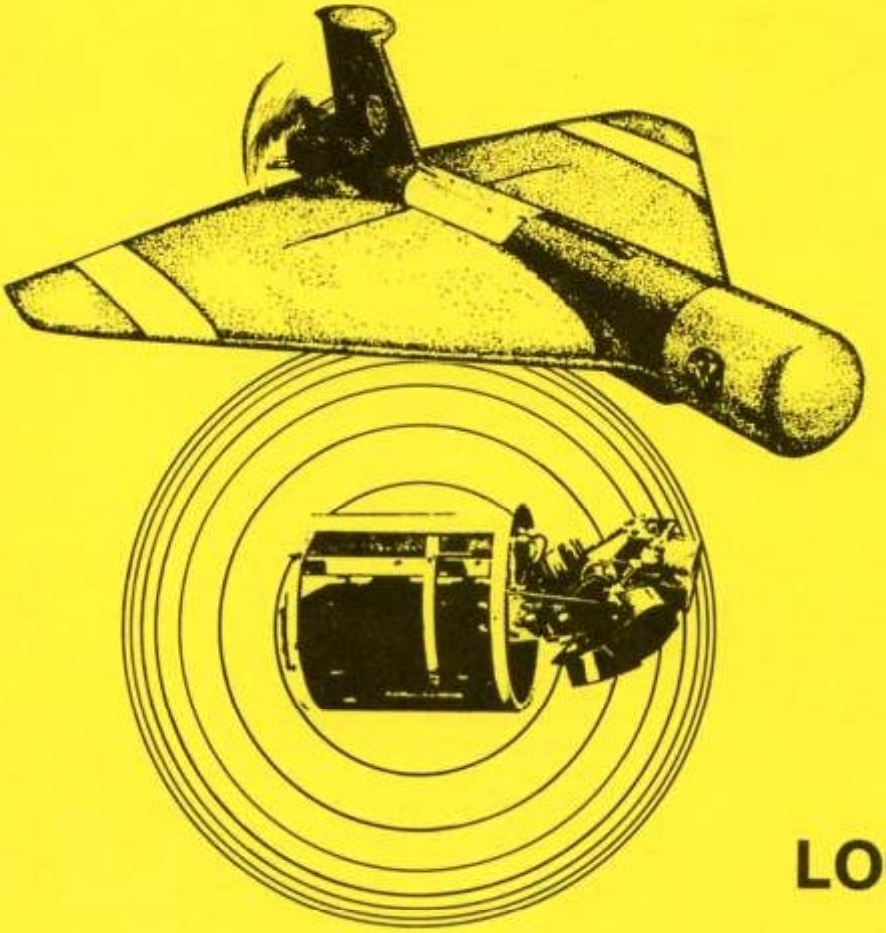
EQUIPMENT GROUP

ENGINEERING JOURNAL



LOCUST

TEXAS INSTRUMENTS
INCORPORATED

SEPTEMBER - OCTOBER
VOLUME 2 NUMBER 5

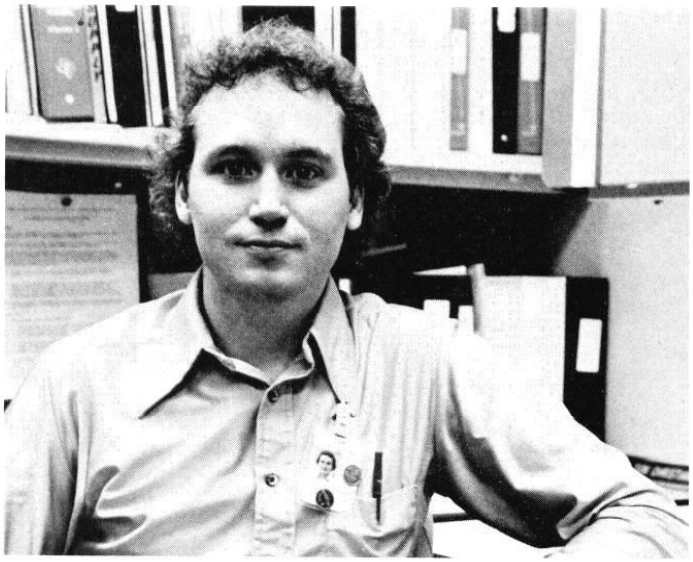
COMPUTATION NOTES

. . . features a variety of articles and information pertaining to computing resources for engineering. Articles related to both hardware and software will be welcomed, including specific program listings, and these may address applications ranging from handheld calculators to mini or super computers.

Observations of the Numerical Performance of the Texas Instruments 990 Minicomputer

DONALD J. CHRISTIAN

Don received a B.S. in Computer and Information Science from Ohio State in 1974, and is currently a graduate student of Computer Engineering at Sourthern Methodist University. In 1974, he joined TI's Science Services Division. His initial assignment was as a scientific programmer on graphics display systems used in seismic exploration. Since then, he has worked with marine navigation and interactive computer graphics systems for electronic design. Currently Don is a member of the technical staff with TI's Vision Aided Manufacturing department, and is involved in image understanding projects in a variety of industrial applications, including manufacturing, inspection, and robotics. He is also a TI company representative to the Association for Computing Machinery. Away from TI, Don's interests include image processing, computer graphics, and digital music synthesis. He has been spending his spare time playing classical and electric guitars and teaching his daughter to walk. He may be reached at X83-5051, MS 452.



THE FLOATING POINT SCHEME used in Texas Instruments 990 family of computers is identical to that used on IBM's 360-370 computers. It is commonly referred to as "chopped hex" because it uses a base 16 exponent and chops (truncates) the mantissa after every operation, rather than rounding it. Although the 360-370 product line has been very successful, its floating point format has long been regarded by the numerical analysis community as crude, and in some instances has rendered the 360-370 unsatisfactory for scientific applications. Table I compares the results from Kahan's problem run on several different computers. Kahan's problem is a short Fortran routine consisting of six adds, four subtracts, four divides, and one multiply. The true solution is zero divided by zero, which is undefined. The source code is listed in the appendixes.

Table I. Kahan's Problem

COMPUTER	SINGLE PRECISION	DOUBLE PRECISION
IBM 370/168	3.0	3.0
WITH FPS 190L AP	8.0	8.0
TI ASC UNVECTORIZED	0/0	0/0
TI ASC VECTORIZED	4.0	4.0
TI 980 - DX980	2.0	2.0
TI 980 - TIMAP (FPAP)	2.9999999962	-0.0000000011
TI 990/10 - PASCAL	3.0	3.0
TI 990/10 - FORTRAN	3.0	3.0
TI SR-10 CALCULATOR	0.0 WITH ERROR INDICATOR LIT.	
TI SR-51 CALCULATOR	BLINKS 9.999999999 99	
TI SR-59 PROG. CALC.	BLINKS 9.9999999 99	

It is interesting to note that the calculators correctly produced an error exception, while the computers (except for the ASC unvectorized Fortran version) did not. Even special purpose hardware such as the Floating Point Systems 190L array processor and the floating point auxiliary processor of the TI 980-TIMAP were incorrect. The usual Fortran remedy, DOUBLE PRECISION, produced the same wrong result but with twice the accuracy. In other words, it gave an answer which was twice as wrong.

The 370 chopped hex format was designed as a method for providing a large range with a one-byte exponent. The exponent's range was increased by making it a power of sixteen, rather than the usual power of two. The mantissa is thus shifted in four bit blocks, or "nybbles". This was a radical departure from the IBM 7090's binary based format, and the change was soon regretted. A large range is needed because no provision was made for extending it. The name "DOUBLE PRECISION" says that only the mantissa has been extended, rather than increasing other attributes as well. So all needs of all users must be satisfied within the range $-10^{**}75$ to $+10^{**}75$. To get an idea of the magnitude

of this range, the number of molecules in the atmosphere of the earth is only 10 **44. It has been my experience that double precision is used primarily as a fix-up when single precision fails. Kahan's test demonstrates that there are numerical problems which cannot be solved by simply lengthening the mantissa. It has been shown that a sanitary single precision scheme will greatly reduce or eliminate this "fix-up" motivation for extending the floating point word length.[6]

All users must bear the burdens associated with a hexadecimally inflated range. A few of the greatest tolls taken are:

1. Inefficient memory utilization — up to three wasted bits per word and 6.25 percent of total possible number configurations unused.
2. "Wobbly" precision — some numbers can be represented with greater accuracy than others.
3. A tendency for elementary arithmetic errors to be promoted in the mantissa. Truncation errors in particular are amplified and propagated throughout computations by oscillating normalization.

Floating point schemes which overcome these problems individually have been known for years, and they are discussed in the section of this article which deals with the pending IEEE floating point standard.

In this article, I will discuss the consequences of the chopped hex floating point scheme on the 990 and present the results of both Fortran and Pascal test codes run on the 990.

How Many Decimal Digits of Accuracy?

To examine this question more precisely, we should first compute the mantissa width of a floating point number in bits and then translate the result to a decimal figure. Single precision numbers in the 990 are allocated a total of 32 bits — 24 mantissa bits, 7 exponent bits, and a sign bit. Holding the exponent at a constant zero (represented as >40), the largest representable fraction is >.FFFFFF, which is approximately equal to 0.999 999. The width of this fraction is 24 bits. The smallest representable fraction is >0.100000, or 1/16. In binary notation this is written as

.0001 0000 0000 0000 0000 0000

Notice that the three leading bits are zeros, and used only as "place holders." These bits are unavailable for use in storing data, and data will be lost if place holders must be inserted into a number containing more than 21 significant bits. Table II shows the variable relationship between magnitude and accuracy for a heavily used region of numbers.

6.7 percent of all single precision floating point numbers are represented with only 21 mantissa bits (90.6 percent bit utilization efficiency). The average efficiency for floating point mantissas is 96.9 percent, or about 23.3 out of 24 bits.

Table II. Maximum Number of Data Bits in Mantissa of a Single Precision or REAL(7) Floating Point Number

A - TI 990 AND IBM 370 CHOPPED HEX FORMAT.
B - IEEE STANDARD FORMAT.

REPRESENTABLE REAL NUMBERS

	BITS OF ACCURACY ---->							
	18	19	20	21	22	23	24	25
0.0625 -				A				B
0.125 -					A			B
0.25 -						A		B
0.5 -							A	B
1.0 -				A				B
2.0 -					A			B
4.0 -						A		B
8.0 -							A	B
16.0 -				A				B
32.0 -					A			B
64.0 -						A		B
128.0 -							A	B
256.0 -				A				B
512.0 -					A			B
	18	19	20	21	22	23	24	25

Decimal numbers read into the machine will be truncated after the bit labelled "A," causing the "A" bit to lose most of its significance if the highest bit truncated was a one.

