# REQUIRED SCIENTIFIC FLOATING POINT ARITHMETIC

Lawrence A Liddiard

University Computer Center, University Of Minnesota

Minneapolis, Minnesota

Abstract - Previous papers in computer arithmetic have shown that correct rounded floating point with good arithmetic properties can be attained using guard digits and careful algorithms on the floating point fractions. This paper combines that body of knowledge with proposed exponent forms that are closed with respect to inversion and detection and recovery of exponent under and over flow. In addition radix 2 is shown to be the only base radix meeting minimal variation of precision, a condition necessary for the safe use of floating point. An effort is made to establish objective criteria in answer to the question; "What is the best division of the computer word into exponent and fraction parts?". Combining the previous results allows a required scientific floating point arithmetic to be portrayed and compared with available arithmetics on current computers.

Index Terms - Floating point exponents, optimal base radix, arithmetic properties, exponent overflow and underflow, grade report on current floating point arithmetics.

## I. INTRODUCTION

Previous papers have pointed out various poor arithmetic characteristics in the fractional arithmetic used in floating point numbers on current computers and have shown that many of these faults can be corrected by the use of guard digits and proper rounding. In this paper the anomalies in poor excess q exponent choice, base radix, and the minimum exponent range and fraction precision for floating point numbers are shown to be correctable. In addition, augmentation of the exponent by one bit is shown to provide attractive properties for exponent underflow and overflow detection. If Knuth's Fundamental Principle "Numerical subroutines should deliver results which satisfy simple useful mathematical laws whenever possible" [10p204] with corollary [10p196] "Floating point routines should be designed to preserve as many of the ordinary mathematical laws", is followed; then a blueprint can be made for good floating point hardware. If future computer designers follow that blueprint, then that will earn them A's on their report card from numerical analysts. In section II through V we will show that certain forms of floating point numbers have attractive or necessary qualities, and the combination of these forms with required properties previously stated by numerical analysts leads to a Required Scientific Floating Point format as presented in section VI. Section VII compares certain current computers with respect to exponent range and precision as to the Required Scientific Floating Point Arithmetic.

## NOTATION

We will use Knuth's notation [10] for floating point numbers. The floating point number is divided into exponent e, and fraction f with p digits using base b and excess q exponent so that:

$$(e,f) = f*b^{e-q}$$

Where e is an integer having a specified range and f is a signed fraction such that $|f| < 1$, which means that the radix point is to the left of the positional representation of f. P digits means that $b^p*f$ is an integer. Finally, (e,f) is said to be normalized if the most significant digit of the representation of f is non-zero, so that either $1/b \leq |f| < 1$ or, if $f = 0$, then $e-q$ has its smallest possible value.

## II. REQUIRED EXPONENT BIAS

By Knuth's fundamental principle it follows that if a floating point number $(e_j,f_j)$, $f_j \neq 0$ exists in (e,f) then its inverse should also exist in (e,f) with $f \neq 0$. This rule is violated in all current computers that we know to exist. For example, the CDC 6000 and CYBER 170 series with a decimal exponent range of approximately $10^{-292}$ to $10^{322}$ have a $10^{30}$ range ($10^{292}$ to $10^{322}$) and the BURROUGHS 5700 and 6700 series with exponent range of approximately $10^{-46}$ to $10^{69}$ have a $10^{23}$ range ($10^{46}$ to $10^{69}$) in which the inverse is set to 0. The two previous examples are extreme cases since both of these computer series have the radix point to the right of the fraction making f an "integer", and thus have a missing inverse range of approximately the square of the precision. Current "fractional" computers such as the DEC 10, HONEYWELL 6000 ($2^{-128}$ to $2^{127}$) and IBM 370 ($16^{-64}$ to $16^{63}$) where the exponent ranges have an even number of states, put the extra state into the negative part of the exponent range. Thus for these computers there exists a small exponent range in which the inverse has an exponent overflow fault. We will show that this extra state should be in the positive part of the exponent range if simple mathematical laws are to hold.

Let $$(1/b)*b^n \le |f_j|*b^n < 1*b^n$$

then the inverse $$\frac{1}{(1/b)*b^n} \ge \frac{1}{|f_j|*b^n} > \frac{1}{b^n}$$

If equality is handled separately $$1*b^{-n+1} > \frac{1}{|f_j|*b^n} > (1/b)*b^{-n+1}$$

Therefore, if the odd state is put into the positive exponent range the inverse of $f_j*b^n$ will exist as $f_i*b^{-n+1}$. Thus if the range of the exponent is $0 \le e < 2^K$, then the bias should be $2^{K-1}-1$ which will insure that the inverse of a number in (e,f), $f \ne 0$ will also exist in (e,f) with $f \ne 0$. Considering the equality condition for the inverse of $(1/b)*b^n = (1/b)*b^{-n+2}$, only the smallest normalized number with exponent $n = -2^{K-1}+1$ will have an inverse that does not exist in (e,f). That one number is handled properly in the augmented exponent field proposed in section III. The requirement for an even number of states in the exponent range suggests that a two's complement number should be used for the exponent part. Without the proposed exponent form the replacement of a division by the use of the product of the reciprocal causes unexpected results when either the inverse is set to zero, or to infinity giving a zero or exponent overflow fault for the product form when the quotient form would have given a result in (e,f).

