-= -,+
$negative_dividend;
        comp          a
        add           a,#01
        x             a,x
        comp          a
        adc           a,#0
        x             a,x
        jsr           $process_divisor            ;skipping return
        jsr           $negate_quotient            ;-/+=-,-
$negate_remainder:                                ;-/-=+,-
        x             a,x
        comp          a
        inc           a
        x             a,x
        ret
$process_divisor:
        ifbit         7,k+1.b
        jp            $negative_divisor
        divd          a,k                         ;?/+
        ret
$negative_divisor:
        x             a,k
        comp          a
        inc           a
        x             a,k
        divd          a,k                         ;?/-
        retsk

        .endsect
```

```
          .title        SUDVLL
          .sect         code,rom8,byte,rel

;Division & Remainder
;Signed 32 by 32 Divide
;         K:A           Dividend
;         -4:6[SP]      Divisor
;         K:A           return
;
;Stack frame as built and used consists of
;top:
;         0, initial subtrahend hi /dividend shifts into subtrahend
;         0, initial subtrahend lo /becomes remainder
;         k, dividend hi /dividend shifts into subtrahend, and
;         a, dividend lo /quotient shifts into dividend
;         b preserved
;         x preserved
;         return address
;         sp-4-12, divisor hi
;         sp-6-12, divisor lo
;Sign flag (0 = negative, 1 = positive, for test sense at exit)
;bit 0, divisor sign (1 = negative)
;bit 1, dividend sign (1 = positive)
;Inc of flag causes bit 1 = (bit 1 xor bit 0) by carry/nocarry out of bit 0
;so that two positives (010) or two negatives (001) indicate a positive
;quotient (011 or 010) in bit 1. Bit 1 always indicates sign if remainder.
;Operation is indicated by bit 3 of the flag, 1 = remainder.
;
          .public signed_divide_32, signed_remainder_32
          .public unsigned_divide_32, unsigned_remainder_32
          .local
signed_divide_32:
          ld            l.b,#0x02
          jp            $shared_signed
;
signed_remainder_32:
          ld            l.b,#0x0a
$shared_signed:
          ifbit         7,k+1.b                       ;Check dividend
          jsr           $negate                       ;Negate dividend and note sign
          ifbit         7,-6+3[sp].b                  ;Check divisor
          jp            $negate_divisor
          jmp           $shared
;


$negate_divisor:
          x             a,-6[sp].w                    ;Negate divisor and note sign
          comp          a
          add           a,#1
          x             a,-6[sp].w
          x             a,-4[sp].w
          comp          a
          adc           a,#0
          x             a,-4[sp].w
          sbit          0,l.b
          jp            $shared
;
unsigned_divide_32:
          ld            l.b,#0x02
          jp            $shared
;
unsigned_remainder_32:
          ld            l.b,#0x0a
```

```
$shared:
          push      x                                   ;Preserve registers
          push      b
          ld        b,sp                                ;Place dividend, becomes quotient
          push      a
          push      k
          ld        x,sp                                ;Set subtrahend, becomes remainder
          clr       a
          push      a
          push      a
          ld        k,#-18                              ;Access divisor argument
          add       k,sp
          ld        a,[k].w
          or        a,2[k].w
          ifeq      a,#0
          jmp       $zero                               ;division by zero
          ld        0.b,#32                             ;Set counter
$loop:
          ld        a,[b].w                             ;Shift Dividend:Quotient
          shl       a
          xs        a,[b+].w
          nop
          ld        a,[b].w
          rlc       a
          xs        a,[b-].w
          nop
          ld        a,[x].w
          rlc       a
          x         a,[x+].w
          ld        a,[x].w
          rlc       a
          x         a,[x-].w
          ifc
          jp        $subtract                           ;Carry out - dividend divisor
          sc                                            ;Check for dividend divisor
          ld        a,[x+].w
          subc      a,[k].w
          ld        a,[x-].w
          subc      a,2[k].w
          ifnc
          jp        $count                              ;dividend divisor
$subtract:
          ld        a,[x].w                             ;Subtract out divisor (c is set)
          subc      a,[k].w
          x         a,[x+].w
          ld        a,[x].w
          subc      a,2[k].w
          x         a,[x-].w
          sbit      0,[b].b                             ;Set quotient bit
$count:
          decsz     0.b                                 ;Count 32 shifts
          jmp       $loop
$zero:
          pop       k                                   ;Get Remainder and/or Quotient
          pop       a                                   ;and clear working off stack
          pop       x
          pop       b
          ifbit     3,1.b
          jp        $exit                               ;want remainder, have it
          ld        a,b                                 ;Want Quotient
          ld        k,x
          inc       1.b                                 ;Divisor's sign Xors Dividend's
```

```
$exit:
        pop         b                                   ;Restore registers
        pop         x
        ifbit       1,1.b
        ret                                             ;positive result
$negate:
        comp        a                                   ;Negate K:A
        add         a,#1
        x           a,k
        comp        a
        adc         a,#0
        x           a,k
        rbit        1,1.b                               ;Note sign (for entrance)
        ret

        .endsect
```

ADDITION

Two's complement numbers can be added by ordinary binary addition, ignoring any carries beyond the MSB. The result will always be the correct sum as long as the result doesn't exceed the range.

If the result is the same as for the subtrahend, then overflow has occurred.

