PDP-11

DESIGN NOTE 11Ø3 ØØ

December 19, 1969

DESIGN NOTE: FLOATING POINT FORMATS

FROM: Bruce Delagi

REVIEWED BY:

_____

11-Engineering

_____

11-Software

_____

K-Project

_____

15-Engineering

ABSTRACT: Binary and hex normalization, various 32, 48 and 64 bit formats, and the formats used by the competition in the larger machine market are discussed. A recommendation on the formats to choose for alternate marketing strategies is made.

Ø.Ø Scope - This design note describes several alternative floating point formats and presents a rationale for the choice made for the 11. The alternatives may be classed according to:

A. The number of bits representing the mantissa.

B. The number of bits representing the exponent.

C. The normalization radix (e.g. binary, octal, hex).

1.Ø Normalization Radix - The choice of normalization radix permits trade off of exponent range against mantissa precision. The truncation/rounding error for the mantissa is considered to be 1 digit of the normalization radix. Thus the error for binary normalization is 1 bit; in octal, 3 bity; in hex, 4 bits. The exponent indicates powers of the normalization radix, so the range in powers of ten is tripled for octal normalization and quadrupled for hex normalization with respect to what they are for any given length exponent field with binary normalization.

Moving two bits from the mantissa to the exponent field also quadruples the exponent range - and only 2 bits (not 3 as in hex normalization) are lost from the mantissa precision. The principal advantage in hex normalization is that it permits delimiting the mantissa - exponent field on a byte boundary. For the 11, this would permit the exponent byte to be separately manipulated from the mantissa. This is useful primarily in scaling, fixing, floating, and I/O conversion. The disadvantages in hex normalization are that:

1. It is hard to emmulate in software (normalization in particular is messy).

2. Precision is lost more rapidly than range is enhanced.

1.1 Emulation of Hex Normalization - A number is hex normalized if one of the most significant four bits of the fraction is different from the sign bit (but - Ø.5 is represented as 1.1111Ø...Ø). A sample hex normalization routine might be as follows:

```
        ;routine normalizes (RØ, R1) to base 16, leaves count in R2

NORM16: MOV #8,,R2
        MOV #174ØØØ, R3
LOOP16: BIT  RS, R1          ;see if top 5 bits all Ø
        BEQ  MORE16          ;if yes, continue normalizatio
        BPL  DONE16 +2       ;if sign positive (Ø) and one
                              of the other bits=Ø, done
        COM  R1              ;if sign negative, test for
                              all other 1
        BIT  R3, R1          ;see if 5 top bits all 1
        BNE  DONE 16         ;if not, normalization
                              complete
        COM  R1              ;if so, check for -½
        BIT  #3777/, R1      ;if all other bits = Ø,
                              normalization complete
        BNE  MORE16
        TST  R2
        BEQ  DONE 16
MORE16: ASL  RØ              ;shift 4 places, try next digi
        ROL  R1
        ASL  RØ
        ROL  R1
        ASL  RØ
        ROL  R1
        ASL  RØ
        ROL  R1
        DEC  R2
        BNE  LOOP 16         ;check loop connt
        RTS  PC

DONE 16: COM  R1
        RTS  PC
```

From the above example and the binary normalization example in the PDP-11 Handbook we see that the ratio of hex normalization time to binary normalization time for the 11 without EAE is roughly:

$$r_{16/2} = \frac{16}{.5}\left\{\frac{\frac{n}{4}+1}{n+1}\right\}$$

Where n is the number of binary digits shifted.

Thus $\sqrt{16/2 + \frac{4}{5}\ \frac{(n+4)}{(n+1)}}$ so that for large shifts, hex normalization is actually faster (by a factor of Ø.8Ø) than binary normalization. Large normalization shifts are common in I/O conversions (lots of fixing and floating) but are uncommon in arithmetic operations for well written programs (large normalization shifts cause wholesale loss of significance). In multiply and divide operations, post normalization is at most 1 place so that hex normalization may be twice as slow as binary.