## III. EXPONENT UNDERFLOW AND OVERFLOW

Continuing the application of the fundamental principle it follows that if numbers $(e_1,f_1)$ and $(e_2,f_2)$ exist then the product $(e_1,f_1)*(e_2,f_2)$ and the quotient $(e_1,f_1)/(e_2,f_2)$ with $f_2 \ne 0$ should also exist in (e,f). This is not true for current computers since the product or quotient may not fit into the bits allowed for the exponent. Current computer hardware usually has an immediate interrupt to allow interpretation of an exponent overflow or underflow condition with the resulting fraction correct and the exponent correct modulo the number of bits in the exponent. Some current computers set exponent underflow results to zero and exponent overflows to "infinity". As computers become more asynchronous and pipelined there has been a tendency to disallow immediate programmer control of exponent faults since complex hardware is required to capture and allow restarting of that machine state that caused the exponent fault. Knuth [10p189] shows that ignoring exponent underflow and setting the result to zero causes inconsistant results in certain cases such as (U*V)*W being set zero when U*V underflows to zero but that (U*W)*V may be non-zero for certain U,V and W's. A simple solution that allows all products and quotients to exist in (e,f) is the addition of what we will call the Exponent Underflow and Overflow (EUO) bit to the exponent field. This can also be done by considering the upper two most bits of the exponent field to be testable fault bits as is done in the CRAY-1 [7].

This has five advantages: In pipelined or asynchronous computers the out of range results can be examined after the arithmetic operation without having a complex hardware solution to the restoration of that machine state that caused the fault. The current practice of having separate interrupts for exponent underflow and overflow because of the modulo exponent result is simplified by having a single range fault since the true exponent and fraction are retained. The concept of the EUO bit in combination with the top most exponent bit allows simple hardware detection and condition bit setting for exponent faults. This extended form allows products and quotients with non-zero divisors to exist in (e,f) for all non-exponent faulted pairs of numbers. And finally, the addition of unnormalized states representing "underflow", "divide-by zero", "infinity" and "undefined" as different powers of two in the fraction of the smallest valued exponent range allows a very complete floating point arithmetic format to be defined.

Example of 3 bit exponent augmented with EUO bit

| Exponent Value | Binary | |
|---|---|---|
| | 3210 | |
| -7 | 0000 | |
| -6 | 0001 | Exponent Underflow Range |
| -5 | 0010 | bit3 = bit2 = 0 |
| -4 | 0011 | |
| -3 | 0100 | |
| -2 | 0101 | |
| -1 | 0110 | |
| 0 | 0111 | |
| 1 | 1000 | |
| 2 | 1001 | |
| 3 | 1010 | |
| 4 | 1011 | |
| 5 | 1100 | |
| 6 | 1101 | Exponent Overflow Range |
| 7 | 1110 | bit3 = bit2 = 1 |
| 8 | 1111 | |

If the exponent bits are augmented by an EUO bit, then the square of any non-exponent faulted number in (e,f) exists in that extended (e,f) range since:
$$(1/b)*b^{2n-1} = (1/b^2)*b^{2n} \le (|f|*b^n)^2 < 1*b^{2n}$$
even when R* rounding is used. Using the exponent example above, the maximum squared value is $[(1-2^p)*2^4]^2 < 1*2^8$ and the minimum squared value is $[(1/2)*2^{-3}]^2 = (1/2)*2^{-7}$ both of which are in the extended (e,f). Since the inverse exists except for the minimum value in (e,f){ie,(1/2)*2**(-7) in the example given above}, then $(e_1,f_1)/(e_2,f_2) = (e_1,f_1)*(1/(e_2,f_2))$ exists and the special case of max(e,f)/min(e,f)exists in (e,f) although 1/min(e,f) does not exist in the restricted (e,f) exponent range. As a practical example, G. E. Forsythe [8] showed that if the square of a floating point number would not cause an exponent fault, then the normal algorithm for solving the quadratic equation would still be useable for equations whose coefficients had large absolute valued exponents.

