ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439-4801

ANL/MCS-TM--138

DE90 014932

ANL/MCS-TM-138

ELEFUNT Test Results under AST Fortran V1.8.0 on the Sequent Symmetry

by

W. J. Cody

Mathematics and Computer Science Division

Technical Memorandum No. 138

July 1990

This work was supported by the Applied Mathematical subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Contents

ELEFUNT Test Results under AST Fortran V1.8.0 on the Sequent Symmetry

by

W. J. Cody

Abstract

This report discusses testing of the floating-point arithmetic and of the elementary function libraries under AST Fortran on a 24-processor Sequent Symmetry computer. The programs MACHAR and PARANOIA were used to check the quality of arithmetic, and the ELEFUNT suite of programs from the book *Software Manual for the Elementary Functions* by Cody and Waite was used to check function performance. Two complete sets of tests were run, one for each type of floating-point processor, Intel 80387 and Weitek 1167, on the machine.

1. Introduction

The Environmental Assessment and Information Sciences Division at Argonne National Laboratory acquired a 24-processor Sequent Symmetry computer in early 1989. The machine provides two different implementations of IEEE-style floating-point arithmetic [IEEE 1985] in Intel 80387 and Weitek 1167 chips. Sequent ATS Fortran V1.8.0 is the default Fortran compiler on this machine, but a second compiler, Sequent Fortran V3.2, is also available on request. This report summarizes and analyzes the results of various tests of the arithmetic and of the Fortran elementary and intrinsic function packages using the ATS compiler (under DYNIX 3.0.17.9 NFS) and both the Intel and the Weitek arithmetic systems. The test results are valid only for the floating-point environment (the particular combination of compiler and floating-point hardware) under which they were run. Hereafter we will use the terms Intel/Weitek to mean the floating-point environment consisting of the ATS compiler and that chip. We abandoned plans to run similar tests under the Sequent Fortran V3.2 compiler when preliminary runs uncovered serious problems not encountered with the ATS compiler.

The next section discusses the computer arithmetic as analyzed by MACHAR [Cody 1988a] and PARANOIA [Karpinski 1985]. Section 3 discusses test results for the elementary and intrinsic functions obtained with the ELEFUNT [Cody and Waite 1980] and INTFUNT test suites. Section 4 summarizes our findings.

This report is one of a continuing series of reports on the quality of the arithmetic and Fortran libraries available on machines at Argonne National Laboratory [Cody 1986a, 1986b, 1986c, 1986d, 1988b, and 1989].

2. The Arithmetic

MACHAR is an evolving subroutine to dynamically determine fundamental parameters for the floating-point arithmetic system. Because it probes the dusty corners of the system, MACHAR is extremely sensitive to compiler optimizations that have no noticeable effect on most other programs. In particular, it seriously malfunctions unless critical intermediate results are stored at working precision.

Table 1. Machine Parameters for i80387 as Determined by MACHAR

‡ denotes results for Weitek 1167

Parameter		Single Precision	Double Precision
β		2	2
t		24	53
rnd		5	5
	‡	2	2
ngrd		0	0
machep		-23	-52
negep		-24	-53
iexp		8	11
minexp		-126	-1022
maxexp		128	1024
eps		0.1192093e-06	0.2220446049250d-15
epsneg		0.5960464e-07	0.1110223024625d-15
xmin		0.1175494e-37	0.2225073858507d-307
xmax		0.3402823e+39	0.1797693134862d+309

IEEE-style systems frequently do all floating-point computations in extended precision, retaining results in extended registers whenever possible. Optimizing compilers on such systems exploit this behavior, and MACHAR malfunctions unless compiler optimization is suppressed. Results in this report were all obtained with the -N flag to kill compiler optimization, although preliminary runs with the default level of optimization were equally successful with Weitek arithmetic (which lacks extended precision). Regardless of the optimization level and floating-point chips used, runs with Sequent Fortran V3.2 were uniformly unsuccessful, terminating with arithmetic exceptions or becoming lost in infinite loops. We have not attempted to circumvent the problems nor to tabulate results for that compiler.

Table 1 lists single-precision and double-precision parameters determined by MACHAR. Here and in subsequent tables, the primary entries are for tests with the Intel chip. Where results differ, entries for the Weitek chip are marked with a double dagger (‡).

Definitions of the tabulated parameters are as follows:

1) β, the radix for the representation scheme;
2) t, the number of base-β digits in the floating-point significand;
3) *rnd*, a parameter indicating the method of rounding in addition and the type of underflow (full or partial):
   a value of 0 indicates truncation with full underflow;
   a value of 1 indicates some non-IEEE form of rounding with full underflow;
   a value of 2 indicates IEEE-style of rounding with full underflow;
   a value of 3 indicates truncation with partial underflow;
   a value of 4 indicates some non-IEEE form of rounding with partial underflow; and
   a value of 5 indicates IEEE-style rounding with partial underflow;
4) *ngrd*, 0 for $rnd \neq 0$; otherwise, the number of base-β guard digits used in multiplication;
5) *machep*, the exponent for the smallest power of β (but bounded below by $t-3$) whose sum with 1.0 is greater than 1.0;
6) *negep*, the exponent for the smallest power of β (but bounded below by $t-3$) whose difference with 1.0 is less than 1.0;
7) *iexp*, the number of bits dedicated to the representation of the exponent (including bias or sign) of a floating-point number;
8) *minexp*, the smallest permissible exponent;
9) *maxexp*, the largest permissible exponent;
10) *eps*, on a binary machine, the floating-point number β^{machep};
11) *epsneg*, on a binary machine, the floating-point number β^{negep};
12) *xmin*, the floating-point number β^{minexp}; and
13) *xmax*, an approximation of the floating-point number β^{maxexp}.

