Subject: VAX floats to IEEE (source code avail.)
Posted by gotwols on Tue, 01 Nov 1994 22:57:00 GMT
View Forum Message <> Reply to Message

There has apparently been some discussion in this newsgroup about
code to convert from VAX floats (REAL*4) to IEE format on various
platforms. Unfortunately all of the items were expired when I got
around to reading them so please forgive me if this is redundant.
   I have written a routine in C that does the job well on a number
of platforms. My routine (vaxf2ieee.c appended below) may not be the
most efficient in the world, but I think it is accurate. I spent
quite a bit of time handling the very small numbers that give trouble
when converting from the VAX number set to the IEEE. This is the domain
of unnormalized fractions in the IEEE world which has no analogy in the
older VAX world. If you don't care about these admittedly improbable
numbers then your code can be a lot simpler.
   Routine vaxf2ieee.c runs well on MAC, PC, HP, and SUN. I also wrote
two wrapper routines that allow it to be hooked into IDL using the
call_external mechanism. Routine vaxf2ieee_w.c is a wrapper in C that
just converts the calling parameters from the damn argc, argv that call_external
uses, to the more intuitive interface of the workhorse routine. Routine
vaxf2ieee.pro is a wrapper routine written in IDL that calls the C wrapper
routine. Why two wrapper routines? Because I wanted a very general
routine in C to do the job that used intuitively obvious variables, not the
totally obscure argc, argv interface demanded by call_external. While
my C routines ought to port easily to almost any machine, my IDL wrapper
is written specifically for use under HPUX and is hardwired for my
particular use. Conversion to other machines and users ought not to be
too difficult I hope. If anyone ports this code to the PC or MAC under
IDL, I would appreciate receiving a copy of the appropriate wrapper code.
   The routines are appended below.

```
***************** This is the IDL wrapper written for HPUX ***************
; vaxf2ieee.pro  Convert VAX floating point format to IEEE format.
;        blg 1/6/93
; 4/10/94 Changed to do the conversion in place.  Name changed to
;  vaxf2ieee.pro

; NOTE!!! On some machines like the PC and MAC it may be necessary to
; read in the x variable as a long integer.  This is because certain
; perfectly valid VAX numbers will appear to the IEEE machine as
; "Not a Number" (NAN) before they are converted, and the operating
; system may trap them and change them before they can be converted.
; I doubt this will happen if they are just read in from disk into a
; float array but I have not tested it.  A certain way to avoid
; trouble is to let x be a long int and after the conversion is done
; convert to a float by the statement:
;   y = float(x,0,n_elements(x))
; Needs to be linked to the library in the standard place.

        pro vaxf2ieee, x, error=err
 err = 0

; Do a bunch of error checking:
 nx = N_ELEMENTS(x)
 szx = size(x)
 xtyp = szx(szx(0)+1)
 if xtyp le 2 or xtyp ge 7 or xtyp eq 5 then begin ; Not supported
     print,'vaxf2ieee:  Input variable must be float, long or complex'
     err = 1
     return
 endif
 if xtyp eq 6 then nx = 2*nx ; Complex has twice as many

 err = call_external('/users/gotwols/idl/c/vaxf2ieee_w.sl',$
     'vaxf2ieee_w', x, x, nx)

        return
        end

********************END OF vaxf2iee.c ***********************************

**************** The C wrapper follows:*****************************
/* vaxf2ieee_w.c   A wrapper program that interfaces between IDL routine
vaxf2ieee.pro and C routine vaxf2iee.c.  The routine here merely converts the
nasty argc, argv calling convention of call_external to the arguments wanted
by vaxf2iee.c.
B. L. Gotwols  6-jan-94
               10-apr-94   Now returns 0.
*/

/* Prototype:   */
void vaxf2iee(long *x_vax, long *x_ieee, long n);
int i;

int vaxf2ieee_w(int argc, char *argv[])
    {
    long *x_vax, *x_ieee, n;
    x_vax = (long *)argv[0];
    x_ieee = (long *)argv[1];
    n = *(long *)argv[2];
    vaxf2ieee(x_vax, x_ieee, n);
    return 0;                    /* Can't make an error  */
    }
```

```
