PDP-11/40 Technical Memorandum # 16 A

Title:	PDP-11 Floating Point Format
Author(s):	Hank Spencer
Index Keys:	Floating Point Formats Exponent Base Radix Point Conversion
Distribution:	PDP-11 Coordinating Committee PDP-11/40 Group
Revision	1
Obsolete:	None
Date:	November 10, 1970

0.0 ABSTRACT

A floating point format has been adopted for PDP-11 computers which features an exponent range of $10^{\pm 38}$ and a fraction in sign-magnitude form, with 24 bits of precision in the single-precision (2 word) form and 56 bits in the double-precision (4 word) form. The following documents are attached as appendixes, to show the history and technical justification for this format:

PDP-11 Floating Point Format	Oct. 1, 1970
Minutes of PDP-11 Floating Point Format Meeting Held 30 Sept. 1970	
Latest Proposal for PDP-11 Floating Point Format	Sept. 15, 1970
Minutes of PDP-11 Floating Point Package Meeting	Sept. 8, 1970
Objections to 360-Floating Point Format	Aug. 25, 1970
PDP-11 Floating Point Format (TM #16 Rev Ø)	Aug. 21, 1970

digital INTEROFFICE MEMORANDUM

SUBJECT: PDP-11 Floating Point Format

DATE: October 1, 1970

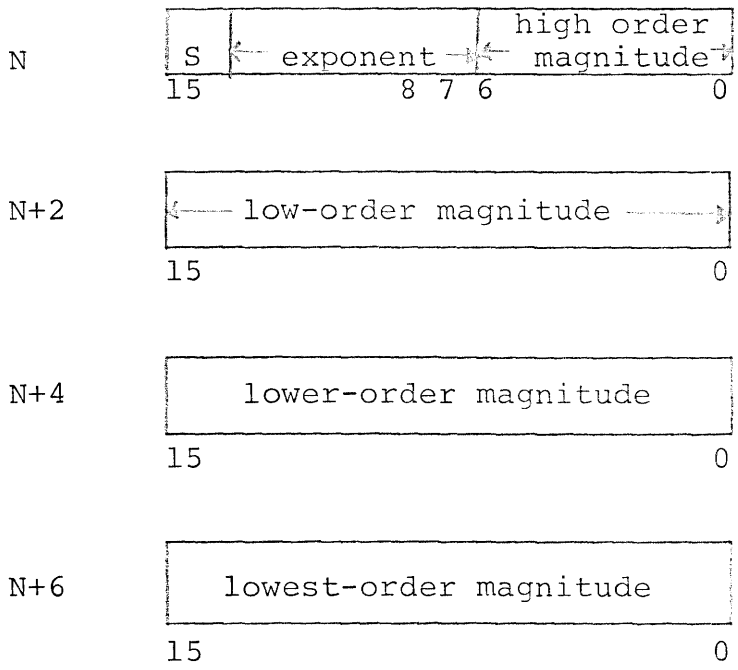
TO: PDP-11 List C
PDP-11 Master List

FROM: Hank Spencer

DEPARTMENT: Programming

Yesterday's meeting on this subject agreed to accept the floating point format shown below as appropriate for the entire PDP-11 line. Possible alternatives, the rationale for choosing this one, and its shortcomings are covered in detail in PDP-11/40 Technical Memorandum #16. This format will be implemented in software in the PDP-11/20, as a floating point package used in the Fortran Object Time System, and in hardware in the PDP-11/40.

The format:



S = sign of fractional portion

Magnitude = size of fraction, unsigned

exponent = binary exponent, excess 128_{10}

sign-magnitude form, binary normalization

Because we limit ourselves to normalized numbers, the highest order bit of the fraction magnitude is always 1, therefore, it is not represented in this format. Thus the single precision form has effectively 24 bits of precision, the double-precision form has 56 bits.