There is another aspect of the chopped hex scheme which is wasteful of memory. The first nybble (hex digit) of the mantissa is not allowed to be zero. A zero would imply that the number is denormalized, which is a feature not supported by either the 990 or the 370. The long integer format, which also uses 32 bits, has no forbidden bit configurations. It can represent a total of 4,294,967,294 different numbers. Disallowing a leading mantissa zero declares 268,435,454 of these symbols illegal. Thus out of 4.29 billion possible numbers, the 990 uses about 4.03 billion, or 93.75 percent. The same percentage of double precision numbers is unused, but here the total number is more than 281 trillion.

With only 93.75 percent of the total symbols acceessible and a bit utilization rate of 96.9 percent for the representable numbers, the overall memory efficiency figure is 90.84 percent. This corresponds to roughly three unused bits in every 32 bit word.

Table III shows the largest respresentable numbers in several 990 mantissa configurations.

Single precision mantissas on the 990 range from 21 to 24 bits in width, or about 6.2 to 7.2 decimal digits

Table III. Equivalent Floating Point Mantissa Widths

MANTISSA WIDTH IN BITS	LARGEST EXACT INTEGER		MANTISSA WIDTH IN DIGITS	
	HEX	DECIMAL	HEX	DECIMAL
25	1FFFFFF	33554431	6 1/8	7 3/10
* 24	FFFFFF	16777215	5	7 2/10 *
* 23	7FFFFF	8388607	5 1/2	6 8/10 *
* 22	3FFFFF	4194303	5 1/4	6 4/10 *
* 21	1FFFFF	2097151	5 1/8	6 2/10 *
20	FFFFF	1048575	4	6 1/10
19	7FFFF	524287	4 1/2	5 5/10
18	3FFFF	262143	4 1/4	5 3/10
17	1FFFF	131071	4 1/8	5 1/10
16	FFFF	65535	4	4 7/10
15	7FFF	32767	3 1/2	4 3/10
14	3FFF	16383	3 1/4	4 2/10

(marked by asterisks in Table III). Under optimum conditions, some numbers can be represented with seven digit accuracy. On the other hand, some numbers can contain only 6.2 decimal digits at best.

If we assume that we lose one bit to an error, the remaining accuracy will be 20 to 23 bits, or 6.1 to 6.8 decimal digits, and will not achieve seven digits for any number. Erosion of the mantissa can become quite severe with just a few operations. Bits can be lost in input and output conversions, number type conversions, and ordinary arithmetic. It is true that 990 single precision can represent seven digit accuracy in some cases, but for typical numbers read in, processed, and printed out, this can rarely be achieved.

Input and Output: Decimal Conversion

All computers which represent numbers in binary (base 2) form employ some sort of scheme to translate decimal numbers to and from the internal binary format. For integers the conversion is straightforward, but for fractions many shortcuts exist which obtain a result quickly at the expense of some accuracy. We must keep in mind that the quality of the floating point computation done on the machine can be no better than the binary-decimal converters.

Both Pascal and Fortran on the 990 use what Sturbenz calls a "truncation conversion transformation" for real number input. The number radix is converted, and the normalized fraction is chopped to fit into a single or double precision word. Rounding is done when real numbers are output as decimal. The problem is complicated when the decimal number cannot be represented exactly in binary form, such as 1/10 or 2/3. Numbers which can be represented exactly decimally but not hexadecimally, or vice versa, are quite common. Table IV shows the results of some decimal to floating point to decimal conversions in 990 Pascal. The internal conversions were done using the decimal type feature of Pascal. Both formatted and unformatted writes were printed directly from a REAL(7) floating point number.

The numbers labelled "rounded internally" are actually truncated, due to a bug in the Pascal compiler option ROUND. The digit loss problem is made visible by the fact that the input numbers are not exacly representable in single precision, and it is compounded by the truncation transforms to and from floating point. In the second example, output rounding is unable to recover the sixth digit in a REAL(7) number. Without rounding at least on output, the situation would clearly be intolerable to even casual users. Input truncation errors are usually masked by other numerical problems encountered during internal processing.

In unformatted Pascal output, "no more precision is ever printed than the value contains." [9] Numerical errors introduced by base conversion or other causes are not taken into account. The accuracy which can be represented by the number may be quite a bit less than the actual accuracy contained by the number after a base change or an arithmetic operation. Each lost bit of accuracy pulls the number down one step lower on the chart in Table III. The printed version of the number, however, remains at a constant width, having been determined prior to run time. This is illustrated by the UNFORMATTED WRITE in the last example of Table IV.

High level languages usually hide the problem of conversion loss, and many users are unaware of its existence. Even software verifiers which "prove" the correctness of a program can be tricked by accuracy lost in conversions or arithmetic. The construction of a software verifier which would compute bounds for errors

Table IV. Pascal Decimal Conversion

ORIGINAL DECIMAL [7 DIGITS]	TRUNCATED TO REAL(7)	UNFORMATTED WRITE	FORMATTED WRITE 11:7	ROUNDED TO DECIMAL INTERNALLY	TRUNCATED TO DECIMAL INTERNALLY
0.2443518	403E8DD6	2.443517E-01	0.2443517	0.2443517	0.2443517
0.0970596	4018D8E5	9.705955E-02	0.0970595	0.0970595	0.0970595
0.0087096	3F23ACAD	8.709598E-03	0.0087096	0.0087095	0.0087095
0.0007809	3E332D53	7.808998E-04	0.0007809	0.0007808	0.0007808
0.0000995	3D685553	9.949999E-05	0.0000995	0.0000994	0.0000994
0.0000096	3CA10FAF	9.599999E-06	0.0000096	0.0000095	0.0000095
0.0000010	3C10C6F7	9.999994E-07	0.0000010	0.0000009	0.0000009

introduced in floating point calculations would be a difficult task indeed.

Truncation in Floating Point Arithmetic

Truncation is involved in all floating point arithmetic operations on the 990. Whenever two floating point numbers are arithmetically combined, the result is chopped (truncated) to fit into a single or double precision word. In other words, when the exact result of a floating point operation falls between two representable numbers, the number closest to zero is chosen, rather than the one closest to the actual result. This means that the output of any single function will be of lesser magnitude or, at best, equal to the desired result.

Table V demonstrates truncation in floating point subtraction. The leftmost column shows the internal hexadecimal representation of the minuend, subtrahend, and result. The middle column shows the same numbers denormalized, as they might appear in the workspace area of a floating subtraction routine or a hardware floating point register. The decimal equivalents in the right column were translated from the numbers in column one by 990 Fortran's truncation transform.

Notice in the second example that the subtrahend introduced a truncation error 16 times greater than itself. In the third example, it caused an error 256 times greater than itself. Repetitive subtractions will continue to introduce errors of this magnitude and will eventually destroy the significance of the minuend.

Table VI demonstrates how truncation errors are multiplied by the wobbly normalization of the mantissa. Positive integers n are compared with 1/(1/N), an algebraic identity.

The problems illustrated by Table VI arise when the number to be inverted falls into the following configuration:

1. The original number is represented with the most significant bit of the mantissa set to one. Fifty percent of all floating point numbers will be of this type.
2. The first inversion [1/N] is performed. The mantissa now has three leading zero bits used as "place holders." Three bits of the mantissa are shifted out to the right and lost. Ninety-eight percent of type one numbers will also be type two.
3. The most significant of the three bits lost in step two happened to be a one. This loses the significance of one more mantissa bit when truncation is performed. Fifty percent of type two numbers will also be type three.
4. Reinversion is performed, yielding M=1/1/N. Since the magnitude of (1/N) was reduced, the magnitude of 1/1/N will be increased. The net effect is that four "garbage bits" have appeared in the mantissa. Two floating point divides have destroyed four bits out of a 24-bit mantissa, or one hex digit out of six.

Table V. Subtraction Truncation Test in 990 Fortran

	REAL *4 INTERNAL	DENORMALIZED HEX	DECIMAL EQUIVALENT	
1	411028F6	1.028F6000	1.0100002289	
	- 3C100000	- 0.00001000	- .0000009537	
	= 411028F5	= 1.028F5000	=1.0099992752	EXACT
2	411028F6	1.028F6000	1.0100002289	
	- 3B100000	- 0.00000100	- .0000000596	
	= 411028F5	= 1.028F5000	=1.0099992752	TRUNC.
3	411028F6	1.028F6000	1.0100002289	
	- 3A100000	- 0.00000010	- .0000000037	
	= 411028F5	= 1.028F5000	=1.0099992752	TRUNC.
4	411028F6	1.028F6000	1.0100002289	
	- 39100000	- 0.00000001	- .0000000002	
	= 411028F6	= 1.028F6000	=1.0100002289	NO OP.

Table VI. Inversion Test in 990 Pascal

ORIGINAL N DECIMAL	REAL	1 / N RECIPROCAL	1/(1/N) RECIPROCAL	MANTISSA BITS LOST
13.	41E00000	40124924	41E00007	3
171.	42AB0000	3F17F405	42AB0007	3
199.	42C70000	3F149539	42C70008	4
211.	42D30000	3F13698D	42D3000A	4
239.	42EF0000	3F112358	42EF000C	4
245.	42F50000	3F10B7E6	42F5000D	4
251.	42FB0000	3F105197	42FB000E	4

----------SUMMARY OF ERRORS----------

BIT LOSS	COUNT
1	641
2	711
3	690
4	361
TOTAL	2403. ERRORS IN 5000. TESTS.

48.06 PER CENT OF NUMBERS HAD ERRORS.

AVERAGE BITS LOST PER ERROR = 2.32

Addition, subtraction, and multiplication exhibit similar weaknesses to truncation noise. This could be incrementally improved if a rounding rule were used. However, the wobbling of the mantissa would still amplify round-off noise. To significantly improve the situation would require the use of a different floating point representation, more like the proposed IEEE standard.

The Numerical Reliability of Standard Library Routines.

Programmers tend to think of low level floating point functions, such as SIN, COS, LN, and SQRT, as elementary operations like addition or multiplication. They also expect the error bounds to be elementary — plus or minus the least significant bit. As we all know, low level functions are actually programs which may use tens or hundreds of elementary floating point operations, as in the sum of a series expansion. Even if we

assume that an algorithm has been "perfectly" implemented, the error bound of a library routine should be significantly greater than that of an elementary operation.