If computers do not use the EUO bit concept, then a reasonable solution is to have one more bit in the exponent range. Thus for the practical applications where the square of a number is required in intermediate calculations, this extra exponent bit will allow the computation of a solution without incurring an exponent overflow or underflow fault. For example, if a floating point unit is to minimally handle exponent ranges up to $10^{\pm 100}$ for final results, the actual exponent range must be $10^{\pm 400}$. Of this range, $10^{\pm 200}$ is required for the typical squares of numbers and complex divisions and absolute values used in the intermediate calculations of most numerical algorithms, with numbers greater than $10^{200}$ or less than $10^{-200}$ being considered as range faults.

Finally, the EUO bit is not a complete panacea since it will not completely handle the classical algorithms for complex division and complex absolute value for all values in (e,f). The classical computation of (a+bi)/(x+yi) and abs(z) had an intermediate sum of $x^2 + y^2$ and previous floating point implementations could only handle numbers with values less than the square root of the maximum exponent range. For example: if the exponent range allowed $10^{\pm 100}$ full range values, then x,y were restricted to be less than $10^{50}$ or greater than $10^{-50}$, if the sum $x^2 + y^2$ was not to have an exponent overflow or underflow. With an EUO bit, full range values are allowed for x and y; but there is a square root of 2 radius at the upper end of the exponent range in which the summation of the squares may overflow out of (e,f). Even this small range may be properly handled if an "infinity" value state is implemented in the floating point arithmetic and a condition test allowed on that state. Else if the fast registers or stack hardware have 1 or 2 more exponent bits then in the exponent format of main store, the preceding classical algorithms can be handled.

## IV. CHOICE OF BASE FOR FRACTION

On the choice of base for the fraction, both [6] and [2] have suggested that the quaternay base representation might be best; since in the first paper average relative representation error (ARRE) and in the second RMS Relative Error Criterion, $f_2(k,p)$ are shown to be better for base 4 than base 2. In both papers, however, using maximum relative representation error (MRRE) or worst case relative error, $f_1(k,p)$ to compare base 2 and base 4; it was shown both bases had identical worst case relative error. As D. Knuth points out [10p204] "the enjoyment of the tools one works with is of course an essential ingredient of successful work". In the case of the floating point fraction it is minimal variation of precision that makes an essential contribution to the safe use of floating point tools. Otherwise, let us illustrate by three examples that the use of a higher than base 2 fraction requires "expertise" in order to achieve the accuracy that should be inherent in the tool.

EXAMPLE 1 [5p117]

" pi/2 = 1.57 ... = $16^1$ * (0.098 ...)

contains three leading zero bits in the fraction. The single-precision fraction on the IBM 360 and SIGMA 7 computers contain 24 bits, of which only 21 can be significant in the case. On the other hand,

pi/4 = 0.707 ... = $16^0$ * (0.707 ...)

has no leading zero bits in its representation, hence can be carried to a full 24 bits of precision. However, the computation of pi/4 as 0.5*(pi/4) results in only 22 bits of accuracy (the 22nd bit of pi/4 is a zero bit agreeing with the first of three "garbage" bits shifted in during the computation). With careful planning, computations can often be rearranged to avoid poor normalization of intermediate results, thus preserving significance in the final result. In our trivial example above, we could have retained full significance in our computation of pi/4 by starting from 2/pi, which has no leading zero bits in its representation, and using division instead of multiplication. In general, we may preserve as much as one decimal place of significance by computing x*pi/2 as x/(2/pi). This saving is nontrivial in any case, and especially so if the arithmetic involved has only seven significant figures to start."

EXAMPLE 2 [5p181]

" There is one other advantage to this approach for base-16 arithmetic. For certain intervals, $e^x/2$ is of the form $16^n(1+\varepsilon)$, while sinh(x) has the representation $16^n(1-\sigma)$. Thus, on these intervals, the quantity $e^x/2$ has poor normalization (three leading zero bits), while the final result has no leading zero bits, but instead three low-order "garbage" bits as a consequence of the poor normalization of the intermediate computation. With the proper choice of v, exp(x + ln v) will have good normalization whenever sinh(x) does. "

EXAMPLE 3 [19]

To calculate the complex absolute value robustly by avoiding unnecessary exponent underflow or overflow the formula $|Z| = V*(1 + (W/V)^2)^{\frac{1}{2}}$ is used. But to do it accurately for base 16 requires $|Z| = 2V*(1/4 + (W/2V)^2)^{\frac{1}{2}}$ to be used so that the square root term does not have leading zero bits which result in loss of precision for Z. Note in addition this paper showed that the most accurate computation of abs(Z) was to use $|Z| = (W^2 + V^2)^{\frac{1}{2}}$ which is the classical algorithm that can be used with the extended exponent range of section III.