```
          .title        SIADD
          .sect         code,rom8,byte,rel
;Signed add (16 by 16)
;         A             Operand1
;         B             Operand2
;         Carry         Return
          .public sign_add
          .local

sign_add:
          ld            0.b,#00
          ifbit 7,(A+1).b
          inc           0.b
          ifbit 7,(B+1).b
          inc           0.b

          ;if bit 0 of 0.b = 1 then op1 and op2 have different sign
          ;if bit 0 of 0.b = 0 then op1 and op2 sign are same
          ;then if bit 1 of 0.b = 0 both operands are positive
          ;else both operands are negative.

          add           a,b                                     ;Perform unsigned addition
          rc
          ifbit 0,0.b                                           ;both operands are different sign
          ret
          ifbit 1,0.b                                           ;both op1 and op2 are negative
          jp $negatives
$positives:                                                     ;both op1 and op2 are positive
          ifbit 7,(A+1).b                                       ;if result sign is negative then
                                                                 set overflow bit
          sc                                                    ;overflow
          ret
$negatives:
          ifbit 7,(A+1).b                                       ;if sign bit of result is
                                                                 negative, then no overflow

          ret
          sc                                                    ;overflow
$exit:
          ret

          .endsect
```

SUBTRACTION

Subtraction can be achieved by negating the subtrahend and perform the addition operation.

Overflow can be detected as mentioned before by checking the signs of minuhend and the negation of the subtrahend and that of the sum.

```
          .title        SISUB
          .sect         code,rom8,byte,rel

;Signed subtract (16 by 16)

;         B             Operand1
;         A             Operand2
;         Carry,A       Return
          .public sign_sub
          .local
sign_sub:
          ld            0.b,#00                            ;initialize sign flags
          ifbit         7,(B+1).b
          inc 0.b
$negate_A:
          comp A
          inc A
$ngative_comp_A:
          ifbit 7,(A+1).b
          inc           0.b
          ;if bit 0 of 0.b = 1 then op1 and op2 have different sign
          ;if bit 0 of 0.b = 0 then op1 and op2 sign are same
          ;then if bit 1 of 0.b = 0 both operands are positive
          ;else both operands are negative.
          add A,B                                          ;Perform unsigned addition
          rc
          ifbit 0,0.b                                      ;both operands are different sign
          ret
          ifbit 1,0.b                                      ;both op1 and op2 are negative
          jp $negatives
$positives:                                                ;both op1 and op2 are positive
          if bit 7, (A+1).b                                ;if result sign is negative then
                                                            set overflow bit
          sc                                               ;bit 0 of byte 0.b is set to
                                                            indicate overflow
          ret
$negatives:
          ifbit 7, (A+1).b                                 ;if sign bit of result is
                                                            negative, then no overflow
          ret
          sc                                               ;sign bit of result is positive,
                                                            hence overflow.
$exit:ret

          .endsect
```

```
          .title        NSISUB
          .sect         code,rom8,byte,rel

;Signed sub (16 by 16)

;         A             Operand1
;         B             Operand2
;         Carry         Return
          .public sign_sub
          .local
sign_sub:
          ld            0.b,#00
          ifbit 7,(A+1).b
          inc           0.b
          ifbit 7,(B+1).b
          inc           0.b
          ;if bit 0 of 0.b = 1 then op1 and op2 have different sign
          ;if bit 0 of 0.b = 0 then op1 and op2 sign are same
          ;then if bit 1 of 0.b = 0 both operands are positive
          ;else both operands are negative.
          sc
          subc          a,b                                   ;Perform unsigned addition
          rc
          ifbit 0,0.b                                         ;both operands are different sign
          jp            $chkovf
          ret                                                 ;both operands are same sign,
                                                               can't produce overflow
$chkovf:
          ifbit         7,(B+1).b
          jp            $negminu
$posminu:
          ifbit         7,(A+1).b
          sc
          ret
$negminu:
          ifbit         7,(A+1).b
          sc
          ret

          .endsect
```

COMPARISON

To do signed comparison on n bit two's complement numbers first add 2**(n − 1) to the numbers. This will basically shift the numbers from −(2**n − 1) to +(2**n − 1 − 1) range to 0 to 2**n − 1.

Now comparison operations on the numbers will produce the correct result.

```
          .title       SICMP
          .sect        code,rom8,byte,rel

;Signed compare (16 by 16)

;         A            Operand1
;         B            Operand2
;         0.b          Return=00                          if a = b
;                             02                          if a > b
;                             01                          if a < b
signed_compare:
          push         a
          push         b
          add          a,#08000
          add          b,#08000
          ifgt         a,b
          jp           $great
          ifeq         a,b
          jp           $equ
$less:
          ld           0.b,#01
          pop          b
          pop          a
          ret
$great:
          ld           0.b,#02
          pop          b
          pop          a
          ret
$equ:
          ld           0.b,#00
          pop          b
          pop          a
          ret

          .endsect
```


National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998