PDP-11/40 Technical Memorandum # 16

Title: PDP-11 Floating Point Format

Author(s): Ad van de Goor
Jim Murphy
Hank Spencer

Index Keys: Floating Point Formats
Exponent Base
Radix Point
Conversion

Distribution: PDP-11 Coordinating Committee
PDP-11/40 Group

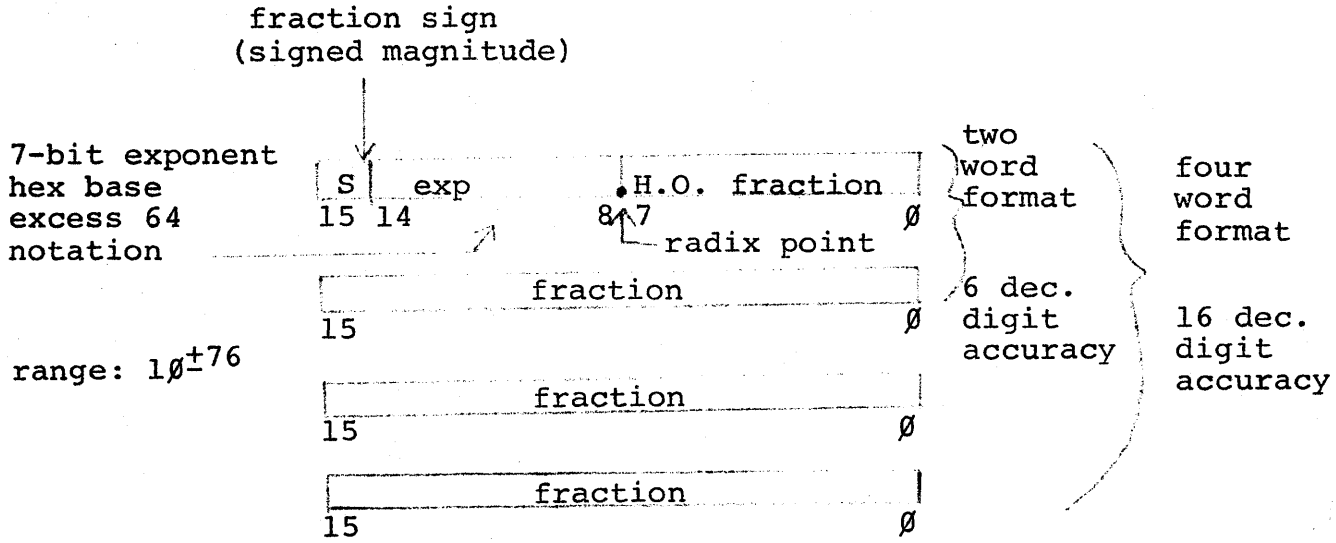
Revision: None

Obsolete: None

Date: August 21, 1970

0.0 ABSTRACT

The IBM 36Ø two and four word (i.e. 32 and 64-bit) floating point formats were selected for the PDP-11 family. These formats consist of a signed 7-bit exponent and a signed 25 or 57 bit fraction with the quantity expressed by the number being the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess 64 binary notation; the fraction is expressed as a hexadecimal number having a radix point to the left of the high-order digit.



This format was chosen because it satisfied more of the requirements of a "good" format than any of the alternatives studied. It

1) provides an exponent range of $1Ø^{\pm 76}$, adequate for the majority of users

2) provides 6 decimal digits of precision with its short (32-bit) format, adequate for most single precision uses on small machines

3) provides compatibility with a major market segment

4) provides convenience of keeping to byte boundaries for ease of software simulation

5) provides convenient way of going to double precision, no special cases or operations required

6) provides sign, exponent and high-order fraction in one word, simplifying both hardware and software operations of tests and comparisons

7) provides sign magnitude fraction notation which is mathematically superior to other forms and is easier to manipulate in software simulation

8) provides excess exponent notation which allows for the representation of Ø with the smallest possible exponent

9) provides hexadecimal based exponent which appears to be faster to manipulate in software on the average (normalization and alignment take longer but the probability of their need is much less) and is certainly faster in hardware

10) provides radix point at the left (fractional representation) which is more traditional, more widely understood and no reason appears to exist for integer representation on the PDP-11

11) provides formats that will allow some 32-bit integer and floating point instructions to be shared.

Section 1 of this memo is concerned with the basic considerations of floating point formats and contains a set of requirements a "good" format has to satisfy.