The math library could then be expected to exhibit poorer overall performance than the floating point arithmetic on which it is based. The test results shown in Table VII confirm this suspicion in the case of the natural logarithm and exponential {LN and EXP} functions of 990 Pascal. The routines were tested by comparing a positive integer N with the algebraic identity EXP (LN(N)).

More complex programs which are supported by the library cannot be expected to perform better than the functions they use. Synergistic floating point error effects can cause normally reliable programs to fail for certain sets of input data.

The Fortran exponentiating function (Y ** X) is an example of a relatively elementary function which is typically implemented using a math library function. The Fortran statement

Z=Y ** X

is expanded to

Z=EXP (LN(Y) * X).

The error bounds on this function can be no better than the bounds on the EXP and LN functions. Table VIII shows some test results from 990 Fortran.

The problem of keeping the floating point error bound low is similar to the analog electronics problem of keeping the noise-to-signal ratio low. Inside a modern digital computer, there is no reason why the uncertainty of any floating point operation should be allowed to exceed the least significant *bit* and not the least significant *digit*. This goal will be difficult if not impossible to achieve without a binary based floating point scheme.

Table VII. Pascal LOG and EXP Test

ORIGINAL NUMBER DECIMAL	REAL(7)	LN(N)	EXP(LN(N))	MANTISSA BITS LOST
84.	42540000	4146E4A0	42540037	6
179.	42B30000	4152FF88	42B30072	7
181.	42B50000	41532D0B	42B50073	7
630.	43276000	416721AB	4327601A	5
1282.	43502000	41727FB3	43502035	6
2487.	439B7000	417D19F0	439B7069	7

SUMMARY OF ERRORS

BIT LOSS	COUNT
1	1168
2	1272
3	838
4	50
5	26
6	29
7	31
TOTAL	3414. ERRORS IN 5000. TESTS.

68.2 PER CENT OF NUMBERS HAD ERRORS.

AVERAGE BITS LOST PER ERROR = 2.03

Table VIII. 990 Fortran Exponential Test

ORIGINAL NUMBER DECIMAL	REAL	N**(1/3)	(N**(1/3))**3	MANTISSA BITS LOST
168.	42A80000	4158491B	42A8006E	7
179.	42B30000	415A2BFE	42B30072	7
181.	42B50000	415A81A6	42B50073	7
241.	42F10000	416391C0	42F1002A	6
1110.	43456000	41A5AA63	43456091	8
1171.	43493000	41A8A56F	43493090	8
1308.	4351C000	41AEFB6C	4351C0A4	8
1334.	43536000	41B02259	435360BD	8
1393.	43571000	41B2B173	435710A7	8
1405.	4357D000	41B33470	4357D0AB	8
1436.	4359C000	41B4B36A	4359C0B7	8
2487	439B7000	41D8C6BD	439B7066	7

SUMMARY OF ERRORS

BIT LOSS	COUNT
1	973
2	1285
3	1663
4	549
6	61
7	63
8	10
TOTAL	4604 ERRORS IN 5000 TESTS.

92.08 PER CENT OF NUMBERS HAD ERRORS.

AVERAGE BITS LOST PER ERROR = 2.32

Numerical Considerations in Pascal

Pascal has many numerical features not found in Fortran. In addition to the standard 16 and 32 bit real and integer formats, it offers fixed binary and fixed decimal representation. These can be very useful for certain classes of computations where integers and floating point numbers are not quite right. Up to 31 fixed bits and 15 decimal digits per number can be stored, with the decimal point position in both cases user selectable.

The details of the conversions between formats, in general, must be specified precisely. For example, when converting from real to integer, either rounding or truncating must be requested. If it is not, the compiler will raise an error flag. Numbers converted from most formats into single precision will be truncated, and the truncation cannot be overridden by the user. This can create problems even in the case of long integer to real conversion. The intervals between numbers in long integer can be 256 times shorter than the intervals between the same numbers expressed in single precision, even though both formats use 32 bits.

When there are extra fractional bits in the result of a fixed binary arithmetic operation, the result is rounded before it is stored. In decimal arithmetic, rounding or truncating may be selected by the compiler option ROUND (see Table IV). In all floating point arithmetic, truncation without rounding is performed and cannot be overridden. Conversion from double to single precision is also truncating. Decimal numbers read as real are truncated, and real numbers written in decimal are rounded. Numerically aware programmers can kludge

around some of these restrictions, at the expense of transportability.

Standard library functions, such as ABS, LN, COS, etc. accept numbers in any format but return a floating point result. Mixed mode arithmetic is normally handled by floating all numbers before performing the arithmetic. This makes mixed mode arithmetic in Pascal especially susceptible to the problems of floating point arithmetic, even when none of the operands are REAL.

Much care was taken in the design of Pascal to insulate the program from individual peculiarities of the host computer. The largest numerical gap in this insulation is the fact that the program must use the local floating point scheme. A Pascal program's performance cannot exceed the basic numerical performance of the machine on which it is running. Identical source code may produce widely varying results on different computers and even on different models manufactured by the same company, as demonstrated by Kahan's test in Table I.

The precision desired in a floating point variable is expressed in decimal digits when the variable is declared. A REAL(10) variable should support ten digits of accuracy throughout all computations. On the 990, single precision is used for variables declared REAL(7). However, single precision can achieve seven digit accuracy in rare cases only. In Table IV, it was demonstrated that a six digit number may lose a digit in a single I/O cycle. The use of floating point arithmetic or library functions can erode two or three decimal digits. It is conceivable that REAL(7) variables may yield three digit accuracy or less, for some numbers in some computations. Pascal users on the 990 should not expect a REAL(N) variable to always support N decimal digits of accuracy.

The IEEE Floating Point Standard

The chopped hex format of the IBM 360-370 has been widely imitated. Some companies, such as Amdahl, Data General, and Texas Instruments, have followed IBM's lead. Others, such as Control Data and Cray Research, have provided more resolution and range than most users can afford. Neither of these approaches is a particularly good compromise between accuracy, efficiency, and cost for the majority of computer users. The old "standards," set in the 1960s, have been made obsolete by powerful new concepts developed during the 1970s.

An IEEE committee was organized in 1978 to combine the best of these new ideas into a floating point standard. The committee seems to be in unanimous agreement about several basic points. Some highlights are:

- Short (single) precision word length should be 32 bits. The exponent should occupy the first byte, and the mantissa the next three bytes.
- Long (double) precision word length should be 64 bits. The exponent, as well as the mantissa, should be extended. The range should be great enough to hold the product of two short numbers without underflow or overflow.
- The mantissa should be encoded with an implicit leading bit. Since normalization rules require that the leading digit (bit) be occupied, the bit need not be stored.
- The exponent should be binary (base two). This is needed for implicit leading bit encoding, and it eliminates the wobbly mantissa of the chopped hex format.
- A gradual underflow feature should be supported to increase the range of utility, at the expense of some accuracy in numbers which underflow.
- Special symbols should be reserved for the abstract quantities infinity and zero (both positive and negative) and "undefined." Arithmetic operations on these symbols should produce exceptions.
- Floating point exceptions (errors and undefined operations) should be handled in a manner which will assist the debugging of the program. Special symbols will be especially valuable here.
- When the true result of a floating point operation is between two representable numbers, the closest number to the true result should be used (rounding).

Several papers have been published which describe the proposed standard in detail, including the more controversial issues. I recommend the Kahan-Palmer and Palmer papers as good overviews. References are given in the bibliography.

Conclusions

The magnitude of errors shown in the examples given here would be intolerable on an inexpensive pocket calculator. On a computer the errors are often obscured by blind faith in the computer's overall numerical reliability, and by the relative inaccessibility of the machine for testing purposes. For the reader to run the tests shown here would require several hours of preparation and access to an expensive computer. On a calculator, tests could be made in less than a minute. This is somewhat ironic because the small number of serial computations done on a calculator usually prevents floating point er-

rors from growing very large. On a computer like the 990, where tens to thousands of operations may be done serially and the accuracy of each operation can be critical, significantly more error per operation is observed.

Floating point numbers are not real numbers, only approximations of real numbers. There is usually no guarantee that they will be good approximations. It is hoped that this paper will inspire an inquisitive spirit in the reader with regard to numerical problems inside a computer.

Bibliography

1. Brent, R., (1973), "On the Precision Attainable with Various Floating Point Number Systems." *IEEE Transactions on Computers*, Vol. C-22 No. 6.
2. Coonen, J. T., "Specifications for a Proposed Standard for Floating Point Arithmetic," Memo no. UCB/ERL M78/12, Elect. Res. Lab., Univ. California at Berkeley, 1978.
3. Eggers, T. W., Leonard, J. S., and Payne, M. H., "Handling of Floating Point Exceptions," draft submitted to IEEE micro. Floating point standards committee by Digital Equipment Corp., 1978.
4. Ginsberg, M., "Overview of Hardware, Software, and Algorithmic Influences on Computation," course notes, Dept. of Computer Science, SMU, Dallas, Texas (1978).
5. Kahan, W., and Palmer, J., "An Impending Floating Point Standard," *SIGNUM Newsletter*, March 1979.
6. Matula, On the Effects of Input and Output Conversions, *CACM*, Vol. 2, No. 1, Jan. 1968, p. 47-50.
7. Palmer, J., "The INTEL Standard for Floating Point Arithmetic," Proc. COMPSAC-77 IEEE, p. 107-112.
8. Sturbenz, P., *Floating Point Compuation*, Englewood Cliffs, N.J., Prentice-Hall (1974).
9. Texas Instruments, Model 990 Computer TI Pascal User's Manual.