Because MACHAR is intended to be used by other programs, it must avoid exceptions that will terminate execution. Thus, it is severely limited in what it can attempt to determine about an arithmetic system. PARANOIA (see Table 2), a second and more probing program for examining computer arithmetic, does not have that handicap. It is a self-contained program that periodically marks its progress by writing recovery information to file. Thus, if execution is terminated for any of a number of anticipated reasons, the program can be restarted with the expectation that saved data will permit it to properly report the reason for its termination and to resume execution beyond the troublesome point. In this way, with possible restarts from time to time, the program is able to run tests on arithmetic characteristics that are not possible with MACHAR.

PARANOIA was originally written in BASIC by W. Kahan at the University of California, Berkeley, and then translated to Fortran by T. Quarles and G. Taylor. It was made available to the general public by R. Karpinski at the University of California, San Francisco [Karpinski 1985]. The particular version used here was further refined at AT&T Bell Laboratories and transmitted privately by David Gay.

The PARANOIA results reported in Table 2 are self-explanatory (except that ULP refers to Units in the Last Place of the significand). The final evaluation issued by PARANOIA was that Intel arithmetic was "excellent," and that Weitek arithmetic was "satisfactory though flawed." We disagree with both of these assessments, primarily because of the evaluation of 0/0 as 1.0 without an error indication, and will notify Karpinski of PARANOIA's failure in this case.

Table 2. Results from PARANOIA for i80387

‡ denotes results for Weitek 1167

Test		Single-Precision Result	Double-Precision Result
Integer Arithmetic		Okay	Okay
β		2	2
epsneg		5.96046448E-08	1.11022302E-16
t		24	53
Extra-Precise Subexpressions		Yes (29 extra bits)	No
	‡	No	No
Subtraction Normalized		Yes	Yes
Guard Digits in $\times$, $\div$, $-$		Yes	Yes
Rounding in $+/-$, $\times$, $\div$		Yes	Yes
Sticky Bit		Yes	Yes
Multiplication Commutative		Yes	Yes
$\sqrt{(i \times i)} = i$		Yes	Yes
Sqrt Monotone		Yes	Yes
Sqrt Correctly Rounded or Chopped		Correctly Rounded	Correctly Rounded
	‡	Neither	Neither
Error Bounds for Sqrt		-0.5 and +0.5 ULP	-0.5 and 0.5 ULP
	‡	-0.6 and +1.0 ULP	-0.6 and 0.5 ULP

We note with surprise that extra-precise subexpressions are generated in the single-precision case for Intel arithmetic although we asked the compiler to kill optimization, and further that these extra-precise single-precision subexpressions are accurate only to double precision and not to extended precision. Flaws detected for Weitek arithmetic include the lack of graceful underflow and the lack of correct rounding in the *sqrt* function despite the machine's using IEEE round-to-nearest-even arithmetic. More interesting is the *overflow exception* triggered by the computation of $\beta^{-2 \times minexp}$. This is serious because

Table 2. Results from PARANOIA for i80387 (Continued)

‡ denotes results for Weitek 1167

Test		Single-Precision Result	Double-Precision Result
z^i for Small Positive i		Okay	Okay
minpos		1.40129846E-45	4.94065564E-324
	‡	1.1754944E-38	2.22507386E-308
$(minpos + minpos)/minpos$		2.0	2.0
$1.375 \times minpos - minpos$	‡	0.0 w/o Underflow Signal	0.0 w/o Underflow Signal
Gradual Underflow		Yes	Yes
	‡	No	No
xmin (underflow threshold)		1.17549435E-38	2.22507386E-308
$\beta^{-2 \times minexp}$		Okay	Okay
	‡	Okay	Overflow Exception
$x^{((x+1)/(x-1))}$ vs $exp(2), x \rightarrow 1$		Okay	Okay
z^q for Nearly Extremal Values		Okay	Okay
Overflow		Exception	Exception
xmax (Overflow Threshold)		3.40282347E+38	1.79769313E+308
$z = xmax \times 1, xmax / 1$		Okay	Okay
1/0		Zero Divide Exception	Zero Divide Exception
0/0		1.0 w/o Exception	1.0 w/o Exception

the exception is misleading and halts execution, whereas the expected underflow would simply return a zero result and continue.

Overall, other than as noted here, the arithmetic seems good. Both floating-point environments provide many of the features of the IEEE floating-point standard, but the Weitek environment lacks graceful underflow and proper rounding in the square root. Other requirements for conformance to the standard, such as support for infinities and NaNs and control of rounding mode, have not been checked. Often these features are not supported by the compiler even if they are available on the hardware.

On the other hand, a defect surfaced during these tests that hampered us throughout the testing process. Formatted output returns a string of asterisks for any double-precision quantity with a three-digit exponent. This is a major flaw in the Fortran environment that renders it useless in many situations. We urge that it be fixed immediately.

3. ELEFUNT/INTFUNT Results

ELEFUNT is the suite of transportable Fortran test programs from the *Software Manual for the Elementary Functions* by Cody and Waite [1980], and INTFUNT is a companion suite of test programs extending the ELEFUNT concepts to tests of intrinsic functions. Each of the test programs exercises one or more of the elementary or intrinsic functions to estimate accuracy, check simple mathematical properties, and assess the response to improper or unusual arguments. The requirement that the test programs be portable has limited the approach in accuracy checking to determining how well the function program tested satisfies certain well-behaved identities.