***********************END of vaxf2ieee_w.c  (C wrapper program)   *********

*********** The workhorse C routine follows:*******************************

/*  Convert vax floating point numbers to IEEE format (HP, SUN, PC, etc.).
    The byte order of the host machine will be determined at run time.
    Note that since the VAX floats do not map exactly onto the IEEE number
    space there can be a minor difference for the smallest numbers.
    In the code that follows I tried to be sure that every valid VAX floating
    point number would be converted to it's nearest IEEE equivalent in the
    byte order of the machine that this routine is being run on.  In the
    huge majority of cases all that is necessary is to do the appropriate
    byte swaps and subtract two from the biased exponent. This is equivalent
    to simply dividing the byte swapped float by floating number 4.0.  Indeed
    I have seen some code obtained off the network that does the conversion
    exactly this way.  However, the problem with this simplistic algorithm is
    that it gets the wrong answer for numbers larger than approximately
    2^127 and smaller than 2^-126. I would rather have the code go a little
    slower but maintain accuracy, so I put a lot of the effort into handling
    ALL of the VAX numbers no matter how improbable they are.  This should
    explain why much of the code is devoted to handling numbers below 2^-126
    where the IEEE numbers become denormalized and thus a lot of manipulation,
    including rounding, is needed.

    This code has been tested on the PC under Micro Soft C v5.1, the VAX
    under VMS, the HP9000/735 under HPUX 9.01, and the Macintosh under
    MPW v3.3a14.


    blg  1-jan-1994
*/

#include <stdio.h>
#include <math.h>

void vaxf2ieee(long xin[], long xout[], int n)

    {
    unsigned short endian = 1;   /* Use to test endian nature of this machine */
    unsigned long frac;
    unsigned int expon;
    int rem;  /* Remainder after shifting right 1 or two bits */
    int i, v0, v1, v2, v3;
    long int xl;
    union {
      long int l;
      unsigned char b[4];
    }y,z,tmp;

/*  Determine the byte order of this machine:               */
   if(*((char*)&endian) == 1)
      {     /* Little endian:  eg. VAX or PC */
      v0 = 2;  /* v0 is the index where the 0th VAX byte gets moved to */
      v1 = 3;  /* Similar for v1, etc.  */
      v2 = 0;  /* These are just short int swaps  */
      v3 = 1;
      }
   else
      {     /* Big endian: eg. HP, SUN, IBM RISC, SGI */
      v0 = 1;  /* These are adjacent byte pair swaps */
      v1 = 0;
      v2 = 3;
      v3 = 2;
      }

   for(i=0; i<n; i++)
      {
      y.l = *(xin+i);              /* Will pick y apart */
      z.b[v0] = y.b[0];    /* Do the necessary byte swaps */
      z.b[v1] = y.b[1];
      z.b[v2] = y.b[2];
      z.b[v3] = y.b[3];
      tmp.l = (z.l<<1);    /* Temporary storage for the exponent */
      expon = tmp.b[v1];
      if(expon >2)         /* The most common case by far so put it first */
         {
         xl = z.l - 0x01000000;   /* Sneaky. It subtracts 2 from expon.  */
         }
      else if(expon == 0)  /* Next most common will be 0.0  */
         {
         xl = 0; /* 0 expon should be all 0 bits. (neg. 0 traps on VAX ) */
         }
      else       /* Exponents of 1 and 2 land here. (Denormalized nums.)*/
         {
         frac = z.l & 0x7fffff;
         if((expon == 2) && (frac == 0x7fffff))  /* A very very spec. case */
            {
            xl = (z.l & 0x80000000) | 0x800000;  /* Round up to 2^-125 */
            xout[i] = xl;
            continue;
            }
         tmp.l=(expon==2)?((frac&0x1)<<7):((frac&0x3)<<6);/* Left justify*/
         rem = tmp.b[v2];  /* remainder  */
         frac = (frac | 0x800000)>>(3-expon); /* Or in hidden bit and shift*/
         if((rem>128) || ((rem == 128) && (frac & 0x01))) frac++; /* Round */
```

```
            xl = (z.l & 0x80000000) | frac;  /* (Exponent will be 0)  */
            }
        xout[i] = xl;
        } /* End of for loop  */

    return;
    }
```

****************END of vaxf2ieee.c (The work horse program) *****************

--

Bruce L. Gotwols
Johns Hopkins University, Applied Physics Lab., Laurel MD 20723
Internet: gotwols@tesla.jhuapl.edu   (128.244.147.15)
SPAN: APLSP::STR::GOTWOLS