Thus to do good work using floating point arithmetic with a base greater than 2, the user must constantly be aware of "tricks" in the representation of constants and formulae. But even these "tricks" will not avoid the ghost of poor normalization that is always ready to strike some intermediate computation and reduce the accuracy of the final results. This is the principle argument against the use of fraction bases greater than 2.

A secondary argument is discovered in the analysis of programs written in higher level languages. In the analysis of FORTRAN programs [11] it was discovered that certain forms such as A*2 and A/2 comprise a high percentage of the total multiplicative operators (although the analysis forgot to count the occurances of 2*A, 2.*A, 2.0*A, etc.). Only in base 2 floating point arithmetic are these operations done quickly with an add to exponent and without the loss of precision that can occur with bases greater than 2.

Conclusion: A base 2 fraction is the best possible under the principle of minimum variation of precision, a requirement for safe use.

## V. MINIMUM NUMBER OF BITS FOR EXPONENT AND FRACTION

The division of a floating point word into exponent and fraction parts has been a historical one based on components available for fast reliable memories, on analyses of computational problems and on the computer designer's whim. The band width of current memory technology and the published papers on floating point arithmetic are such that the overriding consideration in a current computer should only be the accuaracy of the fraction part and the freedom from exponent faults.

### Minimum Precision Advocates

Von Neumann [16] advocated 10 to 14 decimals, finally settling on 12 by designing a 40 bit computer with fractional precision. However, "It would seem therefore not at all clear whether the modest advantages of a floating binary point offset the loss of memory capacity and increased complexity of the arithmetic and control circuits." Gregory [9] reported that in April 1961, a week long international conference on matrix computations held at Gatlinburg, Tenn. concluded "There was unanimous agreement among the participants at that conference that computer manufacturers should be urged to increase the word length on all future scientific computers to at least 48 bits". His final statement is that "An electronic computer should have at least as good an arithmetic unit as a desk calculator". Three years later in April 1964, IBM announced the 360 series with 32 bit precision, down 4 bits from their previous 36 bit standard. Cody [4] stated that there should be 30 to 40 significant bits in single precision and 80 to 100 bits in double precision.

### Minimum Exponent Range Advocates

Numerous problems encounter exponent range limitations. For example: [15] " It is expected that a recomputation of the function on the UNIVAC 1108 which has a floating point of $10^{\pm 300}$ will allow computation of values $F_n^{(-)}$ and $G_n^{(-)}$ in all regions of practical interest without problems of characterristic underflow or overflow." which had occured on the UNIVAC 1107 (range $10^{\pm 38}$) and the IBM 1620 ( range $10^{\pm 99}$). Cody [4] stated " We therefore agree with Kuki that $10^{75}$ is not adquate whereas $10^{300}$ appears to be".

The fact that $10^{125}$ is greater than the estimated number of nucleons in the universe [12] is used by some scientific users as a basis for the exponent range to be required on computers. If computers were only used to count nucleons, this might be a valid argument; but many algorithms used for root solving and evaluation of interesting functions [15] require exponent ranges of $10^{1000}$. If the number $10^{125}$ has any significance with respect to computer exponent ranges, it is as a minimum lower bound.

### Exponent Ranges larger for Greater Precisions

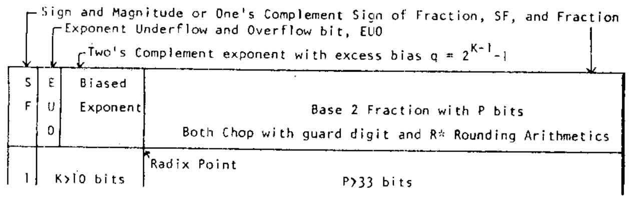
Several computer manufacturers have larger exponent ranges for double precision than for single. For example: The BURROUGHS 6700 and the UNIVAC 1108 computers have this feature. Knuth [10] has a single byte exponent for single precision and a double byte exponent for double precision in the MIX computer.

In order to present a useful criterion of required exponent range versus fraction precision, an acceptable ratio was sought in a matirx problem that would require a larger exponent range as precision increased, thus matching the intuitive analysis used in the MIX computer. The elements of the Hilbert matrix can be considered as random uniform numbers in [0,1] which produce a very ill-conditioned matrix of practical value. When the ratio of the exponent range required to handle the determinant of the Inverse Hilbert matrix are plotted versus the precision required to hold the maximum element of that matrix, a straight line is produced which seems to separate those computers with a "good" exponent range from those with a "too small" exponent range. If in addition it is accepted that a computer should have the minimum 10 decimals precision and two digit exponent ($10^{\pm 99}$) of a good desk calculator then these three lines can be plotted, A-1, against current computers, A-2, in Appendix A. Computers that have a ratio of exponent range to precision above and to the right of this line meet a minimum floating point standard, while those below and to the left do not. It is our contention that this graph, A-1, gives criteria on exponent range and precision that must be met by computers purporting to be Scientific Processors.