The INTFUNT tests interpret results without reporting specific statistics. Table 3 summarizes single- and double-precision results for INTFUNT tests of AINT, ANINT, INT, and MOD. The Intel and Weitek test results were identical most of the time, the only disagreements between the two being in the results for ANINT. ANINT in the Weitek environment, for example, rounded $2^{24}-1$ to 2^{24}. We speculate that the algorithm used does a floating-point add of 1/2 to the argument and then invokes AINT. If this is the case, then $(2^{24}-1) + 1/2$ returns a result halfway between two floating-point numbers, and the IEEE round-to-even rule causes the result to be rounded up to 2^{24}, which is incorrect. Our hypothesis has been tested with additional arguments without contradicting the hypothesis. The analogous algorithm appears to have been used for the default double-precision version of ANINT also. These routines should be corrected.

In contrast with the INTFUNT tests, ELEFUNT tests report statistics without interpretation. A typical accuracy test from the ELEFUNT suite evaluates an identity by using 2000 random arguments uniformly distributed across an interval, and reports the number of times the identity was exactly satisfied, the number of times it was not satisfied on the high side and on the low side, the maximum relative error (MRE) encountered, and the root-mean-square (RMS) relative error. To normalize results, the MRE and RMS errors are reported as an estimated number of erroneous trailing base-β digits in the significand using the equations

$$MRE = t + ln(\max|E_i|) / \ln(\beta),$$

and

$$RMS = \max\,[\,0.0, t + ln(\sum_i E_i^2 / N) / (2\ln(\beta))],$$

where E_i is the error for the i-th argument. Note that the computation of RMS has been adjusted so it never reports a negative loss of significant bits. In general, MRE values between 1.0 and 2.0 are common with ELEFUNT on binary machines; values over 2.5 are rare and often indicate trouble.

The evaluation of the identities used in the accuracy tests inevitably introduces some error. This error is estimated by *calibrating* the test program, i.e., by running the test program in single-precision arithmetic with a function program that accepts single-precision arguments, does all computations in double precision, and returns single-precision results. Such a function returns the best possible single-precision values on a particular machine/compiler combination. Calibration is thus a practical determination of the reliability of our test procedure, a calibration of the testing tool to determine the background noise level on a particular system.

Table 3. INTFUNT Test Results for i80387

‡ denotes results for Weitek 1167

Test		Single-Precision Result	Double-Precision Result
AINT			
$aint(x)$ *vs* $0,\ x = 2^i, i = -1, minexp$		Okay	Okay
$aint(1+x)$ *vs* $1,\ x = 2^i, i = -1, minexp$		Okay	Okay
$aint(x+1/2)$ *vs* $x,\ x = 2^i, i = 1, max(35, t+3)$		Okay	Okay
Parity check		Okay	Okay
$2^t - 1.0$		Okay	Okay
$aint(\pm xmax)$		Okay	Okay
ANINT			
$anint(x),\ x = 2^i, i = -1, minexp$		Okay	Okay
$anint(x+1/2)$ *vs* $1,\ x = 2^i, i = -1, minexp$		Okay	Okay
$anint(x+1/2)$ *vs* $x+1,\ x = 2^i, i = 1, max(t+3, 35)$		Okay	Okay
Parity check		Okay	Okay
$anint(2^t - 1.0)$		Okay	Okay
	‡	2^{24}	2^{53}
$anint(xmax)$		Okay	Okay
$anint(-xmax)$		Okay	Okay
INT			
$int(x)$ *vs* $0,\ x = 2^{-i}, i = 1, 126$		Okay	Okay
$int(1+x)$ *vs* $1,\ x = 2^{-i}, i = 1, 126$		Okay	Okay
$int(x+1/2)$ *vs* $x,\ x = 2^i, i = 1, 30$		Okay	Okay
Parity check		Okay	Okay
$2^n - 1.0, n = min(t, 31)$		Okay	Okay
$int(xmax)$		Invalid Operation Exception	--
	‡	Overflow Exception	--
MOD			
$mod(n \times x + half, x)$, x and n random in (0,1000)		All Bits Correct	All Bits Correct
$mod(x+½, 1.0),\ x = 2^i, i = 1, max(t+3, 35)$		Okay	Okay
Parity check		Okay	Okay
$mod(1.0, 0.0)$		Zero Divide Exception	Zero Divide Exception

We would not expect the MRE (measured in our way) for a single-precision program to exceed that for the corresponding double-precision program, and it rarely does in tests on the Sequent Symmetry. The Intel 80387 provides many of the core computations for the elementary functions, so we would further expect the Intel library to be superb. We cannot say whether the Weitek chip provides similar capabilities. Even if it does not, programs for the elementary functions are usually written in C on UNIX systems, and C normally does all floating-point computation in at least double precision. We would therefore expect the single-precision functions, even those in the Weitek environment, to be accurate to within rounding error.

Table 4 summarizes results for the ELEFUNT accuracy tests. Results labeled single-precision and double-precision are for Intel; those preceded with a ‡ are for Weitek. Where separate Weitek results are not given, they coincide with the Intel results. Intel calibration results are included in the table. Those for the Weitek case, again marked with ‡, are included *only if they are significantly different from the corresponding Intel results.*

Following are detailed discussions of test results for each function. In addition to checking accuracy, most of the test programs also check for preservation of parity, for small argument approximations, for behavior near the boundaries of the function domain, and for response to illegal or ill-advised arguments. Unless otherwise noted, the results for these ancillary checks were satisfactory.