Section 2 shows a set of possible solutions and explains why the IBM 360 format was selected.

Section 3 has a list of references together with the alignment and normalization statistics as published in Sweeney's article.

1.0 Basic Considerations

Some basic considerations which will assist in the selection of the floating point format are discussed in this section.

1.1 The number of words (Note: words are 16 bits long).

Looking at the needs of the customers it can be concluded that more than one floating point format is needed.

A short format is needed where precision is not of prime importance and where speed and/or storage space requirements are dominant (the majority of the FOCAL and BASIC users belong to this group).

A long format is needed there where a high accuracy is the main concern, e.g. inner loops in a matrix inversion routine.

The current PDP-11/20 format has to be considered a compromise between the above two format requirements.

Reasons for having a two and four word format

1) A 32-bit implementation of a PDP-11, as the PDP-11/60 is supposed to be, can make very efficient use of a two/four word format. One/two memory cycles are needed to store or retrieve the data into or from memory. This compared with a 3 word format where always two memory cycles are needed.

2) Indexing can be accomplished more easily because multiplying the index quantity with a power of two can be done through simple shifts. Any format whose lengths are not a power of two requires some form of multiplication of the index quantity.

3) Most competitors offer a two and a four word format.

1.2 Exponent Range

The lower limit of the exponent range can be determined quite easily. Looking at some well known constants, e.g. Avogadro's number N=6.0255*10↑23 and Planck's constant h=1.0545*10↑-27 it is quite clear that an exponent range of $10\uparrow -38 \leq exp \leq 10\uparrow 38$ is a minimum requirement.

The consensus of the marketing inputs was that an exponent range of $10\uparrow -76 < exp < 10\uparrow 76$ was very desirable in order to be competitive as well as for scientific reasons.

Another point of importance is that by having the exponent range big enough for the short as well as the long floating point format, conversions between the short and long formats are made more easy.

1.3 Layout of Formats

A very classical layout is shown in Figure-1. The first bit is the sign of the fraction, the next group of bits is the exponent (including its sign) followed by the fraction

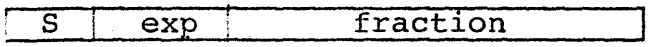


Figure-1. Classical Layout

The main advantage of the above layout is that certain instructions do not have to differentiate between a floating point number and an integer number of the same size, which leads to a savings in hardware. Typical instructions which can be shared are COMPARE, LOAD, STORE, etc.

A few requirements have to be made of the layout of the formats with respect to execution speed of certain instructions.

1.3.1 Easy conversion between single and double precision.

In most formats, double precision is formed by appending two words (i.e. 32 bits) at the end (i.e. least significant part) of the single precision format. Going from double to single precision merely means deleting the two least significant words assuming the floating point number was normalized and the exponent range is the same for the short and long formats.

The IBM 1800 format, as shown in Figure-2 below, is an example where conversion to a double precision format would cause difficulties because a 32-bit word (assuming a 4-word double precision format) would have to be inserted between the lowest bit of the fraction and the sign of the exponent.

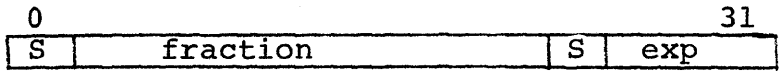


Figure-2. IBM 1800 Single Precision Format

1.3.2 Fast Test and Compare Instructions

This requires that most of the relevant information necessary for testing and comparing is stored in a single, preferably the first word of the format. This information is the sign and the first few high bits of the fraction and the exponent.

The IBM 1800 format of Figure-2 is an example wherein compare instructions always both words have to be read from memory.

