Mosaic Industries

Conversion of HEX ASCII Floating Point Number to Binary IEEE Format

APPLICATION NOTE MI-AN-050

Summary

The following explains how to convert a HEX ASCII floating point number to binary IEEE format.

Description

The situation:

We have 8 hex ascii bytes coming from the QED Board, and we need
to convert these to standard IEEE binary format.

Each hex ascii byte represents 4 bits of the QED-formatted FP number.
The hex ascii character corresponding to the most significant bits
are transmitted first.

QED Format:
 least significant 16 bits (0-15): 16-bit mantissa
 bits 16-23: signed 2s-complement 8bit exponent (unbiased)
 bits 24-31: sign: 0xFF -> negative; 0x01 -> positive; 00-> zero

IEEE Format:
 least significant 23 bits (0-22): 23-bit mantissa
 bits 23-30: biased 8bit exponent; actual exponent = {bits 23~30) - 127.
 bit 31: sign: 1 -> negative; 0 -> positive

Both formats use the "hidden bit" approach, where the magnitude of the
result equals {1.mantissa} * {2^exp};
the sign of the result is given by the sign byte or bit.

The first 2 hex ascii chars represent the sign byte; this is the most significant byte of the QED FP#. The only valid values are
00, 01, or FF. Any other values represents an error condition ("NAN").

The next 2 hex ascii chars represent the signed 2s-complement exponent
of the QED FP#.

The final 4 hex ascii chars represent the 16-bit mantissa of the QED FP#.

We use C pseudo-code to describe the required transformation.
DATA STRUCTURES FOR THE CONVERSION:

Declare the following temporary variables:

```
char  Qsign_var;
char  Qexponent_var;

char  IEEE_sign_var;
char  IEEE_exponent_var;
int   IEEE_mantissa_var;
```

Initialize the temporary variables to zero before each conversion.

Define the following union and zero it before each conversion
 {ie, make_fp.shiftable_fp = 0;}

```
union
{   struct
    {   char sign_exp;     // sign bit and 7 bits of exponent
        int  mantissa;     // lsbit of exponent and 15 msbits of mantissa
        char lower_mantissa;  // lowest 8 bits of mantissa
     }  fp_parts;
      unsigned long shiftable_fp;  // used to zero union, and to do 1-bit shift
      float ieee_fp;          // final ieee answer is here
} make_fp;
```

(Here I'm assuming that your machine stores most-significant byte in
low memory; this is true for motorola processors, but you'll
have to check for your own application).

CONVERSION ALGORITHM:

Accept the first 2 hex ascii chars and store the binary value into Qsign_var.

If Qsign_var == 00:
  accept the remaining six hex ascii chars;
    if all equal 0:
      exit the conversion routine {result = 32bit zero}.
    If the remaining 6 chars not= zero:
      declare an error (invalid FP#) and exit the conversion routine.

Acquire the next 2 hex ascii chars; these represent the 8bit QED exponent.
Store the binary value into Qexponent_var.

Acquire the next 4 hex ascii chars; these represent the 16bit QED mantissa.
Store the binary value into make_fp.fp_parts.mantissa

If Qexponent_var = 0x7F:
  Set make_fp.fp_parts.mantissa = 0xFFFF.

Else if QED exponent is in the range= 0x80 to 0x83:
  Set make_fp.fp_parts.sign_exp = 0x82.
    {Will be 0x01 after we add bias).
  Set make_fp.fp_parts.mantissa = 0x0000.

  {Note: QED exponents in range 80 to 83 map onto +/- 1/ieee.infinity}
Endif

make_fp.fp_parts.sign_exp += 0x7F;
    // add bias to exponent

make_fp.shiftable_fp >> 1;
    // shift right 1 bit position

If Qsign_var = 0xFF, shifted_IEEE_FP.sign_exp |= 0x80.
    // set sign bit.
If Qsign_var = 0x01, shifted_IEEE_FP.sign_exp &= 0x7F.
    // clear sign bit.