The tabulated results for the single-precision ASIN/ACOS functions duplicate the calibration results, so the functions are doing the best they can. Although the Weitek tests sometimes report larger errors than the corresponding Intel tests, the difference is consistent with Weitek's lack of extended precision. Overall, the ASIN/ACOS accuracy test results are satisfactory for both systems and both precisions.

Arguments greater than 1.0 in magnitude cause ASIN or ACOS to terminate execution with an appropriate error message, contrary to the on-line documentation which claims that execution continues with a zero result. In the Intel case, a more reasonable response would be continued execution with a NaN result; apparently the system does not fully exploit the capabilities of the Intel chip.

The accuracy results for the Intel ATAN appear normal; single-precision results either agree with the calibration results or are slightly better (the second test), and the double-precision results are almost identical. However, the Weitek single-precision results are uniformly, and sometimes significantly, worse than the Weitek calibration results. In the last two tests, for example, the reported MRE and RMS values are even worse than the corresponding double-precision ones. While the routine has acceptable accuracy, it is possible to do better. Results for the double-precision Weitek routine compare favorably with those for other machines lacking extended precision.

Each of the ATAN programs handles small and extreme arguments correctly. The ATAN2 programs are correct in their handling of troublesome arguments, including cases where one of the arguments is zero. Further, they are consistent in returning zero for ATAN2(0.0,0.0). While we personally prefer a NaN or an error return for this case, persuasive arguments can be made for a zero result. We find the overall performance of these routines to be satisfactory.

Test results for the EXP programs are generally disappointing. Only the Intel single-precision EXP matches expectations with error statistics duplicating those for the calibration runs. The MRE reported for the Weitek single-precision routine, while acceptable, is about twice as large as expected based on the calibration runs. Considering the lack of extended precision, the double-precision Weitek results are also acceptable. However, the estimated loss of more than 9 bits (i.e., more than 2^8 ULPs, and almost 3 significant decimal digits) in the Intel DEXP for large arguments suggests a major problem in argument reduction. This program is unsatisfactory by today's standards.

Each routine does well on tests with special identities, the Intel routines returning exactly 1.0 for EXP(X)*EXP(-X) most of the time. Auxiliary tests with extreme arguments are satisfactory, although the Intel and Weitek double-precision results disagree after 14 significant decimal figures (an error consistent with the reported MRE for the Intel routine). All routines return zero with quiet underflow for large negative arguments, and all properly terminate execution with an error message for large positive arguments.

Table 4. ELEFUNT Test Results for i80387

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
ASIN						
asin(x) vs Taylor Series	(-1/8, 1/8)		Calibration.	1999	0.10	0.00
			Single Prec.	1999	0.10	0.00
		‡	Single Prec.	1998	0.10	0.00
			Double Prec.	1581	0.99	0.00
		‡	Double Prec.	1068	1.22	0.00
	(3/4, 1)		Calibration	1671	1.00	0.00
		‡	Calibration	1510	1.22	0.00
			Single Prec.	1671	1.00	0.00
		‡	Single Prec.	1510	1.22	0.00
			Double Prec.	1590	1.00	0.00
		‡	Double Prec.	754	2.24	0.72
ACOS						
acos(x) vs Taylor Series	(-1/8, 1/8)		Calibration	1963	0.46	0.00
			Single Prec.	1963	0.46	0.00
			Double Prec.	1576	0.47	0.00
		‡	Double Prec.	1467	0.47	0.00
	(3/4, 1)		Calibration	1534	0.99	0.00
		‡	Calibration	1193	1.99	0.00
	(3/4, 1)		Single Prec.	1534	0.99	0.00
		‡	Single Prec.	1193	1.99	0.00
			Double Prec.	1280	1.55	0.00
		‡	Double Prec.	591	2.49	0.88
	(-1, -3/4)		Calibration	1819	0.73	0.00
	(-1, -3/4)		Single Prec.	1819	0.73	0.00
	(-1, -3/4)	‡	Single Prec.	1777	0.73	0.00
			Double Prec.	1778	0.72	0.00
		‡	Double Prec.	1104	1.72	0.06

The Weitek error messages identify the problem as an argument that exceeds the maximum permissible for the EXP function, but the Intel error messages are for an overflow with no indication that the problem lies in the EXP function. This oversight should be corrected. Indeed, if the compiler and library fully supported the Intel chip, EXP would return an infinity, signal overflow, and continue execution.