Note that we condone the use of smaller exponent and precision formats for special applications, such as: coordinates for grid plotting and graphic use; but not for blind use scientific computation. Thus such short formats must never be allowed to become the standard for scientific work such as has occured with the single precision hexadeciaml format of 32 bits.

## VI. REQUIRED SCIENTIFIC FLOATING POINT FORMAT

Combining the results of sections II through V with previous papers on the form and rounding of the fractional part, we can arrive at a required scientific floating point format.


Sign and Magnitude or One's Complement Sign of Fraction, SF, and Fraction
Exponent Underflow and Overflow bit, EUO
Two's Complement exponent with excess bias q = 2^(K-1)-1
S F
E U O
Biased Exponent
Base 2 Fraction with P bits
Both Chop with guard digit and R* Rounding Arithmetics
Radix Point
1
K>10 bits
P>33 bits

In addition to the minimum 45 bit format portrayed above a letter grade rating sheet for the various current floating point format follows:

## RATING SHEET FOR SINGLE PRECISION FORMATS

| ATTRIBUTE | CRITERION OR PAPER | RATING |
|---|---|---|
| FRACTION | | |
| 1) Form | | |
| Sign and Magnitude | Independent Exponent and Fraction fields | A |
| One's Complement | Exponent field bits depend on Sign of Fraction | B |
| Two's Complement | Exponent field bits depend on SF and there exist numbers U,V for which $U-V \neq -(V-U)$ [10p209ex11] | C |
| 2) Radix Point | | |
| Fractional | Allows symmetric Exponent Range, see section II | A |
| Integer | Asymmetric Exponent Range implies certain inverses do not exist in (e,f) | C |
| 3) Base Radix | See section IV | |
| Two | Minimum precision variation, a requirement for safe use | A |
| Greater Than Two | Blind use gives large variations in precision, thus unsafe usage | C |
| 4) Arithmetic | | |
| R* Rounding | [14] | A |
| R Rounding | [14] | B |
| Other Rounding | If normal arithmetic rules | B |
| | If normal arithmetic rules do not hold | C |
| Chop with Guard | [14] | A |
| Chop without Guard | There exist numbers U,V,W for which U<V but U*W>V*W, if base > 2 [10p209ex10] | C |
| 5) Precision, P | See section V | |
| 44-49 bits, 13-14D | | A |
| 38-43 bits, 11-12D | Von Neumann minimum precision | B |
| 34-37 bits, 10D | Minimum Calculator precision | C |
| <34 bits, 9D | Less than good calculator | F |
| EXPONENT | | |
| 1) Form | See section II | |
| Two's Complement | Bias $q=2^{K-1}-1$ | A |
| | Bias $q=2^{K-1}$, some inverses do not exist | B |
| One's Complement or Sign and Magnitude | Non-contiguous exponent range makes add to exponent difficult and a range of inverses does not exist | C |
| 2) Exponent Bits,K | See section V. Only for fraction precisions of 34 to 50 bits. Section III if no EUO | |
| K > 13 | Allows $10^{\pm 1233}$(3 decimal exp) | A |
| K = 12 - 13 | Allows $10^{\pm 308}$ or $10^{\pm 616}$ | B |
| K = 11 | Allows $10^{\pm 154}$(2 decimal exp.) | C |
| K < 11 | Less than 2 decimal exponent | F |
| ARITHMETIC FAULTS | | |
| 1) Form | See section III | |
| One bit EUO field with all exception states | | A |
| One bit EUO field with only divide-by-zero fault | | B |
| All arithmetic faults have good interrupt recover | | B |
| Underflow set 0, overflow and zero divisor cause faults or recoverable states | | C |
| No indication of any arithmetic fault [10p189] | | F |

## VII. SOME COMPUTER RATINGS

Some computer ratings with respect to exponent range and fraction precision for current computers, if used as scientific processors.

Acceptable Computers for Single and Double Precision

BURROUGHS SCIENTIFIC PROCESSOR (BSP)
CDC 3000, 6000, 7000 and CYBER 170 series
CRAY-1

Acceptable Computers for Double Precision Only

BURROUGHS 5700, 6700 and 7700 series
MIX
UNIVAC 1108 and 1110 series

Non-Acceptable Computers

AMDAHL Vx series
BURROUGHS 1700
CDC STAR
DEC 10 and 20 series
GE 600 series
HONEYWELL 6000 and 66 series
IBM 360, 370, 303x, and 709x series
MU 5
SIGMA 7
TI ASC
UNIVAC 1107

## VIII. CONCLUSIONS

What are the consequences of not requiring good scientific floating point arithmetic on the computers used in solving numerical problems?