1.2 1Ø Bit Exponent Fields - An alternative in trading exponent range against precision is to directly move some number of bits (2) from the mantissa to the exponent field. Thus binary normalization is maintained but other problems are introduced. These problems may be broken into two classes: emulation difficulties and difficulties in the floating point machine in handling a 1Ø bit quantity for exponent manipulation.

The emulation is facilitated by:

1. Not requiring the emulator to make use of the additional mantissa bits,

and 2. Right justification of the exponent within the same word as the low order bits of the mantissa.

With these restrictions, exponent handling in emulation is as easily done as if nicely aligned on a byte boundary. Checking exponent overflow and underflow is done simply with CMP's.

In the floating point machine, several instructions can be added to specifically facilitate exponent handling (e.g. fixing and floating instructions) Scaling can be done by full multiplication.

2.Ø Signed vs Excess Code Exponents - The advantage of excess code is that floating Ø can be represented with Ø exponent without causing loss of precision in operations involving addition or subtraction of Ø. Thus, an array can be filled with normalized floating point Ø's by repeatedly executing an instruction of the form "CLR (R)+." Signed code allows exponent arithmetic in emulations to be done a little more easily (In multiplication and division an additional ADD #N, EXP is required with excess code to readjust the exponent).

The advantages here seem to lie on the side of excess code notation.

3.Ø Choice of Representation Length - The 11's entry into the large processor end of its market will be greatly enhanced by arithmetic power competitive with SEL's SYSTEM 86 XDS' SIGMA 5 and SIGMA 7, and IBM's 36Ø series. Each of these machines offer two floating point formats. The short format is a 32 bit representation and offers 5.7 (6.Ø) digits precision with a ± 153 (76) exponent range (SYSTEM 86 specs in parenthesis). The long format is a 64 bit representation with 15.3 (15.6) digits precision and a ± 153 (76) exponent range.

Several alternatives fro a 64 bit floating format can be considered:

1. 16 bit exponent, 48 bit mantissa, binary normalization, 13.8 digits, ± 96ØØ exp;

2. 18 bit exponent, 56 bit mantissa, binary normalization, 16.2 digits, ± 38 exp;

3. 10 bit exponent, 54 bit mantissa, binary normalization, 15.6 digits, ± 153 exp.

Hex normalization is not considered because of its speed disadvantage in emulation for arithmetically orientated routines.

Note that alternative 1, lacks sufficient precision to be competitive, that alternative 2 has insufficient range, and that alternative 3 outperforms all competitors.

Thus the best 64 bit floating point representation for the 11 seems to be binary normalization with a 1Ø bit exponent and a 54 bit mantissa.