The results of accuracy tests for the LOG functions are all satisfactory, although the reported MRE for the Weitek single-precision routine is a tad large. Ancillary tests of special identities and with special

Table 4. ELEFUNT Test Results for i80387 (Continued)

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
ATAN						
atan(x) *vs Taylor Series*	(-1/16, 1/16)		Calibration	2000	0.00	0.00
			Single Prec.	2000	0.00	0.00
		‡	Single Prec.	965	1.32	0.02
			Double Prec.	1999	0.67	0.00
		‡	Double Prec.	939	1.78	0.17
atan(x) *vs* *atan*$(1/16)+$*atan*$[\frac{(x-1/16)}{(1+x/16)}]$	$(1/16, 2-\sqrt{3})$		Calibration	1436	1.00	0.00
		‡	Calibration	1343	1.00	0.00
			Single Prec.	1556	1.00	0.00
		‡	Single Prec.	1041	1.71	0.02
			Double Prec.	1513	1.00	0.00
		‡	Double Prec.	825	1.95	0.41
2 *atan*(x) *vs* *atan*$[\frac{2x}{(1-x^2)}]$	$(2-\sqrt{3}, \sqrt{2}-1)$		Calibration	1695	0.93	0.00
		‡	Calibration	1495	0.93	0.00
			Single Prec.	1695	0.93	0.00
		‡	Single Prec.	679	2.38	0.73
			Double Prec.	1477	0.93	0.00
		‡	Double Prec.	1357	1.91	0.00
	$(\sqrt{2}-1, 1)$		Calibration	1839	1.00	0.00
	$(\sqrt{2}-1, 1)$	‡	Calibration	1755	1.00	0.00
			Single Prec.	1839	1.00	0.00
		‡	Single Prec.	580	2.54	0.93
			Double Prec.	1746	1.00	0.00
		‡	Double Prec.	1303	1.35	0.00

arguments are also satisfactory. Indeed, the Weitek routines return zero for LOG(X) - LOG(1.0/X) most of the time. Each routine properly terminates execution when given a negative argument, but the Intel error messages are again misleading, reporting a floating invalid operation with no reference to LOG. The Intel error message changes to floating divide by zero for a zero argument, still without reference to LOG. The Weitek error messages are correct and informative.

The POWER/DPOWER pair (aliases for exponentiation) are a mixed bag. Accuracy results for the single-precision routines duplicate those of the corresponding calibration runs, showing that the compute in double-precision strategy was used. However, the double-precision routines return large errors for large arguments. The probable explanation is simple. When higher-precision arithmetic is not available, a good program for the power function is perhaps the most difficult elementary function program to write.

Table 4. ELEFUNT Test Results for i80387 (Continued)

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
EXP						
$exp(x-1/16)$ *vs* $\frac{exp(x)}{exp(1/16)}$	$(1/16-ln(2)/2, ln(2)/2)$		Calibration	1475	1.00	0.00
			Single Prec.	1475	1.00	0.00
		‡	Single Prec.	977	1.99	0.23
			Double Prec.	1469	1.00	0.00
		‡	Double Prec.	986	1.99	0.21
$exp(x-45/16)$ *vs* $\frac{exp(x)}{exp(45/16)}$	$(-5\,ln(2), ln[2^{58} xmin])$		Calibration	1498	1.00	0.00
		‡	Calibration	1514	1.00	0.00
			Single Prec.	1498	1.00	0.00
		‡	Single Prec.	989	1.98	0.19
			Double Prec.	1	9.14	7.80
		‡	Double Prec.	930	1.96	0.25
	$(10\,ln(2), ln[.9\,xmax])$		Calibration	1513	1.00	0.00
		‡	Single Prec.	1509	1.00	0.00
			Single Prec.	1513	1.00	0.00
		‡	Single Prec.	1013	1.93	0.15
			Double Prec.	0	9.14	7.80
		‡	Double Prec.	868	1.96	0.32

Mathematically $x^y = exp[y \times ln(x)]$. A little error analysis shows that the relative error in the function value is roughly the absolute error in the argument to the exponential function. This means that exponentiation using the obvious composition of functions loses as many significant digits as there are digits before the decimal point in the value of $y \times ln(x)$. The reported MRE values in the third Intel test and the third and fourth Weitek tests are consistent with this error analysis, indicating that the composition-of-functions approach was probably used. On the other hand, the last Intel test shows good accuracy for large arguments, which is inconsistent with our guess at the algorithm. To explore this inconsistency, we compared *X**Y* against a result computed from repeated multiplications of *X* with itself, and found good agreement. Based on these results, we believe that DPOWER uses a similar algorithm when the exponent is an integer, and that it uses a more complicated but flawed algorithm otherwise.

We ran further tests to explore this inconsistency in the Intel results. Note that the third test compares output from DPOWER against independently generated values, and that the last test compares one output from DPOWER against another. We ran an additional test to determine whether DPOWER contained an internal error that would not be revealed by our last test. In particular, we selected X = 9.625

Table 4. ELEFUNT Test Results for i80387 (Continued)

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
LOG						
ln(x) *vs Taylor Series*	$(1-\varepsilon,1+\varepsilon)$		Calibration	2000	0.00	0.00
			Single Prec.	2000	0.00	0.00
		‡	Single Prec.	1217	1.87	0.00
			Double Prec.	1999	0.38	0.00
		‡	Double Prec.	1183	1.50	0.00
ln(x) *vs ln*$(17x/16)$–*ln*$(17/16)$	$(1/\sqrt{2}, 15/16)$		Calibration	1434	1.00	0.00
			Single Prec.	1719	1.00	0.00
		‡	Single Prec.	1401	1.30	0.00
			Double Prec.	1695	1.00	0.00
		‡	Double Prec.	935	1.41	0.07
ln$(x \times x)$ *vs* 2 *ln*(x)	$(16, 240)$		Calibration	1959	0.96	0.00
			Single Prec.	2000	0.00	0.00
		‡	Single Prec.	1535	1.00	0.00
			Double Prec.	1951	0.99	0.00
		‡	Double Prec.	1820	1.00	0.00
LOG10						
log(x) *vs log*$(11x/10)$–*log*$(11/10)$	$(1/\sqrt{10}, .9)$		Calibration	922	2.13	0.38
			Single Prec.	941	2.03	0.37
		‡	Single Prec.	770	2.42	0.57
			Double Prec.	962	2.05	0.37
		‡	Double Prec.	792	2.41	0.58