1.4 Position of the Radix Point

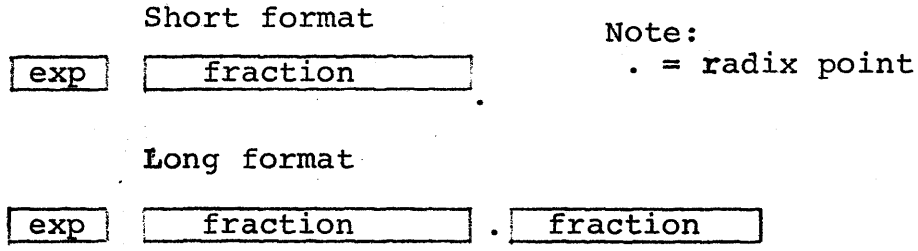
The more standard way of representing floating point numbers is by having the radix point to the left (i.e. .X). Some machines (e.g. Burroughs B5000/B5500/B6500/ B7500 and the CDC STAR) have the radix point to the right (i.e. X.). This allows integers to be a subset of unnormalized floating point numbers.

It should be noted that having the radix point at the right makes the range of floating point numbers asymmetric with a bias towards large integers which may be less desirable.

The only reason we know of, for having the radix point to the right is the possibility of treating integers as floating point numbers. In order to allow for this unnormalized floating point arithmetic has to be used (like in the STAR) or the integer case has to treated differently while the standard mode is normalized arithmetic (like in the Burroughs machines). The latter leads to a more complicated floating point unit.

Some interesting problems arise when going from single to double precision which can be done in two ways.

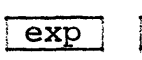
1) The Burroughs method



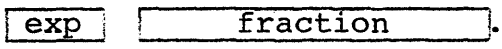
The advantage of going from short to long format this way is that the exponent does not have to be adjusted. Disadvantage: the radix point is in the middle which excludes long integers.

2) The CDC STAR method

short format



long format



This method allows for long integers (the STAR has 24 and 48-bit integers) at the expense of having to adjust the exponent upon conversion between formats.

Considering the existence of 16-bit integers in the current PDP-11 architecture it is questionable whether the addition of a 24-bit integer (obtained by having the radix point at the right) would be of any use. Considering that or the floating point unit has to be more complex or unnormalized arithmetic has to be done we would like to drop this issue from further consideration.

1.5 Representations for Indefinite and Infinity.

The CDC STAR computer allows for representations of infinity (and indefinite through the condition code) together with computation rules on these quantities. The advantage is that a program run does not have to be suspended upon the occurrence of infinity or indefinite such that other "better behaving" sections of the program still can produce meaningful results. Also interrupts can be avoided this way. This comes at the expense of more complex hardware (and software in case of a software floating point package), however, because the operations +, -, * and / have to be defined for regular numbers, indefinite and infinity.

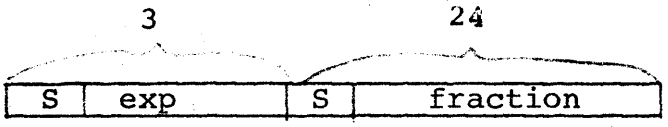
The effect of indefinite and infinity can be simulated, to a large extent, by trapping the occurrence of indefinite and infinity and allowing for this trap to be passed to the user. PDP-10 ALGOL will have a feature like this.

The introduction of indefinite and infinity introduces several special cases which will make floating point arithmetic more complicated, also the reasons why STAR and the CDC 6600 have such features do not apply to the PDP-11 family. Therefore, the above features are not considered necessary.

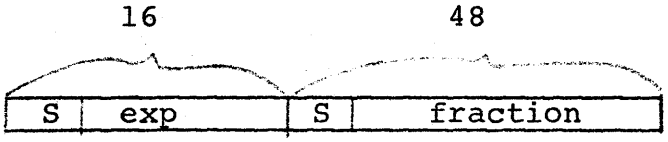
2.0 Possible Floating Point Formats

Four floating point formats are considered in this section. Each format has a short (32-bit) and a long (64-bit) version.

2.1 The CDC STAR Format



Short format



Long format

The radix point is to the right. The exponent and the fraction are 2's complement numbers. The normalization is binary.

Disadvantages:

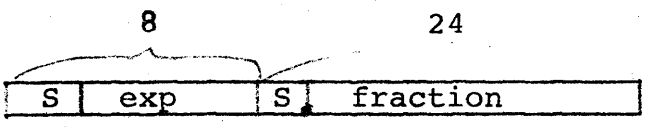
1) The exponent range of the short format is not very big.