APPENDIX A — PASCAL TEST CODES

```
PROGRAM PCTEST;

          PROGRAM   P C T E S T

          THIS ROUTINE TESTS THE FLOATING POINT TO DECIMAL
          CONVERSION ROUTINES OF PASCAL

VAR   KERRS   ARRAY [1..12] OF INTEGER;
      HEXX   LONGINT;
      IECNT, Ndigits_lost, TOTAL_BITS   INTEGER ;
      Total_digits, XNUM, PERCEN, Avg_digit_loss, Real7_number   REAL(7);
      ODECIMAL, ODECIMAL2, ODECIMAL3, DIFF   DECIMAL(8,7) ;
      TKLIM   REAL(7) ;
      TOTAL_ERRORS, OREAL   REAL(7) ;
      TKOUNT   REAL(7) ;
      STEP_DOWN   DECIMAL(8,7) ;

BEGIN
      WRITELN; WRITELN;
            PASCAL DECIMAL CONVERSION TEST (PC) -  TI 990/10';
      WRITELN; WRITELN; WRITELN;
          ORIGINAL      TRUNCATED      ',
          'UNFORMAT          FORMAT      ROUNDED      TRUNCATED ' );
      WRITELN;
          DECIMAL      TO REAL(7)    ',
            WRITE            WRITE      TO DECIMAL   TO DECIMAL' );
      WRITELN;
          (7 DIGITS)                    ',
                                 8,7      INTERNALLY   INTERNALLY' );
      WRITELN ;

      TKLIM  := 1000.0 ;
      STEP_DOWN  :=  0.95000000  ;
      TKOUNT  := 0.0     ;
      IECNT  := 0        ;
      ODECIMAL  := 0.30000000 ;

{---------- MAJOR FUNCTIONAL LOOP OF PROGRAM  ---------------------}

      REPEAT
      TKOUNT  := TKOUNT + 1.0   ;
      ODECIMAL  := ODECIMAL * STEP_DOWN ;
      Real7_number  := FLOAT(ODECIMAL,7)    ;   { TRUNCATE TO REAL(7)   }
           (*$NO ROUND*)
      ODECIMAL2  := DEC(8,7,Real7_number)   ;   { TRUNCATE BACK TO DECIMAL}
           (*$ROUND*)
      IF  ODECIMAL <> ODECIMAL2 THEN

{------ PRINT OUT SEVERE ABBERATIONS  ---------------------------}

        BEGIN
                (*$ROUND*)
          ODECIMAL3  := DEC( 8,7, REAL7_NUMBER );  { ROUND TO DECIMAL }
          HEXX: REAL  := Real7_number ;
          WRITELN(
          '  ', ODECIMAL:8:7,'    ', HEXX:8 HEX,'   ',Real7_number:,'   ',
          Real7_number:11:7,'    ',ODECIMAL3:8:7,'    ', ODECIMAL2:8:7);
          IECNT  := IECNT + 1      ;
        END
      UNTIL  TKOUNT > TKLIM  OR  REAL7_NUMBER < 0.000001
END
```

```
PROGRAM PDTEST;

          PROGRAM   P D T E S T

          THIS ROUTINE TESTS THE FLOATING POINT DIVIDE FUNCTION
          OF THE 990 VIA DOUBLE RECIPROCATIONS.

CONST TKLIM = 5000.0; IELIM = 40 ; THRESH = #8 ;

TYPE  REAL_DOUBLE = REAL(14) ;
      INTEGER_64BITS = RECORD
        LONGINT1 : LONGINT ;
        LONGINT2 : LONGINT
      END;

VAR   KERRS : ARRAY [1..8] OF INTEGER;
      HEXD, HEXX, HEXV : LONGINT;
      IECNT, DIFF, NBITS, N : INTEGER ;
      TOTAL_BITS, TOTAL_ERRORS : REAL(7) ;
      ONUM, VNUM, XNUM, PERCEN, TKOUNT, AVG_BIT_LOSS : REAL(7);
      VNUM2 : REAL(14) ;
      HEXV2 : INTEGER_64BITS ;

BEGIN
      WRITELN; WRITELN;
        ' PASCAL REAL NUMBER INVERSION TEST (PD) -  TI 990/10';
      WRITELN; WRITELN;
        '  ORIGINAL NUMBER      RECIPROCAL',
                                  1/(1/N)    MANTISSA' );
      WRITELN;
        '  DECIMAL    REAL        REAL      ',
             '    DOUBLE            FROM        BITS   ' );

      WRITELN;
        '                                    ',
                                  SINGLE      LOST ' );

      TKOUNT  := 0.0     ;
      IECNT  := 0        ;
      ONUM  := 1.0       ;

{---------- MAJOR FUNCTIONAL LOOP OF PROGRAM  ---------------------}

      REPEAT
      TKOUNT  := TKOUNT + 1.0   ;
      ONUM  := ONUM + 1.0       ;
      VNUM  := 1.0 / ONUM       ;
      XNUM  := 1.0 / VNUM       ;
      IF  ONUM <> XNUM THEN

{------ ERROR DETECTED.  DETERMINE HOW SEVERE IT IS.  -----------}

        BEGIN
        HEXD: REAL  := ONUM ;
        HEXX: REAL  := XNUM ;
        DIFF  := ABS( HEXX - HEXD );

        IF  DIFF <= 1 THEN NBITS  := 1 ELSE
         IF  DIFF <= 3 THEN NBITS  := 2 ELSE
          IF  DIFF <= 7  THEN NBITS  := 3 ELSE
           IF  DIFF <= #F  THEN NBITS  := 4 ELSE
            IF  DIFF <= #1F  THEN NBITS  := 5 ELSE
```

```
            IF  DIFF <= #3F  THEN NBITS  := 6 ELSE
              NBITS  := 7 ;
          KERRS[NBITS] := KERRS[NBITS] + 1 ;  { STATISTICAL TALLY }

{----- PRINT OUT THE FIRST FEW SEVERE ABBERATIONS  ----------------}

          IF  DIFF >= THRESH  OR IECNT <= IELIM THEN
            BEGIN
            HEXV  := REAL  = VNUM ;
            VNUM2  := 1.0D0 / DNUM ;
            HEXV2 := REAL_DOUBLE  := VNUM2 ;

            WRITELN(
            ' ',DNUM:7:0,'   ',HEX0:8 HEX,'   ',HEXV:8 HEX,' ',
            HEXV2.LONGINT1:8 HEX,  HEXV2.LONGINT2:8 HEX,
            '     ', HEXX:8 HEX,  '     ', NBITS:2 );
            IECNT  := IECNT + 1     ;
            END
          END
      UNTIL TKOUNT >= TKLIM    ;


{---------- GENERATE A SUMMARY ERROR REPORT  ----------------------}

      WRITELN; WRITELN; WRITELN(
            '          ** SUMMARY OF ERRORS **' ) ;
      WRITELN; WRITELN('           BIT LOSS    COUNT' );

      TOTAL_ERRORS  := 0 ;
      TOTAL_BITS    := 0 ;

      FOR N  := 1 TO 8 DO
      IF  KERRS[N] <> 0    THEN
         BEGIN
         TOTAL_ERRORS  := TOTAL_ERRORS + KERRS[N];
         TOTAL_BITS    := TOTAL_BITS + N * KERRS[N] ;
         WRITELN(
               '          ', N:5,  '     ', KERRS[N]:5 );
         END ;

    PERCEN  :=  TOTAL_ERRORS / TKOUNT * 100.0 ;
    WRITELN('          TOTAL     ');
    TOTAL_ERRORS:7:1,  ' ERRORS  IN', TKOUNT:6:0,  ' TESTS  ');
    WRITELN; WRITELN(
    '          ',PERCEN:6:2,' PER CENT OF NUMBERS HAD ERRORS ' );
    AVG_BIT_LOSS  := TOTAL_BITS / TOTAL_ERRORS ;
    WRITELN ; WRITELN(
    '          AVERAGE BITS LOST PER ERROR =', AVG_BIT_LOSS:6:2 );
END
```

```
PROGRAM PKTEST;
{-------------------------------------------------------------------}

          PROGRAM    P K T E S T

          KAHAN'S PROBLEM     A TEST FOR FLOATING POINT ARITHMETIC.
                                                                    }
TYPE  REAL16 = REAL(16) ;
      INTEGER_64BITS = RECORD
        HALF1, HALF2 : LONGINT;
      END;

VAR   H, X, Y, E, F, G : REAL;
      DH, DX, DY, DE, DF, DG  REAL(15);
      HEXE, HEXF, HEXG  LONGINT;
      HEXDE, HEXDF, HEXDG : INTEGER_64BITS ;
BEGIN
      WRITELN; WRITELN;
      WRITELN(
            KAHAN''S PROBLEM - [K1] - 990/10') ;
      WRITELN; WRITELN;
{-------------------------------------------------------------------
          CALCULATE ZERO IN SINGLE PRECISION.  { REAL (7) }
                                                                    }
      H  := 1.0E0 / 2.0E0                                         ;
      X  := 2.0E0 / 3.0E0  - H                                    ;
      Y  := 3.0E0 / 5.0E0  - H                                    ;

      E  := ( X + X + X )  - H                                    ;
      F  := ( Y + Y + Y + Y + Y )  - H                            ;
      G  := 2.0E0 * F / E                                         ;
      HEXE:  REAL  := E  ;
      HEXF:  REAL  := F  ;
      HEXG:  REAL  := G  ;

      WRITELN; WRITELN(
'                                    DECIMAL        INTERNAL');
      WRITELN; WRITELN(
'  SINGLE PRECISION KAHAN ZERO1 = ', F,  ' = ', HEXF:8 HEX );
      WRITELN(
'                         ZERO2 = ', E,  ' = ', HEXE:8 HEX );
      WRITELN(
'                  ZERO1 / ZERO2 = ', G,  ' = ', HEXG:8 HEX  );
{-------------------------------------------------------------------
          EXECUTE THE SAME CALCULATIONS IN DOUBLE PRECISION.
                                                                    }
      DH  := 1.0D0 / 2.0D0                                        ;
      DX  := 2.0D0 / 3.0D0  - DH                                  ;
      DY  := 3.0D0 / 5.0D0  - DH                                  ;

      DE  := ( DX + DX + DX )  - DH                               ;