and Y = 80, both of which are exactly representable in the floating-point system. Then we computed X**Y, (X*X)**(Y / 2), and (X*X*X*X)**(Y / 4) in a loop in which Y was increased by 1.0 for each loop iteration. Each argument was exact, hence contributed no error to the final result. Strangely, the computed values agreed whenever the corresponding exponents were both integers, or both non-integers, but disagreed when one exponent was an integer and the other was not. This strongly suggests that DPOWER uses different algorithms for integer and non-integer exponents, and that one of the internal algorithms has a consistent error in it.

If our guesses are correct, the algorithms in both the Intel and Weitek DPOWER functions must be replaced by ones in which the intermediate value of $y \times ln(x)$ is computed to more than working precision.

Table 4. ELEFUNT Test Results for i80387 (Continued)

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
POWER						
x *vs* x^1	(1/2, 1)		Calibration	2000	0.00	0.00
			Single Prec.	2000	0.00	0.00
			Double Prec.	2000	0.00	0.00
$(x \times x)^{1.5}$ *vs* $(x \times x) \times x$	(1/2, 1)		Calibration	1986	0.95	0.00
		‡	Calibration	1986	0.98	0.00
			Single Prec.	1986	0.95	0.00
		‡	Single Prec.	1986	0.98	0.00
			Double Prec.	1188	1.97	0.01
		‡	Double Prec.	709	2.22	0.56
	$(1, xmax^{1/3})$		Calibration	1986	0.98	0.00
			Single Prec.	1986	0.98	0.00
		‡	Single Prec.	1978	0.98	0.00
			Double Prec.	1	9.99	8.63
		‡	Double Prec.	3	10.45	9.25
x^y *vs* $(x \times x)^{y/2}$	X: (1/10, 10), Y: $\left(\frac{ln[xmin]}{ln[100]}, \frac{-ln[xmin]}{ln[100]}\right)$		Calibration	2000	0.00	0.00
			Single Prec.	2000	0.00	0.00
			Double Prec.	2000	0.00	0.00
		‡	Double Prec.	1698	9.00	6.21

(It is possible that the Intel algorithm for non-integer exponents is intended to be of this type.) Such algorithms have been known and used for more than 20 years [Cody and Waite 1980] and are now an industry standard. The disturbing factor here is that a similar problem was pointed out to Sequent almost four years ago in our test report on the Sequent Balance [Cody 1986d]. We are disappointed that the advice there has either not been applied here or has been incorrectly applied.

There are again some flaws in the handling of Intel error conditions. Each program correctly aborts execution with a clear error message identifying the problem when asked to raise 0.0 to the 0.0 power. (We prefer this to the approach espoused by some that 0.0**0.0 be 1.0.) The Fortran Standard [ANSI 1978] says nothing about what should happen for X**Y when X is negative. At one time this was considered to be an error, but the accepted practice now is to return a proper mathematical result when Y is a floating-point integer and to return an error otherwise. Each exponentiation program on the Symmetry correctly computes the result for negative X and integer Y. The Weitek library also terminates execution

Table 4. ELEFUNT Test Results for i80387 (Continued)

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
SIN						
$sin(x)$ vs $3sin(x/3)-4sin(x/3)^3$	$(0, \pi/2)$		Calibration	1634	1.00	0.00
		‡	Calibration	1250	1.32	0.00
			Single Prec.	1634	1.00	0.00
		‡	Single Prec.	1072	2.04	0.00
			Double Prec.	1263	1.27	0.00
		‡	Double Prec.	1109	1.53	0.00
	$(6\pi, 6.5\pi)$		Calibration	1630	1.00	0.00
		‡	Calibration	1215	1.25	0.00
			Single Prec.	1630	1.00	0.00
		‡	Single Prec.	954	1.84	0.07
			Double Prec.	1242	1.37	0.00
		‡	Double Prec.	982	1.86	0.01
COS						
$cos(x)$ vs $4cos(x/3)^3-3cos(x/3)$	$(7\pi, 7.5\pi)$		Calibration	1600	0.99	0.00
		‡	Calibration	1243	1.15	0.00
			Single Prec.	1600	0.99	0.00
		‡	Single Prec.	1053	1.53	0.00
			Double Prec.	1230	1.35	0.00
		‡	Double Prec.	1082	1.58	0.00

with a clear error message when both arguments are negative, but the Intel library terminates with a "Segmentation violation" error message that confuses at best.

Error statistics for the SIN/COS functions all appear normal. The Weitek single-precision results are slightly worse than the Calibration results, especially in the first test. The differences are not enough to cause concern, but the routine could be "tuned" a little. The programs also pass all tests for parity and small-argument approximations. Small-argument tests for the Intel single-precision SIN reveal that a double-precision function value is returned and used in intermediate results. This is contrary to expectations and should be corrected, in our opinion. The Intel routines continue normal execution when given arguments large enough that perhaps half of the significance should be lost during argument reduction. The single-precision routines can do this safely because internal computations are done in higher precision. That is not the case for the double-precision programs, and we believe that a warning should be given for arguments greater than, say, 10^9 in magnitude. The Weitek routines take a different, but flawed approach for large arguments. SIN returns a value correct to only about two decimal digits for an argument of 10^8. Both SIN and DSIN terminate execution for arguments of 10^{10}. The error message for DSIN correctly identifies the problem as too large an argument, but that for SIN is an overflow message.

