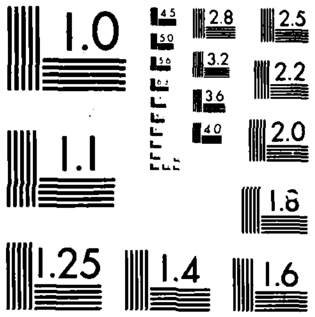
AD-A164 050 DEVELOPMENT AND EVALUATION OF MATH LIBRARY ROUTINES FOR A 1750A AIRBORNE MICROCOMPUTER(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 1/2

UNCLASSIFIED J J FRIED 04 DEC 85 AFIT/GCS/MA/85D-3 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

1

DEVELOPMENT AND EVALUATION OF MATH LIBRARY ROUTINES FOR A 1750A AIRBORNE MICROCOMPUTER

THESIS

Jennifer J. Fried
Captain, USAF

AFIT/GCS/MA/85D-3

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

1

AFIT/GCS/MA/85D-3

DEVELOPMENT AND EVALUATION OF MATH LIBRARY ROUTINES FOR A 1750A AIRBORNE MICROCOMPUTER

THESIS

Jennifer J. Fried
Captain, USAF

AFIT/GCS/MA/85D-3

Preface

The purpose of this thesis was to develop, test, and evaluate the performance of run time math library routines for those architectures conforming to MIL-STD-1750A, the instruction set architecture for the airborne computers used in Air Force avionic weapon systems. The routines implemented include several algebraic functions that are intended to serve as a benchmark for future contractor development. Appendix A contains descriptions of the pseudo-operations used to explain the design of these functions, and will be useful in following the logic.

In developing and performing the evaluation of the math library, and in learning how to use the different support tools and hardware, I have had a great deal of help from others. In that respect I am deeply indebted to my thesis advisor, Dr. Panna Nagarsenker, for her continuing patience and assistance when I needed it. Capt Steve Hotchkiss has my undying gratitude for his friendship and help in these trying times. I also wish to thank Mr. Bobby Evans and Mr. Dale Lange, from the sponsoring organization, for all the help that they gave me in getting the needed equipment and outside information. Finally, I am eternally grateful to Tim for his unending love and encouragement.

Jennifer J. Fried

Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Table of Contents

	Page
Preface	ii
List of Figures	v
List of Tables	vi
Abstract	vii
I. Introduction	1
Background	1
Problem	3
Scope	4
Assumptions	5
General Approach	5
Sequence of Pesentation	11
II. Theoretical Development	12
General Discussion	12
Approximation Techniques	13
Polynomials	14
Definition of Chebyshev Polynomials	16
Chebyshev Economization	17
Rational Approximations	19
The Second Algorithm of Remes	21
III. Development and Design of the Functions	27
General Discussion	27
Square Root Implementation	27

Page

Exponential Implementation 32
Natural Logarithm Implementation 38
Common Logarithm Implementation 41

IV. Validation Verification and Performance Evaluation 43

General Discussion 43
Manual Static Analysis Methods 45
Critical Value Testing 45
Performance Evaluation 46

V. Conclusions and Recommendations 49

Conclusions 49
Recommendations 49

Appendix A: Pseudo Operations 54

Appendix B: Function Source Code 59

Appendix C: Support Software 78

Appendix D: VMS Command Files 96

Appendix E: Approximation Algorithms 101

Bibliography 139

Vita 141

List of Figures

Figure | Page

1. Chart of (a) Waterfall; and (b) Logicalized Model of a Software Development Cycle 7

2. Calculation of Remes Rational Approximations 20

3. Square Root Structured Flowchart 28

4. Exponential Structured Flowchart 35

5. Natural Logarithm Structured Flowchart 40

6. Common Logarithm Structured Flowchart 42

7. Bit Layout of 1750A Floating-Point Numbers 70

List of Tables

Table Page

1. Information Flow of the Logicalized Software Development Cycle 8

2. Chebyshev Polynomials and Powers of Chebyshef Polynomials 17

3. Constants for Exponential Determination 34

4. Coefficients for Polynomial Approximation to Exp 38

5. Coefficients for Polynomial Approximation to Alog 59

AFIT/GCS/MA/85D-3

Abstract

This project produced a run-time math library for the MIL-STD-1750A embedded computer architectures. The math library consists of the algebraic functions. In addition, the steps required for a performance analysis of the math library have been outlined.

Several approximation methods were investigated. The Chebyshev Economization of Maclaurin series polynomials, and rational approximations derived from the second algorithm of Remes were determined to be the best methods available. Each function's implementaion was designed to take advantage of features of MIL-STD-1750A architectures. The recommended test procedures provide measures of the average and worst case generated errors within each approximation.