2) Because of the 2's complement notation for the exponent and their representation of zero and infinity, software simulation will be costly.

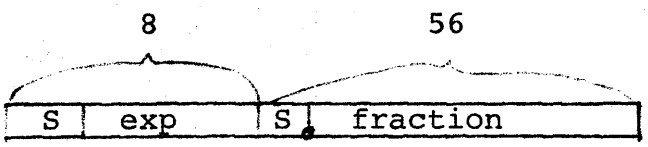
3) The 16 bit exponent in the long format will be objectionable for customers because they rather have more significant digits then such an unusual large exponent range. The PDP-10I's long format is different from that of the PDP-10 because customers wanted the extra bits where a dummy exponent was stored to be used for more precision.

4) Conversion between the single and double precision formats requires exponent adjustments.

2.2 The Byte-Exponent Format



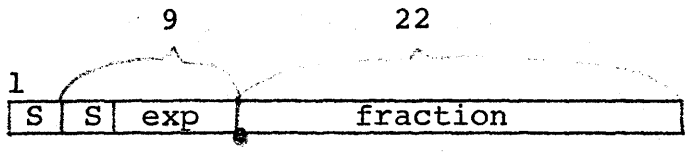
Short format



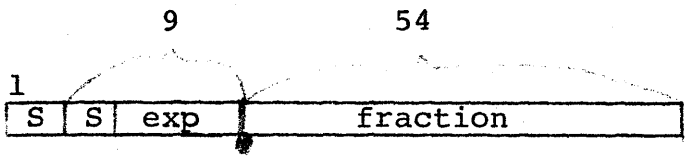
Long format

In order to get the required exponent range with the above format, quad normalization (i.e. radix = 4) has to be used. Considering the PDP-11/20 instruction set quad normalization is practically as difficult as hex normalization. Furthermore, by not having the classical layout of Figure-1 the economy of sharing some 32-bit integer with 32-bit floating point instructions is lost.

2.3 The Binary Exponent Format



Short format



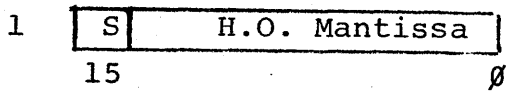
Long format

Because of the 9-bit exponent binary normalization can be used to get the required exponent range. Considering the PDP-11/20's byte oriented instruction set, packing and unpacking of this format is more difficult.

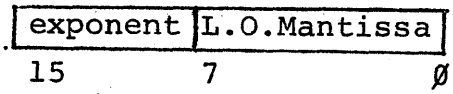
Keeping to byte boundaries:

This section illustrates the penalties (time and core) in going to greater than byte sized exponent fields. The problems occur mostly in the loading and storing operations in the floating point package.

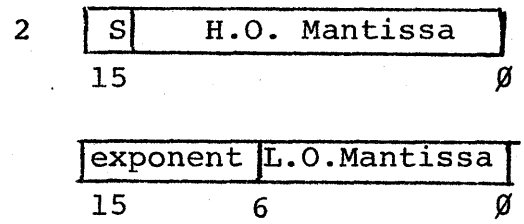
Let's take two hypothetical formats and compare the code necessary to break them down for processing:



8 bit exponent




```
                MOV   (R1)+,R4          ;H.O. Mantissa
                MOV   @R1,  R5
                MOV    R5,  R1
3 words     ⎧   CLRB   R1               ;exponent
6.9 usecs   ⎨   SWAB   R5
            ⎩   CLRB   R5               ;L.O. Mantissa
```



9 bit exponent

```
                MOV   (R1)+, R4          ;H.O. Mantissa
                MOV   @R1,   R5
                MOV    R5,   R1
6 words         BIC   #177, R1           ;exponent - L.O. Mantissa bits
13.4 usecs      SWAB   R5
                ROL    R5
                BIC   #777, R5           ;L.O. Mantissa - exponent bits
```

2.4 The IBM 360 Format

This format is shown in section Ø. Because so many people are concerned about hex normalization due to the claimed loss of accuracy, this point deserves some attention.