Table 4. ELEFUNT Test Results for i80387 (Continued)

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
SINH						
sinh(x) *vs Taylor Series*	(0, 1/2)		Calibration	1982	0.98	0.00
		‡	Calibration	1979	0.98	0.00
			Single Prec.	1982	0.98	0.00
		‡	Single Prec.	1979	0.98	0.00
			Double Prec.	1969	0.99	0.00
sinh(x) *vs* $\frac{[sinh(x+1)+sinh(x-1)]}{2cosh(1)}$	(3, $ln(xmax)-1/2$)		Calibration	999	1.43	0.04
		‡	Calibration	956	1.57	0.15
			Single Prec.	1014	1.39	0.04
		‡	Single Prec.	956	1.57	0.15
			Double Prec.	934	1.67	0.16
		‡	Double Prec.	126	12.94	11.05
COSH						
cosh(x) *vs Taylor Series*	(0, 1/2)		Calibration	1967	0.96	0.00
			Single Prec.	1967	0.96	0.00
			Double Prec.	1952	0.99	0.00
		‡	Double Prec.	1765	1.00	0.00
cosh(x) *vs* $\frac{[cosh(x+1)+cosh(x-1)]}{2cosh(1)}$	(3, $ln(xmax)-1/2$)		Calibration	999	1.40	0.04
			Calibration	960	1.61	0.15
			Single Prec.	991	1.40	0.04
		‡	Single Prec.	960	1.61	0.15
			Double Prec.	983	1.62	0.16
		‡	Double Prec.	122	12.96	11.07

Unfortunately, the overflow message is also returned for much larger arguments in DSIN. We find much to complain about here. First, a message is warranted when accuracy degenerates as it does in SIN, say for arguments greater than about 10^4 in magnitude. Second, if a message is to be given, it should be the correct message and it should identify the offending program.

Error statistics for SINH/COSH are nominal except those for the double-precision Weitek programs. There are no obvious explanations for the inaccuracies for large arguments, because the Weitek DEXP program appears to be good. The accuracy of the Weitek DSINH/DCOSH is clearly unsatisfactory.

All programs passed parity tests and tests with small arguments. The Intel programs and the Weitek COSH/DCOSH programs also correctly terminated execution with a meaningful error message when supplied large arguments. The Weitek SINH/DSINH programs failed this test, however, continuing

Table 4. ELEFUNT Test Results for i80387 (Continued)

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
SQRT						
x *vs* $\sqrt{x \times x}$	$(1/\sqrt{2}, 1)$		Calibration	2000	0.00	0.00
			Single Prec.	2000	0.00	0.00
		‡	Single Prec.	1217	1.44	0.00
			Double Prec.	2000	0.00	0.00
		‡	Double Prec.	1841	0.50	0.00
	$(1, \sqrt{2})$		Calibration	2000	0.00	0.00
			Single Prec.	2000	0.00	0.00
		‡	Single Prec.	1543	1.00	0.00
			Double Prec.	2000	0.00	0.00
		‡	Double Prec.	1998	1.00	0.00
TAN						
$tan(x)$ *vs* $\frac{2tan(x/2)}{[1-tan(x/2)^2]}$	$(0, \pi/4)$		Single Prec.	1	1.01	0.00
		‡	Single Prec.	1078	1.74	0.01
			Double Prec.	1113	1.92	0.03
		‡	Double Prec.	1093	1.92	0.03
	$(7\pi/8, 9\pi/8)$		Single Prec.	0	1.27	0.00
		‡	Single Prec.	1299	1.50	0.00
			Double Prec.	1266	1.24	0.00
		‡	Double Prec.	1122	1.88	0.00
	$(6\pi, 6.25\pi)$		Single Prec.	1	1.18	0.00
		‡	Single Prec.	1075	1.77	0.02
			Double Prec.	1085	1.99	0.04
		‡	Double Prec.	1067	1.83	0.05

execution after delivering the result 1.0. Such performance is unbelievable.

Test results for the Intel SQRT/DSQRT are all nominal, but the reported MRE for the Weitek SQRT is too large in the first test. On the other hand, the Weitek routines correctly respond to negative arguments with a meaningful error message and program termination, while the Intel routines abort execution with a "Floating invalid operation" message.

Error statistics for the TAN/DTAN routines are reasonable. The low frequency of exact results for the Intel Tan tests is caused by the way the compiler handles some intermediate results at a crucial point. This is a weakness in the test program itself. Slightly modifying the test source results in a count of exact results consistent with the other TAN tests. The MRE values also degenerated slightly, but not

Table 4. ELEFUNT Test Results for i80387 (Continued)

‡ denotes results for Weitek 1167

Test	Interval		Precision	Exact	MRE	RMS
TANH						
$tanh(x)$ vs $\frac{[tanh(x-1/8)+tanh(1/8)]}{[1+tanh(x+1/8)tanh(1/8)]}$	(1/8, *ln* [3]/2)		Single Prec.	1615	0.99	0.00
		‡	Single Prec.	1077	1.50	0.00
			Double Prec.	426	3.97	1.57
		‡	Double Prec.	1086	1.65	0.00
	(1/8+*ln* [3]/2, 59 *ln* [2]/2)		Single Prec.	1588	0.76	0.00
		‡	Single Prec.	924	1.13	0.00
			Double Prec.	663	2.08	0.39
		‡	Double Prec.	834	10.58	6.16

significantly. We report the results for the unmodified tests to be consistent with previous reports.