      DF  := ( DY + DY + DY + DY + DY )  - DH                     ;
      DG  := 2.0D0 * DF / DE                                      ;

      HEXDE:  REAL16  := DE ;
      HEXDF:  REAL16  := DF ;
      HEXDG:  REAL16  := DG ;

      WRITELN;
      WRITELN(
'DOUBLE PRECISION KAHAN ZERO3 = ', DF,  ' = ', HEXDF.HALF1:8 HEX,
            HEXDF.HALF2:8 HEX );
      WRITELN(
'                         ZERO4 = ', DE,  ' = ', HEXDE.HALF1:8 HEX,
            HEXDE.HALF2:8 HEX );
      WRITELN(
'                  ZERO3 / ZERO4 = ', DG,  ' = ', HEXDG.HALF1:8 HEX,
            HEXDG.HALF2:8 HEX );

    END.
```

```
PROGRAM PMTEST;
{-------------------------------------------------------------------

          PROGRAM    P M T E S T

        MALCOLM'S  FLOATING POINT ANALYSIS PROGRAM.

        THIS ROUTINE ANALYTICALLY DETERMINES THE BASE, MANTISSA
        LENGTH, AND ROUNDING RULE FOR BOTH SINGLE & DOUBLE PRECISION.

-------------------------------------------------------------------}
VAR   Big_Integer, Increment, T, EPSILON   REAL;
      DBig_Integer, DIncrement, DT, DEPSILON   REAL(14);
      Base, DBase, Mlength, DMlength,   INTEGER;
      Sround, Dround   BOOLEAN;

BEGIN
      WRITELN;  WRITELN;
      WRITELN
('          MALCOLM''S FLOATING POINT ANALYZER [PM] PASCAL - 990/10');

    Big_Integer  := 2.0E0        ;
    Increment    := 2.0E0        ;

{         INCREASE Big_Integer UNTIL IT IS TOO BIG TO HOLD INTEGERS    }
    WHILE ( Big_Integer + 1.0E0 ) - Big_Integer = 1.0E0
       DO  Big_Integer  := Big_Integer * 2.0E0 ;

{         FIND THE SMALLEST NUMBER WHICH WILL INCREMENT Big_Integer    }
    WHILE Big_Integer + Increment = Big_Integer
       DO  Increment  := 2.0E0 * Increment ;

    Base  := ROUND(( Big_Integer + Increment ) - Big_Integer )    ;
    Sround  := DBig_integer + (Base-1.0E0) <> Big_Integer ;
    T  := 0.0E0                 ;
    Big_Integer := 1.0E0     ;

    REPEAT                          { How many digits in the mantissa ? }
       BEGIN
       Mlength  := Mlength + 1      ;
       Big_Integer  := Big_Integer * Base ;
       END
     UNTIL ( Big_Integer + 1.0E0 ) - Big_Integer <> 1.0E0  ;

{         COMPUTE THE INTERVAL SIZE FOR SINGLE PRECISION INTEGERS    }

     EPSILON  := 1 ;
     FOR I  := 1 TO Mlength DO
           EPSILON  := EPSILON / FLOAT(Base,7) ;
     IF Sround THEN EPSILON  := EPSILON / 2.0E0  ;

    WRITELN;
    WRITELN(
            '  PASCAL REAL(7) EXPONENT BASE =', Base:3 );
    IF Sround THEN WRITELN(
            '  MANTISSA LENGTH = ',Mlength:4,' DIGITS, ROUNDED ' )
           ELSE WRITELN(
            '  MANTISSA LENGTH = ',Mlength:4,' DIGITS, TRUNCATED ' );
    WRITELN(
            '  BEST PRECISION IS ',EPSILON,',  WORST PRECISION IS ',
            (EPSILON*BASE) );
{-------------------------------------------------------------------

       Perform the same analysis, using DOUBLE PRECISION variables.

                                                                    }
    DBig_Integer   := 2.0q0 ;
    DIncrement     := 2.0q0 ;

    WHILE  ( DBig_Integer + 1.0q0 ) - DBig_Integer = 1.0
       DO  DBig_Integer  := DBig_Integer * 2.0q0 ;

    WHILE DBig_Integer + DIncrement = DBig_Integer
       DO DIncrement  := 2.0q0 * DIncrement ;

    DBase  := ROUND(( DBig_Integer + DIncrement ) - DBig_Integer )    ;
    Dround  := Big_Integer + ( DBase - 1.0q0 ) <> DBig_Integer ;
    DT  := 0.0q0                ;
    DBig_Integer  := 1.0q0                ;

    REPEAT
      BEGIN
      DMlength  := DMlength + 1      ;
      DBig_Integer  := DBig_Integer * DBase              ;
      END
    UNTIL (DBig_Integer + 1.0q0 ) - DBig_Integer <> 1.0q0 ;

{         COMPUTE THE INTERVAL SIZE FOR SINGLE PRECISION INTEGERS    }

     DEpsilon  := 1 ;
     FOR I  := 1 TO Mlength DO
           DEpsilon  := DEpsilon / FLOAT(Base,14) ;
     IF Sround THEN DEpsilon  := DEpsilon / 2.0E0  ;

    WRITELN; WRITELN;
    WRITELN(
            '  PASCAL REAL(14) EXPONENT BASE =', DBase:3 );
    WRITELN;
    IF Dround THEN WRITELN(
          '  MANTISSA LENGTH = ', DMlength:3,' DIGITS, ROUNDED ' )
            ELSE WRITELN(
          '  MANTISSA LENGTH = ', DMlength:3,' DIGITS, TRUNCATED ' );
    WRITELN; WRITELN(
            '  BEST PRECISION IS ',DEpsilon,',  WORST PRECISION IS ',
            (DEpsilon*BASE) );

END.
```

```
PROGRAM PXTEST;
{
          PROGRAM   P X T E S T

          THIS ROUTINE TESTS THE EXPONENTIAL AND LOG FUNCTIONS
          OF THE 990                                          }

CONST TKLIM = 5000.0; IELIM = 40 ; THRESH = 5 ;

TYPE  REAL_DOUBLE = REAL(14) ;
      INTEGER_64BITS = RECORD
        LONGINT1 : LONGINT ;
        LONGINT2 : LONGINT
      END;

VAR   KERRS   ARRAY [1 .. 8] OF INTEGER;
      HEXD, HEXX, HEXV, HEXX2  : LONGINT;
      IECNT, DIFF, NBITS, N : INTEGER ;
      TOTAL_BITS, TOTAL_ERRORS : REAL(7) ;
      DNUM, VNUM, XNUM, XNUM2, PERCEN, TKOUNT, AVG_BIT_LOSS :REAL(7);
      DNUM2, VNUM2 :REAL(14);
      HEXV2 :INTEGER_64BITS;

BEGIN
      WRITELN; WRITELN(
        ' EXPONENTIAL TEST IN PASCAL (PX) -  TI 990/10');
      WRITELN; WRITELN(
        '  ORIGINAL NUMBER              ',
        '              MANTISSA' );
      WRITELN(
        '  DECIMAL   REAL(7)     LN(N)  ',
        '     EXP(LN(N)) BITS LOST' );
      WRITELN ;

      TKOUNT := 0.0     ;
      IECNT  := 0       ;
      DNUM   := 1.0     ;

{--------- MAJOR FUNCTIONAL LOOP OF PROGRAM  ----------------------}

      REPEAT
      TKOUNT := TKOUNT + 1.0    ;
      DNUM   := DNUM + 1.0      ;
      VNUM   := LN( DNUM )      ;  { CONVERT TO LOGARITHM  }
      XNUM   := EXP( VNUM )     ;  { CONVERT BACK TO NATURAL NUMBER }
      IF  DNUM <> XNUM THEN
         BEGIN

{----- ERROR DETECTED   DETERMINE HOW SEVERE IT IS.  ----------------}

         HEXX := REAL  := XNUM ;
         HEXD := REAL  := DNUM ;
         DIFF  := ABS( HEXX - HEXD );

         IF DIFF <=  #1 THEN NBITS  := 1 ELSE
          IF DIFF <=  #3 THEN NBITS  := 2 ELSE
           IF DIFF <=  #7 THEN NBITS  := 3 ELSE
            IF DIFF <=  #F THEN NBITS  := 4 ELSE
             IF DIFF <= #1F THEN NBITS  := 5 ELSE
              IF DIFF <= #3F THEN NBITS  := 6 ELSE
               IF DIFF <= #7F THEN NBITS  := 7 ELSE
                NBITS  := 8 ;
         KERRS(NBITS) := KERRS(NBITS) + 1 ; { STATISTICAL TALLY }

{----- PRINT OUT THE FIRST FEW SEVERE ABBERATIONS.  ----------------}

          IF  IECNT <= IELIM  AND  NBITS >= THRESH  THEN
            BEGIN
            ONUM2  := ONUM ;
            HEXV := REAL := VNUM ;

            WRITELN(
            ' ',ONUM:7:0,'   ',HEXD:8 HEX,'  ',HEXV:8 HEX,' ',
            '    ', HEXX:8 HEX, '     ', NBITS:2 );
            IECNT := IECNT + 1      ;
            END
           END

      UNTIL TKOUNT >= TKLIM  ;


{---------- GENERATE A SUMMARY ERROR REPORT.  --------------------}

      WRITELN; WRITELN; WRITELN(
              '          ** SUMMARY OF ERRORS **' ) ;
      WRITELN; WRITELN('          BIT LOSS    COUNT' );

      TOTAL_ERRORS := 0 ;
      TOTAL_BITS   := 0 ;

      FOR N  := 1 TO 8 DO
      IF  KERRS(N) <> 0    THEN
         BEGIN
         TOTAL_ERRORS := TOTAL_ERRORS + KERRS(N);
         TOTAL_BITS   := TOTAL_BITS + N * KERRS(N) ;
         WRITELN(
                 '          ', N:5, '     ', KERRS(N):5 );
         END ;

    PERCEN  :=  TOTAL_ERRORS / TKOUNT * 100.0 ;
    WRITELN('      TOTAL     ',
       TOTAL_ERRORS:5:0,  ' ERRORS  IN',TKOUNT:6:0,  ' TESTS  ');
    WRITELN; WRITELN(
    '              ',PERCEN:6:1,' PER CENT OF NUMBERS HAD ERRORS ' );
    AVG_BIT_LOSS := TOTAL_BITS / TOTAL_ERRORS ;
    WRITELN ; WRITELN(
    '          AVERAGE BITS LOST PER ERROR =', AVG_BIT_LOSS:6:2 );

END.
```