First: There will continue to be ordinary problems that should be amenable to computer solution that will fail because of limited exponent range. The complex rescaling required for these problems, will either require unnecessary programming time or the problem will be abandoned by the casual computer user.

Second: The short fraction in a number of current computers will continue to produce answers that can not be trusted; but the casual computer user will not be aware of that possibility. As an industry that must have standards, the erroneous results produced by these too small floating point fractions, if not corrected by the computing industry, will cause an external agency to apply "Truth in Computing" laws to clean up the past industry mistakes.

Third: If the required scientific floating point arithmetic is not in the hardware, it will not be used. Two examples: Knuth [11] when delving into users programs showed that only a few times "we observed double precision being used, although the numerical analysis professors in our department strongly recommend against the short precision operators of the 360; it serves as another indication that our department seems to have little impact on the users of our computer." The original PASCAL compiler code generation for the CDC 6000 [18] allowed the choice of a correct rounding option for additive and multiplicative operations that was approximately 2.5 times slower than that generated by the machine rounding instructions. Needless to say the latest PASCAL code generation [1] in order to be as fast as competing languages has quietly eliminated that option. Unlike [3] this paper is an attempt to influence present and/or future machine design, either by direct appeal or by economic means when computers are selected on the basis of the Required Scientific Floating Point Arithmetic criteria presented in this and previous papers.

## FUTURE RESEARCH QUESTIONS

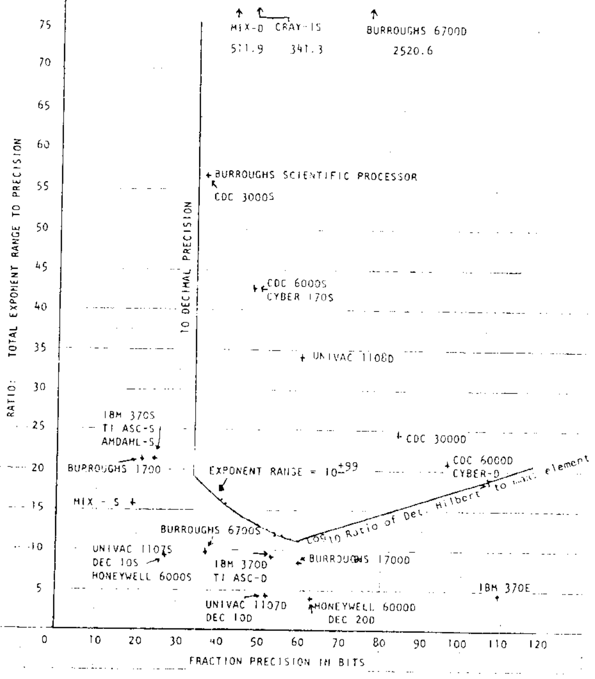
1) Is there a preferred double precision floating point format?
2) Is the type of accumulator on the GOLEM B and HONEYWELL 6000 computers part of a required arithmetic unit for quality single and double precision results?
3) Can it be shown that one higher precision accumulation of current precision combined with full upper and lower parts for multiplicative operations might obivate the need for significant digit arithmetic?

## ACKNOWLEDGEMENTS

People: M. Stein for starting me down the bit fiddling path in computing. K. Frankowski for encouraging me to write this paper.
Computers: The CDC 6000 for being the first large scale computer with microprocessor instructions and large exponent range and floating point precision(chop and round arithmetic need improving). The CRAY-1 and its designer for the EUO concept.
Talks: G. Amdahl for the Spring 1977 U of Minnesota talk in which he stated that all future computers would converge to the IBM 360 instruction set. I had pointed out then that the IBM floating point format was very discouraging for scientific work.
Papers: W. Cody for the interesting examples of the contortions needed to keep from losing precision in a hexadecimal computer. W. Kahan for those that suggest perfection is attainable. D. Knuth for the hope of a uniform theory in programming.