Tests of parity and tests with special arguments uncovered no problems. As with the SIN/COS programs, the Intel TAN/DTAN programs accepted large arguments without complaint, whereas the Weitek programs terminated execution with an appropriate error message for arguments greater than about 10^{10} in magnitude. The online documentation for TAN claims that results are garbage for arguments greater than about 2^{31}. This is probably the threshold built into the Weitek programs, but we found the Intel DTAN to be accurate well beyond that point. Nevertheless, we cannot believe that the Intel results for arguments like 10^{30}, let alone 10^{200}, have any significance at all, and strongly recommend some sort of error return for arguments beyond some sensible threshold. The documentation also claims that the TAN routines return large values at their "singular points." Of course, there are no "singular points" in the computer, so this statement is nonsense and should be removed from the documentation.

There are no COT routines, but the Fortran standard does not require any.

The MRE values for the Intel DTANH in the first test and the Weitek DTANH in the second test are the largest we can recall for these functions in our series of tests on various machines. Because they occur for moderate arguments, we suspect blunders in the programs. All of the TANH family of programs performed satisfactorily with extreme arguments.

4. Summary

The libraries tested on the Sequent Symmetry offer extreme contrasts in quality. Some routines are excellent, some are terrible. We uncovered a number of major problems in this exercise:

(1) formatted print statements fail whenever the exponent is greater than 99 in magnitude;

(2) 0.0/0.0 evaluates to 1.0 in both the Intel and Weitek environments;

(3) $\beta^{-2\times minexp}$ generates overflow in the Weitek environment;

(4) the Weitek ANINT routines fail for arguments like $\beta^{24}-1$;

(5) the Intel DEXP routine is unacceptably inaccurate for large arguments;

(6) double-precision exponentiation is unacceptably inaccurate in both environments, and the Intel program seems to have an internal inconsistency;

(7) the Intel single-precision SIN returns and uses double-precision values (is this the only culprit in the Intel library?);

(8) the Intel SIN/DSIN and TAN/DTAN programs need error returns for large arguments;

(9) the error returns in the Weitek SIN/DSIN programs are wrong;

(10) the Weitek DSINH/DCOSH programs are unacceptably inaccurate for large arguments;

(11) the Weitek DSINH program returns 1.0 and continues execution for out-of-range arguments;

(12) both the Intel and the Weitek DTANH programs are unacceptably inaccurate for large arguments; and

(13) many of the error messages for faulty arguments to Intel routines (EXP/DEXP, LOG/DLOG, exponentiation, SQRT/DSQRT) are totally misleading (overflow, invalid operation, etc.) and do not identify the offending routine.

Some of these errors are so blatant that we are surprised they were not discovered during product testing. Many are significant enough that they may undermine confidence in scientific computation on and military use of the Sequent Symmetry. We urge that they be remedied quickly.

The remaining problems, accuracy of the Weitek ATAN and EXP routines and incorrect documentation for ASIN/ACOS and TAN, are minor. We suggest that they be corrected, but not at the expense of correcting the major problems.

5. Epilogue

Systems people at Sequent who reviewed a draft of this report have stated privately that the deficiencies uncovered by our tests will be corrected in a forthcoming software release. Based on our past experiences with Sequent, we confidently expect improved software soon.

References

ANSI [1978]. *American National Standard Programming Language FORTRAN.* ANSI X3.9-1978. New York: American National Standards Institute, Inc.

W. J. Cody [1986a]. *An Alternative Library under 4.2 BSD UNIX on a VAX 11/780.* Argonne National Laboratory Report ANL-86-10.

W. J. Cody [1986b]. *ELEFUNT Test Results under X1.4 on the Encore Multimax.* Technical Memorandum ANL/MCS-TM-68, Argonne National Laboratory.

W. J. Cody [1986c]. *ELEFUNT Test Results under FX/FORTRAN Version 1.0 on the Alliant FX/8.* Technical Memorandum ANL/MCS-TM-78, Argonne National Laboratory.

W. J. Cody [1986d]. *ELEFUNT Test Results under NS32000 Fortran V2.5.3 on the Sequent* Balance. Technical Memorandum ANL/MCS-TM-80, Argonne National Laboratory.

W. J. Cody [1988a]. "Algorithm 665. MACHAR: A subroutine to dynamically determine machine parameters." *ACM Trans. on Math. Soft. 14*, pp. 303-311.

W. J. Cody [1988b]. *ELEFUNT Test Results under FORTRAN-PLUS on the Active Memory Technology DAP 510-8.* Technical Memorandum ANL/MCS-TM-125, Argonne National Laboratory.

W. J. Cody [1989]. *ELEFUNT Test Results Using Titan Fortran under Ardent UNIX 2.0 on the* Titan. Technical Memorandum ANL/MCS-TM-129, Argonne National Laboratory.

W. J. Cody and W. Waite [1980]. *Software Manual for the Elementary Functions.* Englewood Cliffs, N.J.: Prentice-Hall.

IEEE [1985]. *IEEE Standard for Binary Floating-Point Numbers.* ANSI/IEEE Std 754-1985. New York, IEEE.

R. Karpinski [1985]. "PARANOIA: A floating-point benchmark." *BYTE* 10, no. 2.