The 360 single precision format, as shown in section Ø, has 24 bits of mantissa not including the sign bit. Because of hex normalization, however, a normalized number can have the three leading mantissa bits equal to "000" such that the worst case precision of the mantissa for normalized numbers is 24-3 =21 bits, which is more than 6 decimal digits.

Section 2.3 shows a floating point format with binary normalization and the same exponent range as the 360 format. The format of section 2.3 has 22 bits of mantissa, not including the sign bit.

So the worst case precision for normalized numbers, assuming the format of section 2.3, is 22 bits. This is only one more bit as that for the IBM 360 format and also only 6 decimal digits.

Considering the normalization and alignment statistics of Sweeney's article the loss of significance because of hex normalization and alignment is lessened because of its lower probability of occurrence.

The total number of normalized floating point numbers for the IBM 360 format is:

(2 for positive and negative)*(number of possible exponents)*(number of hex normalized numbers)=
2*2↑7*2↑20*(1+2+4+8)=15*2↑28

The total number of normalized floating point numbers for the format of section 2.3 is:

2*2↑9*2↑21=2↑31=8*2↑28

From the above it can be seen that although the 360 worst case precision is one bit less, it allows almost twice as many numbers to be expressed in normalized form.

People from the Lawrence Radiation Laboratory in Livermore, Calif. came to the conclusion that the differences between hex and binary normalization were minimal, which supports our argument.

It should be noted that the IBM 360 format is quite well known; it is not only supported by IBM but also by RCA (Spectra Series), XDS (on the Sigma 5/7 in a slightly modified way) and SEL's System 86.

Probability of less normalization and alignment shifts with non-binary based exponents:

When a non-binary based exponent format is used, software simulation is slightly more costly corewise and also slower in the normalization and alignment processes. The core cost does not appear to be prohibitive and if these procedures are centralized will be an insignificant part of the total package.

With regards to execution time, one must look beyond comparing the time for binary normalization (alignment) versus the time for non-binary normalization. One must consider the probability of having to normalize or align with the different exponent bases. D. W. Sweeney in his article shows statistically that there will be considerable less need to normalize (align) as one increases the exponent base.

Referencing the table from D. W. Sweeney's article, let us look at and compare the time requirements of binary and hexadecimal normalization:

Binary:

> The mantissa to be normalized is in 2's complement form, thus as soon as bits 15 and 14 of the high order mantissa (R2) word differ the number is normalized. Negative power of 2, ($2^{\uparrow}N$), are special cases.

```
M.NORL:  DEC  R3          ;decrement binary exponent
M.NOR2:  ASL  R4          ;shift low order fraction
         ROL  R2          ;shift high order fraction
         BVC  M.NORL      ;not normalized if C and bit
                          ;15 is alike
         BCC  M.NOR3      ;special -(2↑N) check
         TST  R2
         BNE  M.NOR4      ;branch if not -(2↑N)
         SEC              ;restore to 14ØØØØ
         ROR  R2
         INC  R5          ;to avoid overflow problem
         INC  R3
M.NOR4:  SEC              ;restore negative sign
M.NOR3:  ROR  R2          ;C to bit 15 (sign)
         ROR  R4
M.RET2:
```

execution time:
a) positive number 14.4+n(9.5) usecs
b) negative number 20.8+n(9.5) usecs
c) -(2↑N) 29.2+n(9.5) usecs

where n = number of bit shifts needed to normalize the number

Hexadecimal:

This method assumes either

1) signed magnitude representation of the mantissa

or

2) 2's complementing negative numbers prior to normalization

```
M.NOR2:  BIT #74ØØØ,R2    ;HEX -- 74ØØØ
         BNE M.RET2       ;NORMALIZED
         ASL R4           ;LOW ORDER FRACTION
         ROL R2           ;HIGH ORDER FRACTION
         ASL R4
         ROL R2

         ASL R4
         ROL R2
         ASL R4
         ROL R2
         DEC R3           ;DECREMENT EXPONENT
         BR  M.NOR2
```

Execution Time: 7.Ø+n(3Ø.3)usecs

where n = number of field shifts required to normalize the number

One can see that the critical loop times differ considerably (9.5 usecs and 3Ø.3 usecs); however, these times are misleading for two reasons:

1) 9.5 is a bit coefficient and 3Ø.3 is a field (4 bits) coefficient.