APPENDIX A-1

Plots of various computers versus the line graph of 10 decimal, 2 digit exponent and $\log_{10}$ of the ratio of the determinant of the Inverse Hilbert Matrix to its maximum element



APPENDIX A-2

FLOATING POINT CHARACTERISTICS FOR CURRENT COMPUTERS, RANKED BY INCREASING MINIMUM PRECISION IN BITS

| COMPUTER SERIES | FLOATING POINT BIT LENGTH | BASE RADIX | RADIX POINT | PRECISION TYPE | PRECISION LENGTH IN RADIX BYTES | MINIMUM PRECISION IN BITS | EXPONENT LOWER RANGE | EXPONENT UPPER RANGE | TOTAL EXPONENT RANGE IN BYTES | RATIO: EXPONENT RANGE / PRECISION |
|---|---|---|---|---|---|---|---|---|---|---|
| MIX | 31 | 64 | F | S | 4 | 19 | -32 | 31 | 63 | 15.8 |
| IBM 370, TI ASC | 32 | 16 | F | S | 6 | 21 | -64 | 63 | 127 | 21.2 |
| BURROUGHS 1700 | 34 | 2 | F | S | 24 | 24 | -256 | 255 | 511 | 21.3 |
| UNIVAC 1107, DEC 10 HONEYWELL 6000 | 36 | 2 | F | S | 27 | 27 | -128 | 127 | 255 | 9.4 |
| CDC 3000, BSP | 48 | 2 | F | S | 36 | 36 | -1023 | 1023 | 2046 | 56.8 |
| BURROUGHS 6700 | 47 | 8 | I | S | 13 | 37 | -51 | 76 | 127 | 9.8 |
| MIX | 61 | 64 | F | D | 8 | 43 | -2048 | 2047 | 4095 | 511.9 |
| CDC 6000, CYBER 170 | 60 | 2 | I | S | 48 | 48 | -975 | 1070 | 2045 | 42.6 |
| CRAY-1 | 64 | 2 | F | S | 48 | 48 | -8192 | 8191 | 16383 | 341.3 |
| UNIVAC 1107, DEC 10 | 72 | 2 | F | D | 52 | 52 | -101 | 127 | 228 | 4.4 |
| IBM 370, TI ASC | 64 | 16 | F | D | 14 | 53 | -64 | 63 | 127 | 9.1 |
| BURROUGHS 1700 | 70 | 2 | F | D | 60 | 60 | -256 | 255 | 511 | 8.5 |
| UNIVAC 1108 | 72 | 2 | F | D | 60 | 60 | -1024 | 1023 | 2047 | 34.1 |
| HONEYWELL 6000, DEC 10 | 72 | 2 | F | D | 63 | 63 | -128 | 127 | 255 | 4.0 |
| BURROUGHS 6700 | 95 | 8 | M | D | 26 | 76 | -32755 | 32780 | 65535 | 2520.6 |
| CDC 3000 | 96 | 2 | F | D | 84 | 84 | -1023 | 1023 | 2046 | 24.4 |
| CDC 6000, CYBER 170 | 120 | 2 | M | D | 96 | 96 | -927 | 1070 | 1997 | 20.8 |
| IBM 370 | 128 | 16 | F | E | 28 | 109 | -64 | 63 | 127 | 4.5 |

PRECISION TYPE S - SINGLE D - DOUBLE E - EXTENDED
RADIX POINT F - FRACTIONAL I - INTEGER M - MIDDLE OF TWO PARTS

APPENDIX A-3

| N | DETERMINANT HILBERT MATRIX | DETERMINANT INVERSE HILBERT | LARGEST ELEMENT OF INVERSE | BINARY BITS TO REPRESENT LARGEST ELEMENT | RATIO EXPONENT RANGE TO PRECISION |
|---|---|---|---|---|---|
| 1 | $1.0 \times 10^{0}$ | $1.0 \times 10^{0}$ | $1.0 \times 10^{0}$ | 1 | |
| 2 | $8.3 \times 10^{-2}$ | $1.2 \times 10^{1}$ | $1.2 \times 10^{1}$ | 4 | |
| 3 | $4.6 \times 10^{-4}$ | $2.2 \times 10^{3}$ | $1.9 \times 10^{2}$ | 8 | |
| 4 | $1.7 \times 10^{-7}$ | $5.9 \times 10^{6}$ | $6.5 \times 10^{3}$ | 13 | |
| 5 | $3.7 \times 10^{-12}$ | $2.7 \times 10^{11}$ | $1.8 \times 10^{5}$ | 18 | 4.8 |
| 6 | $5.4 \times 10^{-18}$ | $1.9 \times 10^{17}$ | $4.4 \times 10^{6}$ | 23 | 6.0 |
| 7 | $4.8 \times 10^{-25}$ | $2.1 \times 10^{24}$ | $1.3 \times 10^{8}$ | 27 | 6.3 |
| 8 | $2.7 \times 10^{-33}$ | $3.7 \times 10^{32}$ | $4.3 \times 10^{9}$ | 32 | 7.3 |
| 9 | $9.7 \times 10^{-43}$ | $1.0 \times 10^{42}$ | $1.2 \times 10^{11}$ | 37 | 7.8 |
| 10 | $2.2 \times 10^{-53}$ | $4.5 \times 10^{52}$ | $3.5 \times 10^{12}$ | 42 | 8.8 |
| 11 | $3.0 \times 10^{-65}$ | $3.3 \times 10^{64}$ | $1.2 \times 10^{14}$ | 47 | 9.3 |
| 12 | $2.6 \times 10^{-78}$ | $3.8 \times 10^{77}$ | $3.7 \times 10^{15}$ | 52 | 10.4 |
| 13 | $1.4 \times 10^{-92}$ | $7.1 \times 10^{91}$ | $1.1 \times 10^{17}$ | 57 | 10.8 |
| 14 | $4.9 \times 10^{-108}$ | $2.0 \times 10^{107}$ | $3.5 \times 10^{18}$ | 62 | 12.0 |
| 15 | $1.1 \times 10^{-124}$ | $9.1 \times 10^{123}$ | $1.1 \times 10^{20}$ | 67 | 12.4 |
| 16 | $1.4 \times 10^{-142}$ | $7.1 \times 10^{141}$ | $3.5 \times 10^{21}$ | 72 | 13.5 |
| 17 | $1.2 \times 10^{-161}$ | $8.3 \times 10^{160}$ | $1.1 \times 10^{23}$ | 77 | 14.0 |
| 18 | $5.4 \times 10^{-182}$ | $1.9 \times 10^{181}$ | $3.5 \times 10^{24}$ | 82 | 14.9 |