3.1 Short Format Floating Point - The observable factors in the market bearing on short format seem to be these:

1. IBM's short format (5.7 digits precision) seems to be recognized as of very limited use in that class machine.

2. Most users of FOCAL on the PDP-8 use the 3 work package (6.6 digits instead of the four word 1Ø.2 digits).

3. Most of the small 16-bitters use a 24 bit mantissa with binary normalization (6.6 digits, ± 38 range).

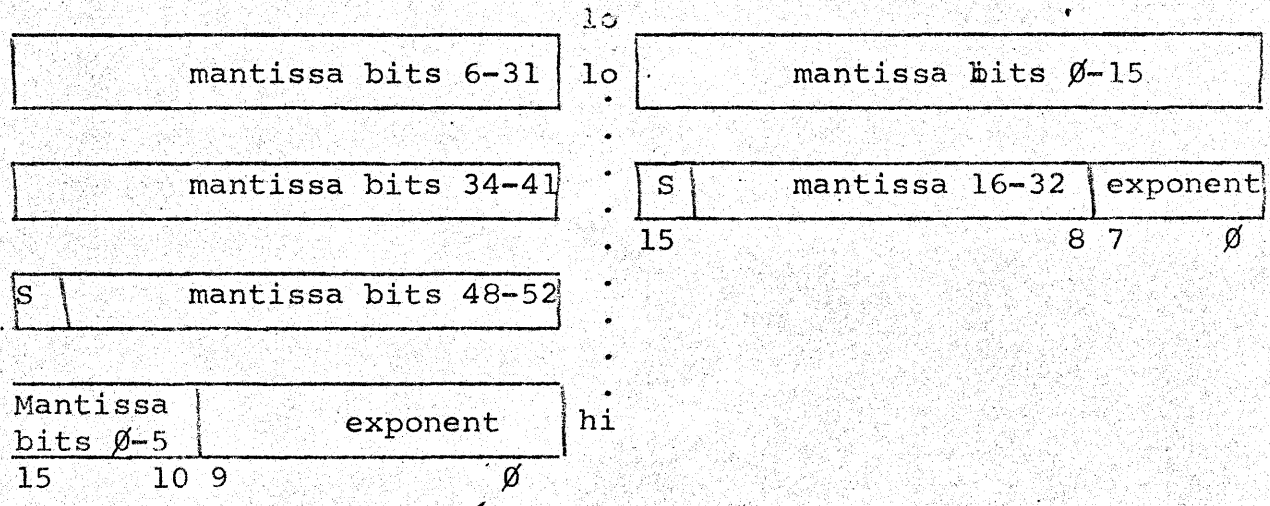
I think this argues that a 24-bit binary normalization mantissa (6.6 digits) is probably sufficient for some reasonable size market segment. This is a 2 16-bit word representation and effectively increases the variable storage fro a program like FOCAL or BASIC, 20% over that with a 3 word representation.

It seems to me that we will get beat on in both the large machine and small machine sectors of the 11's market by not having both a short, low precision, limited range and a long, high precision, extended range format.

Further, choosing formats that are multiples of a 32 may offer some advantage in a 32 bit processor implementation of the 11.

With these points in mind, it seems that a short format with 8 bits of exponent, 24 bits of mantissa, and binary normalization will be desireable (6.6 digits, ± 38 range).

Note, however, that as shown in DN-11ØI 29 bits mantissa plus sign (8.4 digits) is just as easy to emmulate as 24 bits on a 16 bit machine with EAE. Also if the short and long formats are of different formats, instructions will have to be provided to convert between them:



(word order is as discussed in DN-11Ø2)

3.2 48 Bit Format - If we can compete in the larger machine market without 15 digit precision and ± 15Ø exponent range, a single 48 bit format may suffice for all floating point operations.

Three formats will be considered:

1. 16 bit exponent, 32 bit mantissa, binary normalization, 9.Ø digits precision, ± 96ØØ range.

2. 8 bit exponent, 4Ø bit mantissa, binary normalization, 11.4 digits precision, ± 38 range.

3. 10 bit exponent, 38 bit mantissa, binary normalization, 1Ø.8 digits precision, ± 153 range.

Of these, I favor format 2 since 9 digits seems to me awfully small when compared with 15 of the competition (7.8 was after all a handicap to the 1Ø) and the awkardness of 1Ø bit exponents doesn't seem to me worth the extended range that is almost that of 3 -- given that we need not compete point for point with the 64 bit format machines.

4.Ø Conclusions:

Hex normalization does not appear to offer any important advantages over binary (unless most floating operations are "fixing"and "floating"). The decision on whether we can or need to compete with SYSTEMS 86, SIGMA 7 et al with a 48 bit format will need to be made by marketing. If we require greater precision and range to compete, then we should use the 1Ø bit exponent, 54 bit mantissa format. If we choose a 64-bit format, we will need a short format as well; 48 bits, however, seems awfully close to 64 so I would recommend an 8-bit exponent, 24-bit mantissa as the short format.