APPENDIX B — FORTRAN TEST CODES

```
C
C          PROGRAM   G 2 T E S T
C
C          THIS ROUTINE TESTS THE FORTRAN SQUARE ROOT AND
C          SQUARING FUNCTIONS OF THE 990.
C
C
      INTEGER KERRS(30)
      INTEGER DIFF, THRESH, IONUM(2), IXNUM(2), I2NUM(2)
      INTEGER * 4 L2NUM, LXNUM
      DATA KERRS / 30 * 0 /
C
      EQUIVALENCE (IXNUM,XNUM), (IONUM,ONUM)
      EQUIVALENCE (LXNUM,XNUM), (I2NUM, L2NUM)
C
      WRITE(6,1001)
 1001 FORMAT(//,8X,'SINGLE PRECISION EXPONENT TEST (G2) TI 990/10')
      WRITE(6,1002)
 1002 FORMAT( //,  '  ORIGINAL NUMBER    N**1/3',
     *          '    (N**1/3)**3     MANTISSA ' )
      WRITE(6,1003)
 1003 FORMAT( '   DECIMAL    REAL                ',
     *                '                BITS LOST' )
C
      TKOUNT = 0.0
      TKLIM  = 5000.
      IECNT = 0
      IELIM = 35
      THRESH = 6
      ONUM = 1.0
C
C          MAJOR FUNCTIONAL LOOP OF PROGRAM.
C
  100 CONTINUE
      TKOUNT = TKOUNT + 1.0
      ONUM = ONUM + 1.0
      VNUM = ONUM **(0.3333333)
      XNUM = VNUM ** 3.0
      IF( ONUM .EQ. XNUM ) GO TO 200
C
C          ERROR DETECTED.   IF IT IS LARGE ENOUGH, PRINT IT OUT
C
      K2NUM(1) = LAND(IONUM(1),>00FF )
      K2NUM(2) = IONUM(2)
      IXNUM(1) = LAND( IXNUM(1), >00FF )
      DIFF = LXNUM - L2NUM
      IF ( DIFF .LT. 0 ) DIFF = -DIFF
      NBITS = 1
      IF( DIFF .GE.    2 ) NBITS = 2
      IF( DIFF .GE.    4 ) NBITS = 3
      IF( DIFF .GE.    8 ) NBITS = 4
      IF( DIFF .GE.   16 ) NBITS = 5
      IF( DIFF .GE.   32 ) NBITS = 6
      IF( DIFF .GE.   64 ) NBITS = 7
      IF( DIFF .GE.  128 ) NBITS = 8
      IF( DIFF .GE.  256 ) NBITS = 9
      IF( DIFF .GE.  512 ) NBITS = 10
      IF( DIFF .GE. 1024 ) NBITS = 11
      IF( DIFF .GE. 2048 ) NBITS = 12
      IF( DIFF .GE. 4096 ) NBITS = 13
C
      KERRS(NBITS) = KERRS(NBITS) + 1


      IF( NBITS .LT. THRESH ) GO TO 200
C
C          PRINT OUT THE FIRST FEW SERIOUS ERRORS
C
      IF( IECNT .GE. IELIM ) GO TO 200
      XNUM = VNUM ** 3.0
      WRITE(6,1010) ONUM,ONUM, VNUM, XNUM, NBITS
 1010 FORMAT( 1X,F7.0, 2X,Z8, 2X,Z8, 3X,Z8, 4X,I2 )
      IECNT = IECNT + 1
C
  200 CONTINUE
      IF( TKOUNT .LT. TKLIM ) GO TO 100
C
C
C          GENERATE A SUMMARY ERROR REPORT
C
  998 CONTINUE
      WRITE(6,9060)
 9060 FORMAT(///, 10X,'** SUMMARY OF ERRORS **', //,10X,
     *        'BIT LOSS   COUNT' )
C
      ITOTAL = 0
      TBITS  = 0.0
C
      DO 610 N = 1, 30
      IF( KERRS(N) .EQ. 0 ) GO TO 610
      ITOTAL = ITOTAL + KERRS(N)
      TBITS  = TBITS + N * KERRS(N)
      WRITE(6,9061) N, KERRS(N)
 9061 FORMAT( 10X,I5, 4X,I5 )
  610 CONTINUE
C
      PERCEN =  ITOTAL / TKOUNT * 100.0
      WRITE(6,9065) ITOTAL, TKOUNT, PERCEN
 9065 FORMAT(/, 10X,'TOTAL',4X, I5,' ERRORS   IN', I6,' TESTS.',/,
     *        10X, F6.2,' PER CENT OF NUMBERS HAD ERRORS.' )
C
      AVGBIT = TBITS / TKOUNT
      WRITE(6,9067) AVGBIT
 9067 FORMAT(/,10X,'AVERAGE BITS LOST PER ERROR = ', F6.2 )
C
      STOP
      END
```

```
C
C            PROGRAM   S 2 T E S T
C
C
C       THIS ROUTINE TESTS FLOATING POINT SUBTRACTION WITH
C       NUMBERS BELOW THE SIGNIFICANCE THRESHOLD
C
C
      REAL * 4 ONE
      REAL * 4 TNUM(2)
      REAL * 4 DECR(6)
      INTEGER *4 NONE, NDECR, NONEX
      INTEGER *4 KFACT, TRUNC
      DATA KFACT / >10000000 /
      DATA TNUM  / 1 0, 1 01 /
      DATA DECR(1) / >3E100000 /
      DATA DECR(2) / >3D100000 /
      DATA DECR(3) / >3C100000 /
      DATA DECR(4) / >3B100000 /
      DATA DECR(5) / >3A100000 /
      DATA DECR(6) / >39100000 /
      DATA NSAMPS / 6 /
C
      DO 501 M=1,2
      WRITE(6,1000)
 1000 FORMAT(///,10X,'SUBTRACTION SIGNIFICANCE TEST (S2) - 990/100',/)
      WRITE(6,1009)
 1009 FORMAT(/,10X,'REAL *4',4X,'DENORMALIZED FRACTION',4X,
     *    'DECIMAL' )
      WRITE(6,1008)
 1008 FORMAT(10X,'INTERNAL',9X,'HEX * 16**8',8X,'EQUIVALENT' )
      DO 500 N=1,NSAMPS
      ONE  = TNUM(M)
      ONEX = ONE - DECR(N)
C
      NONE  = (ONE-1 0) * KFACT * 16
      NDECR = DECR(N) * KFACT   * 16
      NONEX = ONEX * KFACT      * 16
      LONEX = 0
      IF( ONEX .GE. 1 0 ) LONEX = 1
C
      TRUNC = '  OK'
      IF( (ONEX + DECR(N)) .NE. ONE ) TRUNC = '
      IF( ONEX .EQ. ONE )  TRUNC = 'NOOP'
C
      WRITE(6,1001) N, ONE, NONE, ONE
 1001 FORMAT( /,' ',13,5X, Z9, 8X,'1 ',Z8, 10X,      F12 10)
      WRITE(6,1002) DECR(N), NDECR, DECR(N)
 1002 FORMAT(   8X,     '-',Z9, 4X,'= 0 ',Z8, 9X, '-', F12.10)
      WRITE(6,1003) ONEX, LONEX, NONEX, ONEX, TRUNC
 1003 FORMAT(   8X,     '=',Z9, 4X,'=',I2,' ',Z8,9X,'=', F12.10, 2X, A4 )
C
  500 CONTINUE
C
      WRITE(6,2000)
 2000 FORMAT(' ')
  501 CONTINUE
C
      STOP
      END
```