## REFERENCES

[1] U. Ammann, "On Code Generation in a PASCAL Compiler", Software-Pract. and Expr. Vol 7, pp391-423 (1977)

[2] R. P. Brent, "On the Precision Attainable with Various Floating-Point Number Systems" IEEE Trans. Comput. Vol. C-22, pp601-607, July 1973

[3] W. J. Cody, "The Influence of Machine Design on Numerical Algorithms", Proc. SJCC, Vol. 30 pp305-309

[4] W. J. Cody, "Desirable Hardware Characteristics for Scientific Computations, A Preliminary Report", SIGNUM Newsletter Vol. 6, No. 1, January 1971 pp16-31

[5] W. J. Cody, "Software for the Elementary Functions" in Mathematical Software, Academic Press, New York, NY, 1971, pp171-186

[6] W. J. Cody Jr., "Static and Dynamic Numerical Characteristics of Floating Point Arithmetic", IEEE Trans. Comput. Vol. C-22 pp598-601, July 1973

[7] CRAY-1 Computer System Reference Manual, CRAY RESEARCH INC., 1976

[8] G. E. Forsythe, "Solving a Quadratic Equation on a Computer" in The Mathematical Sciences, A Collection of Essays, National Research Council (COSRIMS) The M.I.T. Press, 1969

[9] R. T. Gregory, "On the Design of the Arithmetic Unit of a Fixed-Word-Length Computer from the Stand Point of Computational Accuracy" IEEE Trans. on Electronic Comput. Vol.15 pp255-257, April 1966

[10] D. E. Knuth, The Art of Computer Programming, Volumne II, Addison-Wesley, Reading, Mass. 1969, pp180-229

[11] D. E. Knuth, "An Emperical Study of FORTRAN Programs", Software-Pract. and Expr. Vol. 1, No. 2 pp105-134 (1971)

[12] D. E. Knuth, "Mathematics and Computer Science: Coping with Finiteness", SCIENCE, Vol. 194, No. 4271, 17 Dec. 1976, pp1235-1242

[13] W. Kahan, "Implementation of Algorithms, Part I", Technical Report 20, University of California, 1973, NTIS AD-769-124

[14] H. Kuki and W. J. Cody, "A Statistical Study of the Accuracy of Floating Point Number Systems", Commun. Assn. Comput. Mach. Vol. 16, pp223-230, April 1973

[15] E. J. Martin Jr. and P. C. Patton, "Evaluation of Certain Definite Integrals Frequently Encountered in Radiation and Diffraction Problems Involving Circular Geometry", The Comput. J. Vol. 8, No. 3, pp256-263

[16] John von Neumann, Collected Works, Volumne V, Design of Computers, Pergamon Press, Oxford, 1963

[17] A. Padegs, "Structured Aspects of the System/360 Model 85 - III Extension to Floating Point Architecture", IBM Sys. J. Vol. 7, No. 1, 1968 pp22-29

[18] N. Wirth, "On 'PASCAL' Code Generation and the CDC 6000 Computer", STAN CS-72-257, Computer Science Dept., Stanford University, 1972

[19] J. Wisniewski, "Some Experiments with Computing the Complex Absolute Value", SIGNUM Newsletter Vol 13, No 1, March 1978, p11

[20] Many Different Computer Reference Manuals