2) Probability of not having to normalize is in favor of hexadecimal exponentiation:

   Using D. W. Sweeney's normalization shift frequency table and expanding it to number of shifts * probability *9.5 or 3Ø.3, one can see that hexadecimal exponentiation provides for shorter time spent in the normalization routine:

   binary: 91.81 * 9.5 = 872.195

   hex: 18.46 * 3Ø.3 = 559.338

The lessening of alignments has another significant offshoot: "less bits of precision are shifted right out of the calculation." This fact may go a long way to compensate for the loss of precision due to the >1 form of normalized numbers.

2.4.1 Hardware Implications

The floating point format is not expected to have a big influence on the hardware cost of the processor (assuming hardware floating point). The implications of selecting the 360 format are that parallel-shift-by-four (left and right) shift paths are desirable for fast normalization and alignment. Again considering the normalization and alignment statistics of Sweeney's article it can be stated that the average floating point execution times will be better than with binary normalization.

3.0 References

1) Sweeney, D. W. "An Analysis of Floating-point Addition". IBM Systems Journal, Vol. 4, No. 1, 1965.

2) Goldberg, B. "27 Bits are not Enough for 8-Digit Accuracy". CACM, Vol. 10, No. 2, Febr. 1967.

3) Knuth, D. E. "Seminumerical Algorithms" Volume 2 of "The Art of Computer Programming", Addison-Wesley Publishing Co., Reading, Mass.

4) Burroughs B5500/B65000 Programming Manual.

5) CDC STAR Floating Point Format Description.

Alignment shift frequencies for various radices

Shift	Radix 2	4	8	10	16	32	64
0	32.64	38.24	45.77	47.15	47.32	52.52	55.84
1	12.11	18.54	19.77	23.22	26.02	26.37	26.64
2	8.61	12.83	11.02	11.27	10.47	5.92	3.77
3	6.72	9.87	6.26	3.26	2.24	1.82	2.35
4	7.17	3.04	1.73	1.39	1.31	2.08	1.98
5	3.88	2.05	1.10	0.93	1.70	1.87	
6	4.39	1.01	0.80	1.54	1.24		
7	4.82	0.72	1.52	1.28			
8	1.29	0.63	1.00				
9	1.28	0.94					
10	1.31	0.72					
11	0.48	0.97					
12	0.58	0.74					
13	0.38	0.27					
14	0.38						
15	0.32						
16	0.33						
17	0.32						
18	0.40						
19	0.48						
20	0.36						
21	0.53						
22	0.48						
23	0.33						
24	0.36						
25	0.36						
26	0.19						
over	9.50	9.43	10.04	9.96	9.70	9.42	9.42

Normalization shift frequencies for various radices

Shift	Radix 2	4	8	10	16	32	64
result 0	1.42	1.42	1.42	1.42	1.42	1.42	1.42
overflow	19.65	10.67	6.52	7.19	5.50	5.69	2.60
0	59.38	72.11	79.40	79.80	82.35	83.86	87.36
1	6.78	7.96	8.75	8.04	7.29	5.99	6.04
2	3.47	3.35	1.64	1.55	1.38	0.87	1.23
3	2.35	1.49	0.38	0.28	1.01	0.88	0.47
4	1.91	0.34	0.43	1.03	0.30	0.41	0.88
5	1.06	0.14	0.71	0.16	0.32	0.88	
6	0.56	0.92	0.25	0.25	0.43		
7	0.48	0.18	0.22	0.28			
8	0.16	0.13	0.28				
9	0.14	0.15					
10	0.08	0.18					
11	0.09	0.17					
12	0.32	0.27					
13	0.55	0.52					
14	0.16						
15	0.02						
16	0.04						
17	0.09						
18	0.08						
19	0.07						
20	0.12						
21	0.07						
22	0.07						
23	0.00						
24	0.11						
25	0.16						
26	0.52						