PASCAL DECIMAL CONVERSION TEST (PC) - TI 990/10

ORIGINAL DECIMAL (7 DIGITS)	TRUNCATED TO REAL(7)	UNFORMAT WRITE	FORMAT WRITE 8.7	ROUNDED TO DECIMAL INTERNALLY	TRUNCATED TO DECIMAL INTERNALLY
0.2443518	403E8DD6	2.443517E-01	0.2443517	0.2443517	0.2443517
0.1796208	402DFBA0	1.796207E-01	0.1796207	0.1796207	0.1796207
0.1621077	40297FE3	1.621076E-01	0.1621076	0.1621076	0.1621076
0.1540023	40276CB1	1.540022E-01	0.1540022	0.1540022	0.1540022
0.1463021	40257400	1.463020E-01	0.1463020	0.1463020	0.1463020
0.0970596	4018D8E5	9.705955E-02	0.0970595	0.0970595	0.0970595
0.0875962	40166CB4	8.759618E-02	0.0875962	0.0875961	0.0875961
0.0832163	40154DA9	8.321625E-02	0.0832162	0.0832162	0.0832162
0.0790554	40143CF9	7.905537E-02	0.0790554	0.0790553	0.0790553
0.0751026	401339EC	7.510257E-02	0.0751026	0.0751025	0.0751025
0.0713474	401243D2	7.134734E-02	0.0713474	0.0713473	0.0713473
0.0677800	40115A07	6.777996E-02	0.0677800	0.0677799	0.0677799
0.0643910	40107BED	6.439096E-02	0.0643910	0.0643909	0.0643909
0.0087096	3F23ACAD	8.709598E-03	0.0087096	0.0087095	0.0087095
0.0082741	3F21E405	8.274097E-03	0.0082741	0.0082740	0.0082740
0.0078603	3F20321F	7.860299E-03	0.0078603	0.0078602	0.0078602
0.0074672	3F1E95ED	7.467199E-03	0.0074672	0.0074671	0.0074671
0.0070938	3F1D0E63	7.093798E-03	0.0070938	0.0070937	0.0070937
0.0067391	3F1B9A75	6.739099E-03	0.0067391	0.0067390	0.0067390
0.0064021	3F1A3916	6.402098E-03	0.0064021	0.0064020	0.0064020
0.0060819	3F18E955	6.081898E-03	0.0060819	0.0060818	0.0060818
0.0057778	3F17AA76	5.777799E-03	0.0057778	0.0057777	0.0057777
0.0054889	3F167887	5.488899E-03	0.0054889	0.0054888	0.0054888
0.0052144	3F155BB1	5.214397E-03	0.0052144	0.0052143	0.0052143
0.0049536	3F144A39	4.953597E-03	0.0049536	0.0049535	0.0049535
0.0047059	3F13467E	4.705898E-03	0.0047059	0.0047058	0.0047058
0.0044706	3F124FC3	4.470598E-03	0.0044706	0.0044705	0.0044705
0.0042470	3F11654D	4.246999E-03	0.0042470	0.0042469	0.0042469
0.0040346	3F108695	4.034597E-03	0.0040346	0.0040345	0.0040345
0.0009109	3E38B25D	9.108998E-04	0.0009109	0.0009108	0.0009108
0.0007809	3E332D53	7.808998E-04	0.0007809	0.0007808	0.0007808
0.0007418	3E309D56	7.417998E-04	0.0007418	0.0007417	0.0007417
0.0007047	3E2E2EE7	7.046999E-04	0.0007047	0.0007046	0.0007046
0.0006359	3E29ACA1	6.358998E-04	0.0006359	0.0006358	0.0006358
0.0006041	3E279710	6.040998E-04	0.0006041	0.0006040	0.0006040
0.0005738	3E259AC4	5.737999E-04	0.0005738	0.0005737	0.0005737
0.0005451	3E23B942	5.450998E-04	0.0005451	0.0005450	0.0005450
0.0004919	3E203CB6	4.918999E-04	0.0004919	0.0004918	0.0004918
0.0004673	3E1E9FFE	4.672999E-04	0.0004673	0.0004672	0.0004672
0.0004439	3E1D1767	4.438998E-04	0.0004439	0.0004438	0.0004438
0.0004217	3E1BA2F3	4.216998E-04	0.0004217	0.0004216	0.0004216
0.0004006	3E1A40F3	4.005998E-04	0.0004006	0.0004005	0.0004005
0.0003433	3E167F9E	3.432999E-04	0.0003433	0.0003432	0.0003432
0.0003261	3E155F0C	3.260998E-04	0.0003261	0.0003260	0.0003260
0.0003097	3E144BE7	3.096999E-04	0.0003097	0.0003096	0.0003096
0.0002942	3E1347D8	2.941999E-04	0.0002942	0.0002941	0.0002941
0.0002794	3E124F8D	2.793998E-04	0.0002794	0.0002793	0.0002793
0.0002654	3E1164AC	2.653999E-04	0.0002654	0.0002653	0.0002653
0.0002521	3E108589	2.520999E-04	0.0002521	0.0002520	0.0002520
0.0000995	3D6B5553	9.949999E-05	0.0000995	0.0000994	0.0000994
0.0000945	3D631726	9.449999E-05	0.0000945	0.0000944	0.0000944
0.0000852	3D5956B3	8.519999E-05	0.0000852	0.0000851	0.0000851
0.0000809	3D54D46D	8.089999E-05	0.0000809	0.0000808	0.0000808
0.0000768	3D5087D7	7.679999E-05	0.0000768	0.0000767	0.0000767
0.0000729	3D4C70F1	7.289999E-05	0.0000729	0.0000728	0.0000728
0.0000692	3D488FB8	6.919999E-05	0.0000692	0.0000691	0.0000691
0.0000657	3D44E435	6.569999E-05	0.0000657	0.0000656	0.0000656
0.0000624	3D416E3F	6.239999E-05	0.0000624	0.0000623	0.0000623
0.0000562	3D3AEE12	5.619999E-05	0.0000562	0.0000561	0.0000561
0.0000506	3D350ED5	5.059999E-05	0.0000506	0.0000505	0.0000505
0.0000480	3D3254E6	4.799999E-05	0.0000480	0.0000479	0.0000479
0.0000411	3D2B1BB2	4.109999E-05	0.0000411	0.0000410	0.0000410
0.0000390	3D2BE4FB	3.899999E-05	0.0000390	0.0000389	0.0000389
0.0000370	3D26CC1C	3.699999E-05	0.0000370	0.0000369	0.0000369
0.0000351	3D24CE15	3.509999E-05	0.0000351	0.0000350	0.0000350
0.0000333	3D22EAE6	3.329999E-05	0.0000333	0.0000332	0.0000332
0.0000316	3D21228F	3.159999E-05	0.0000316	0.0000315	0.0000315
0.0000270	3D1C4FC1	2.699999E-05	0.0000270	0.0000269	0.0000269
0.0000256	3D1AD7F2	2.559999E-05	0.0000256	0.0000255	0.0000255
0.0000230	3D1B1E03	2.299999E-05	0.0000230	0.0000229	0.0000229
0.0000218	3D16DBE4	2.179999E-05	0.0000218	0.0000217	0.0000217
0.0000196	3D148D55	1.959999E-05	0.0000196	0.0000195	0.0000195
0.0000176	3D127476	1.759999E-05	0.0000176	0.0000175	0.0000175
0.0000158	3D109147	1.579999E-05	0.0000158	0.0000157	0.0000157
0.0000096	3CA1DFAF	9.599999E-06	0.0000096	0.0000095	0.0000095
0.0000091	3C98AC33	9.099999E-06	0.0000091	0.0000090	0.0000090
0.0000086	3C904BB7	8.599999E-06	0.0000086	0.0000085	0.0000085
0.0000072	3C78CBC3	7.199999E-06	0.0000072	0.0000071	0.0000071
0.0000060	3C64A9CD	5.999999E-06	0.0000060	0.0000059	0.0000059
0.0000054	3C5A9BD2	5.399999E-06	0.0000054	0.0000053	0.0000053
0.0000048	3C5087D7	4.799999E-06	0.0000048	0.0000047	0.0000047
0.0000042	3C4A7ADC	4.199999E-06	0.0000042	0.0000041	0.0000041
0.0000035	3C3ABB62	3.499999E-06	0.0000035	0.0000034	0.0000034
0.0000029	3C30A7E7	2.899999E-06	0.0000029	0.0000028	0.0000028
0.0000023	3C269660	2.299999E-06	0.0000023	0.0000022	0.0000022
0.0000018	3C1E32F0	1.799999E-06	0.0000018	0.0000017	0.0000017
0.0000017	3C1CB571	1.699999E-06	0.0000017	0.0000016	0.0000016
0.0000016	3C1AD7F2	1.599999E-06	0.0000016	0.0000015	0.0000015
0.0000012	3C1421F5	1.199999E-06	0.0000012	0.0000011	0.0000011
0.0000011	3C127476	1.099999E-06	0.0000011	0.0000010	0.0000010
0.0000010	3C10C6F7	9.999994E-07	0.0000010	0.0000009	0.0000009

PASCAL REAL NUMBER INVERSION TEST (PD) - TI 990/10

ORIGINAL DECIMAL	NUMBER REAL	RECIPROCAL REAL	1/(1/N) REAL	MANTISSA BITS LOST
13	41E00000	40124924	41E00007	3
171	42AB0000	3F17F405	42AB0007	3
194	42C20000	3F151D07	42C20008	4
199	42C70000	3F149539	42C70008	4
201	42C90000	3F1460CB	42C90007	3
206	42CE0000	3F13E22C	42CE0007	3
211	42D30000	3F13698D	42D3000A	4
212	42D40000	3F13521C	42D4000A	4
216	42D80000	3F12F684	42D80008	4
217	42D90000	3F12E025	42D90008	4
219	42DB0000	3F12B404	42DB0007	3
222	42DE0000	3F127350	42DE0008	4
224	42E00000	3F124924	42E00007	3
230	42EA0000	3F11CF06	42EA0008	4
235	42EB0000	3F116E06	42EB0007	3
239	42EF0000	3F112358	42EF000C	4
243	42F30000	3F10DB20	42F30009	4
245	42F50000	3F1087E6	42F5000D	4
250	42FA0000	3F10624D	42FA000C	4
251	42FB0000	3F105197	42FB000E	4
253	42FD0000	3F103091	42FD000B	4
254	42FE0000	3F102040	42FE0007	3

** SUMMARY OF ERRORS **

BIT LOSS	COUNT
1	29
2	49
3	30
4	14
TOTAL	122 ERRORS IN 300. TESTS

40.66 PER CENT OF NUMBERS HAD ERRORS.

AVERAGE BITS LOST PER ERROR = 2.23

KAHAN'S PROBLEM - (X1) - 990/10

	DECIMAL	INTERNAL
SINGLE PRECISION KAHAN ZERO1 =	-1.788139E-07	= BB300000
ZERO2 =	-1.192093E-07	= BB200000
ZERO1 / ZERO2 =	3.000000E+00	= 41300000
DOUBLE PRECISION KAHAN ZERO3 =	-4.163336342344337E-17	= B330000000000000
ZERO4 =	-2.775557561562891E-17	= B320000000000000
ZERO3 / ZERO4 =	3.000000000000000E+00	= 4130000000000000

MALCOLM'S FLOATING POINT ANALYZER (PM) PASCAL - 990/10

```
PASCAL REAL(7) EXPONENT BASE = 16
MANTISSA LENGTH =     6 DIGITS, ROUNDED.
BEST PRECISION IS  2.980232E-08.  WORST PRECISION IS  4.768372E-07

 PASCAL REAL(14) EXPONENT BASE = 16
MANTISSA LENGTH =  14 DIGITS, ROUNDED.
```

EXPONENTIAL TEST IN PASCAL (PX) - TI 990/10

ORIGINAL DECIMAL	NUMBER REAL(7)	LN(N)	EXP(LN(N))	MANTISSA BITS LOST
84.	42540000	4146E4A0	42540037	6
179.	42830000	4152FFB8	42830072	7
181.	42850000	41532D08	42850073	7
630.	43276000	416721A8	4327601A	5
633.	43279000	41673520	4327901A	5
661.	43295000	4167E66A	43295018	5
663.	43297000	4167F2CA	43297018	5
670.	4329E000	41681DCF	4329E018	5
701.	432BD000	4168D712	432BD01C	5
706.	432C2000	4168F42F	432C201C	5
714.	432CA000	41692256	432CA01D	5
716.	432CC000	41692DC8	432CC01C	5
724.	432D4000	41695B4E	432D401D	5
1282.	43502000	41727FB3	43502035	6
1308.	4351C000	4172D1F0	4351C036	6
1324.	43520000	417303BD	43520037	6
1344.	43540000	41734126	43540037	6
1351.	43547000	4173566D	43547037	6
1352.	43548000	41735975	43548037	6
1362.	43552000	417377A4	43552037	6
1370.	4355A000	41738FA1	4355A037	6
1375.	4355F000	41739E8D	4355F037	6
1384.	43568000	4173B946	43568037	6
1391.	4356F000	4173CDF0	4356F037	6
1393.	43571000	4173D3D3	43571039	6
1395.	43573000	4173D9B3	43573037	6
1402.	4357A000	4173EE34	4357A039	6
1405.	4357D000	4173F6F5	4357D038	6
1417.	43589000	417419CB	43589039	6
1426.	43592000	417433BA	43592039	6
1427.	43593000	41743699	43593039	6
1430.	43596000	41743F33	43596039	6
1431.	43597000	41744210	43597039	6
1434.	4359A000	41744AA4	4359A039	6
1785.	436F9000	4177CB79	436F9010	5
1791.	436FF000	4177D93B	436FF012	5
1801.	43709000	4177F006	43709010	5
2487.	439B7000	417D19F0	439B7069	7
2520.	439D8000	417D4FEE	439D8069	7
2532.	439E4000	417D6363	439E4068	7
2571.	43A08000	417DA1FF	43A08069	7

** SUMMARY OF ERRORS **

BIT LOSS	COUNT
1	1168
2	1272
3	838
4	50
5	26
6	29
7	31

TOTAL 3414 ERRORS IN 5000 TESTS.