Now the floating point number can be referenced as make_fp.ieee_fp

DONE

```
\ *********** SOME CONVERSION CORRESPONDENCES ***********

              MS         LS       MS         LS
FP number      QED hex            IEEE hex

0.0           00 00 00 00       00 00 00 00
1.0           01 00 00 00       3F 80 00 00
2.0           01 01 00 00       40 00 00 00
-2.0          FF 01 00 00       C0 00 00 00
1.0E6         01 13 E8 48       49 74 24 00
-3.8E-10      FF E0 A1 D1       AF D0 E8 80




\ *********** HC11 ASSEMBLY CODED VERSION: ***********
\ the following 2 routines perform conversions in the QED-Forth kernel.
\ they are included for reference.
```

```
      7F CONSTANT EXP.BIAS          \ offset of exponent for ieee fp format

\ must be callable from C
CODE FP_QtoC      ( qed.fp -- ieee.fp )
\ generates normalized ieee representations:
\ sign bit in msbit (bit 31), biased exponent in next 8 bits (23-30);
\ mantissa in next 23bits (0-22).  zero is always positive.
\ maps QED fp with exponent = 7F onto +/-ieee.largest normalized number
\      (biased exp. = FF, mant=0)
\ maps QED fp with exponent in range 80-83 onto
\      ieee exponent = 1, ieee mantissa = 0.
      0 IND,Y LDD                   \ A <- sign byte; B <- exponent
      EQ IF,
            RTS                     \ if sign and exp = 0, result = 0
      ENDIF,                        \ we're counting on valid QED fp number(32bit 0)!
      2 IND,Y LDX                   \ X <- mantissa
      7F IMM CMPB                   \ exponent = 7F maps onto +/- ieee largest num
      EQ IF,                              \ ieee exponent will be FE
            FFFF IMM LDX            \ make top 16bits of mantissa=largest possible
      ELSE,
            83 IMM CMPB             \ exponent 80-83 maps onto +/- 1/ieee.infinity
            LE IF,
                  82 IMM LDAB       \ make exponent 82; ieee result will be 01
                  0 IMM LDX         \ make mantissa 0 for consistent results
            ENDIF,
      ENDIF,
      PSHA                          \ save sign
      1 IND,Y STX                   \ put mantissa into middle 2 bytes
      3 IND,Y CLR                   \ put 0 in lsbyte
      EXP.BIAS IMM ADDB
      0 IND,Y STAB                  \ exponent + bias into msbyte of scratch
      CALL -1
      CALL DSCALE                   \ shift right 1 place; msbit gets garbage
      PULA                          \ retrieve sign
      TSTA
      MI IF,                        \ if sign byte was FF...
            80 0 IND,Y BSET         \ make it negative
      ELSE,
            80 0 IND,Y BCLR         \ else make it positive
      ENDIF,
      RTS
END.CODE
```

```
\ must be callable from C
CODE FP_CtoQ      ( ieee.fp -- qed.fp )
\ converts denormalized numbers to zero; ie if biased.exp = 0, answer = 0
\ we assume input is valid ieee number; NAN (not a number) inputs are
\ converted to +/- infinity depending on sign bit.
\ lsbit of mantissa is not rounded, resulting in up to 1 lsbit error
\ during the conversion.  This is less of a problem because we
\ are using this conversion only to print strings, not as intermediate
\ steps in math operations.  All math is done in the high precision ansi format.
      0 IND,Y LDD                    \ msbit of A = sign; rest of A and msbit of B=
exponent
      PSHA
      ASLD                           \ A has biased exponent
      TSTA
      EQ IF,                         \ if biased exponent = 0, result = 0
            CLRB                     \ we're converting denormalized numbers to 0 also
            0 IND,Y STD
            2 IND,Y STD
            PULA                     \ clear sign byte off rstack
            RTS         ( 0\0 -- )   \ done
      ELSE,
            FF IMM CMPA              \ if biased exponent was FF...
            EQ IF,                   \ ieee input was a signed infinity
                  CALL FDROP         ( -- )
                  CALL INFINITY
                  PULA               \ A <- sign byte
                  TSTA
                  MI IF,             \ if sign was negative...
                        CALL FNEGATE
                  ENDIF,
                  RTS   ( +/-infinity -- ) \ done
            ENDIF,
      ENDIF,                   \ if we're here, input is a nonzero normalized number
      2 IND,Y LDX              \ X <- lowest 16bits of mantissa;B has other 7 bits
      MI IF,                   \ if msbit of X is set...
            01 IMM ORAB        \ set lsbit of B; B now has top 8 bits of mantissa
      ENDIF,
      EXP.BIAS IMM SUBA        \ unbias the exponent
      1 IND,Y STAA             \ put unbiased exponent to dstack
      PULA                     \ A <- original sign byte
      TSTA
      MI IF,
            FF IMM LDAA
      ELSE,
            01 IMM LDAA
      ENDIF,
      0 IND,Y STAA             \ put sign byte
      XGDX
      ASLD                     \ A has lower 8 bits of QED mantissa
      PSHA
      XGDX                     \ B has top 8 bits of QED mantissa
      2 IND,Y STAB
      PULA
      3 IND,Y STAA             \ put mantissa to stack
      RTS         ( fp -- )    \ done
END.CODE
```

```
// Copyright 1997 Mosaic Industries, Inc.  All Rights Reserved.
//  Disclaimer: THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, WITHOUT ANY
//  WARRANTIES OR REPRESENTATIONS EXPRESS OR IMPLIED, INCLUDING, BUT NOT
//  LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS
//  FOR A PARTICULAR PURPOSE.
```


Mosaic Industries

5437 Central Ave Suite 1, Newark, CA 94560 Telephone: (510) 790-8222 Fax: (510) 790-0925