68.2 PER CENT OF NUMBERS HAD ERRORS.
AVERAGE BITS LOST PER ERROR = 2.03

SINGLE PRECISION EXPONENT TEST (Q2) TI 990/10

ORIGINAL DECIMAL	NUMBER REAL	N**1/3	(N**1/3)**3	MANTISSA BITS LOST
168.	42A80000	41584918	42A8006E	7
169.	42A90000	415875DC	42A9006D	7
173.	42AD0000	41592722	42AD006E	7
179.	42B30000	415A2BFE	42B30072	7
181.	42B50000	415A81A6	42B50073	7
239.	42EF0000	41634B0B	42EF0030	6
241.	42F10000	416391C0	42F1002A	6
242.	42F20000	416384F5	42F2002B	6
246.	42F60000	416440D3	42F60030	6
251.	42FB0000	4164ED8A	42FB003D	6
1110.	43456000	41A5AA63	43456091	8
1171.	43493000	41ABA5AF	43493090	8
1260.	434EC000	41ACD02A	434EC033	6
1266.	434F2000	41AD1647	434F2034	6
1282.	43502000	41ADD02A	43502034	6
1308.	4351C000	41AEF86C	4351C0A4	8
1322.	4352A000	41AF9A36	4352A036	6
1324.	4352C000	41AFB0DE	4352C035	6
1326.	4352E000	41AFC7B1	4352E035	6
1334.	43536000	41B02259	4353608D	8
1340.	4353C000	41B06552	4353C036	6
1352.	43548000	41B0EBB7	43548036	6
1362.	43552000	41B15B1C	43552037	6
1363.	43553000	41B1663B	43553037	6
1370.	4355A000	41B183D4	4355A036	6
1375.	4355F000	41B1EB1B	4355F037	6
1384.	43568000	41B24E44	43568037	6
1389.	4356D000	41B2852B	4356D037	6
1391.	4356F000	41B29B1B	4356F038	6
1393.	43571000	41B2B173	435710A7	8
1395.	43573000	41B2C6E1	43573038	6
1402.	4357A000	41B3134D	4357A037	6
1405.	4357D000	41B33470	4357D0A8	8
1412.	43584000	41B3B00A	43584038	6
1417.	43589000	41B3B637	43589038	6

** SUMMARY OF ERRORS **

BIT LOSS	COUNT
1	973
2	1285
3	1663
4	489
5	60
6	61
7	63
8	10

TOTAL 4604 ERRORS IN 5000 TESTS.
92.08 PER CENT OF NUMBERS HAD ERRORS.

AVERAGE BITS LOST PER ERROR = 2.33

SUBTRACTION SIGNIFICANCE TEST (S2) - 990/100

	REAL ** INTERNAL	DENORMALIZED FRACTION HEX * 16**6	DECIMAL EQUIVALENT	
1	41100000	1.00000000	1.0000000000	
	- 3E100000	- 0.00100000	- .0002441406	
	= 40FFF000	= 0.FFF00000	= .9997558594	OK
2	41100000	1.00000000	1.0000000000	
	- 3D100000	- 0.00010000	- .0000610352	
	= 40FFFF00	= 0.FFFF0000	= .9998779419	OK
3	41100000	1.00000000	1.0000000000	
	- 3C100000	- 0.00001000	- .0000009537	
	= 40FFFFF0	= 0.FFFFF000	= .9999990463	OK
4	41100000	1.00000000	1.0000000000	
	- 3B100000	- 0.00000100	- .0000000596	
	= 40FFFFFF	= 0.FFFFFF00	= .9999999404	OK
5	41100000	1.00000000	1.0000000000	
	- 3A100000	- 0.00000010	- .0000000037	
	= 40FFFFFF	= 0.FFFFFF00	= .9999999404	
6	41100000	1.00000000	1.0000000000	
	- 39100000	- 0.00000001	- .0000000002	
	= 41100000	= 1.00000000	=1.0000000000	NOOP

SUBTRACTION SIGNIFICANCE TEST (S2) - 990/100

	REAL ** INTERNAL	DENORMALIZED FRACTION HEX * 16**6	DECIMAL EQUIVALENT	
1	411028F6	1.028F6000	1.0100002289	
	- 3E100000	- 0.00100000	- .0002441406	
	= 411027F6	= 1.027F6000	=1.0097560883	OK
2	411028F6	1.028F6000	1.0100002289	
	- 3D100000	- 0.00010000	- .0000610352	
	= 411128F6	= 1.128F6000	=1.0097656531	OK
3	411028F6	1.028F6000	1.0100002289	
	- 3C100000	- 0.00001000	- .0000009537	
	= 411028F5	= 1.028F5000	=1.0099992752	OK
4	411028F6	1.028F6000	1.0100002289	
	- 3B100000	- 0.00000100	- .0000000596	
	= 411028F5	= 1.028F5000	=1.0099992752	
5	411028F6	1.028F6000	1.0100002289	
	- 3A100000	- 0.00000010	- .0000000037	
	= 411028F5	= 1.028F5000	=1.0099992752	
6	411028F6	1.028F6000	1.0100002289	
	- 39100000	- 0.00000001	- .0000000002	
	= 411028F6	= 1.028F6000	=1.0100002289	NOOP

CALENDAR OF EVENTS

. . . schedule information on upcoming events of interest throughout the EG engineering community. These events may include seminars, meetings, product demonstrations, training classes, conferences, and colloquiums. Activities of local technical societies will also be featured as available; because this information is difficult to obtain, members of the various societies are asked to help. Please provide the name and telephone extension of a contact, the name of the event (as well as the date, time, and location of occurrence), and a brief description of the event. Mail to: Editor, MS 3407, or call Tom Calvert, X868-4871.

Professional Organizations Schedules

Members of professional organizations such as IEEE, ACM, ASME, ASQC, etc., are encouraged to submit a schedule of upcoming events for their particular organizations. Send professional organization schedule information to Journal Editor, MS 3407.

IEEE Microwave Theory and Techniques Group All IEEE Microwave Theory and Techniques Group meetings are held at Hewlett Packard, 201 E. Araphaho Road, Richardson, Texas. The meetings are preceded by a speaker's dinner at 6:15 p.m. at the Steak and Ale Restaurant at LBJ and Abrams Road. Direct questions concerning the IEEE Microwave Theory and Techniques Group activities and meeting schedule to Jim Griffin, Dallas Chapter Chairman, at X83-5977, MS 96.

American Society for Quality Control (ASQC)

The ASQC holds regular monthly meetings. Persons interested in attending may contact Bob Stalcup, X865-7262, MS 3154, for schedule and meeting place information.

Association for Computing Machinery

The Dallas Chapter of the ACM holds monthly dinner meetings which usually feature a presentation of interest to computer professionals. The meetings are open to the public and the lecture — presentation portion is free. The location changes each month, but is usually held at a local restaurant or hotel.

The details of a particular meeting can be obtained by entering:

T ACM.MEETINGS.79."MONTH" "MONTH" = 01, 02, 03, 04, etc.

TI representatives to the local ACM are: Marc Parks (North Bldg.)—X83-5105 and Don Christian (South Bldg.), X83-5051.

Electronic Connector Study Group

The Electronic Connector Study Group (ECSG) has a dynamic program planned for the fall and winter months. Sponsors for the upcomming meetings are as follows:

October 1979 — ITT Cannon
November 1979 — Molex
December 1979 — Hollingsworth

The ECSG welcomes nonmenbers to all of their meetings. Contact Bob McCurley at X865-7232 for the time and place of the future meetings.

Equipment Group Training and Development Courses

Supervisors' Training Program, Phase I	Sep 11 - Oct 23	3:00 5:00 p.m.
	Oct 9 - Nov 20	12:30-2:30 p.m.
	Oct 30 - Dec 13	8:15 - 10:15 a.m.
Supervisors' Training Program, Phase II	Sep 5 - Oct 12	8:15 - 10:15 p.m.
	Oct 22 - Nov 28	3:00 - 5:00 p.m.
	Nov 14 - Dec 21	8:15 - 10:15 a.m.
Supervisors' Training Program, Phase III	Oct 9 - Oct 25	8:15 - 10:15 a.m.
	Nov 5 - Nov 21	10:30 - 12:30 p.m.
	Dec 4 - Dec 20	3:00 - 5:00 p.m.
Cost Management	Sep 26 - Oct 19	1:00 - 4:00 p.m.
	Oct 24 - Nov 16	1:00 - 4:00 p.m.
Effective Negotiations	Oct 1 - Oct 22	8:15 - 10:15 a.m.
Product Design	Oct 24 & Oct 26	8:15 a.m. - 12:15 p.m.
Engineering Orientation	Nov 15	8:30 a.m. - 2:30 p.m.
	Dec 13	8:30 a.m. - 2:30 p.m.

Research Colloquium

Colloquiums presented by the Central Research Laboratories (CRL) are coordinated and scheduled by Mooshi Namordi. X83-3626.

Announcements of upcoming colloquiums are mailed from a list maintained by Mooshi Namordi; you may contact him to have your name placed on the announcement list.

Seminars at TI Expressway Site

To receive an announcement of all seminars scheduled, contact Lois Banks at X83-4860 or Chuck Braun at X83-2473. They will gladly place your name on the seminar announcement distribution list.

AST Software Education

AST software education classes are scheduled throughout 1979. To view the near-future schedule, use the inquiry **T AST** on any TI 914/914A Video Display Terminal in the IMS mode.

In Future Issues . . .

Radar Signal Processing Development for Application of VHSI

Computer Designed Impulse Multipliers

Program Cost and Price Analysis on a TI-59 Calculator

Flat Plate Antennas

Proportional Drive for Power Transistors

Review of Programmable Calculator Books

Linear Microelectronics Design and Development Branch

Automated Mixer Matching Measurements

Non-Linear FM Pulse Conpression

The Design of Broadband Microwave Amplifiers

Water Soluble Fluxes — Can the Military Use Them?

Complexities of Capturing and Analyzing Logic Data in a Real-Time Environment

EG ENGINEERING JOURNAL
MAIL STATION 3407
ATTENTION: EDITOR

ADDRESS CHANGE REQUESTED
IF UNDELIVERABLE,
RETURN TO MAIL STATION 3407