I. Introduction

Background

The Air Force is interested in reducing the life-cycle costs of its avionics weapon systems. Standardization of high order languages and an Instruction Set Architecture (ISA) are two of the many ways the Air Force can reduce these costs. In the past, a major cost contributor was the proliferation of unique avionics systems and subsystems. Costs increased with respect to purchasing and inventorying small-lot spares at many bases, training technicians to maintain complex and/or unique flight and test equipment, developing and maintaining software development facilities, training programmers to write application programs in seldom used high order languages, and training programmers to maintain software (especially operating systems) in seldom used machine languages. (1: 8.1)

MIL-STD-1750A defines a standard 16-bit instruction set architecture intended primarily for avionics weapon systems. The major cost advantage of this standard ISA comes in the form of common support software tools. An extensive set of support software tools has already been developed and includes a 1750A assembler/crossassembler, a J73 compiler with 1750A ISA code generator, a linker program, a loader program, and a 1750A acceptance test program. (1: 8.4) Other cost benefits are realized through the independent development of software and hardware, (2: 1) and common maintenance and test equipment. (3: 168)

Standardization of languages also has an impact on cost reduction. "In 1978, the Department of Defense had in its inventory, software written in about 150 different programming languages. This linguistic proliferation increased maintenance problems due to programmer training requirements and lack of support tools for many of the languages." (1: 6.1) The D.O.D. and Air Force recognized this as a problem, and they took steps to correct it. The D.O.D. Instruction 5000.31, "Interim List of D.O.D. Approved High Order Programming Languages," states that only approved languages may be used for new defense system software. JOVIAL is one of the few languages approved by this instruction.

As previously mentioned, the development of a standard ISA such as MIL-STD-1750A helps reduce total life-cycle costs for Air Force avionics weapon systems. This reduction is partially due to the use of common support software tools, many of which have already been developed. As was previously mentioned, one of the support software tools that has been developed is a JOVIAL compiler that generates 1750A ISA code. However, a math library containing the algebraic and trigonometric functions required by this language has not been developed. The sponsor for this thesis is the Aeronautics System Division, Language Control Branch. They are the D.O.D. JOVIAL and ADA compiler validation site, and are responsible for the development of such libraries. Completion of this thesis, with its development of a math library for software support of 1750A systems, can help the Air Force reduce avionics weapon systems costs.

Problem

Prior to the completion of this thesis effort, there were no math libraries written to take advantage of the 1750A instruction set. In keeping with the intent of recent standardization policies of both the D.O.D and Air Force, the library created by this thesis is written in the D.O.D approved language JOVIAL. Actual coding of the library was only a small part of this thesis. Most of the detail has gone into verification, validation, and evaluation of the product. As such, the focus of this report is divided into two primary categories: software development and software testing.

Math libraries are important because they provide the programmer several tools that serve as building blocks for applications. Math libraries prevent programmers from having to recreate each function whenever one is required for use. Libraries also provides a means for using functions that take full advantage of the computer architecture for which they were written.

The design of a procedure for computing the value of functions is not mathematically complete unto itself. An understanding of a computer architecture's operation is necessary to insure that the computation of any given function is as efficient as possible, while also providing the highest degree of accuracy. Such architectural considerations include word size, number of bits in both the exponent and coefficient fields of a floating point number, the way mathematical operations are performed by the architecture, memory size of the architecture, and execution time. Other considerations include overflow, underflow, and precision. These considerations for the mathematic functions define some of the problems addressed by this thesis effort.

Scope

This effort was limited to the design, code, and evaluation of algebraic functions. The functions were included in a math library targeted for MIL-STD-1750A computer architectures, and are the ones typically found in most FORTRAN libraries. Specifically, these functions include square root (sqrt), exponential (exp), natural logarithm (alog), and common logarithm (alog10).

All functions have been written to accept and return only extended precision floating-point values. The specific floating-point functions are invoked by using the name given above.

Performance summaries for each of the functions, and algorithms are provided in Appendix E. They may be used to determine the polynomial coefficients for computing any of functions addressed by this paper. These algorithms produce coefficients that are valid for any nonvector architecture.

Assumptions

During a design review held in May of 1985, it was made clear that certain events could cause overflow and underflow errors, and division by zero. Since the functions are to be used within an embedded avionics weapon system, it is necessary that such conditions are detected and handled gracefully. The consensus of opinion from all participants of the design review was that the functions should not be aborted, and that default values should be returned. The error conditions and values returned are discussed in the individual design sections of this thesis. This constitutes an important assumption on how to handle such error conditions, and needs further investigation before implementation on a real-time system.

Another factor discussed during the design review was the distinction between fixed-point and floating-point functions. Floating-point functions have greater precision than fixed-point algorithms, but take longer to execute. Although the fixed-point functions are faster, the algebraic math routines and the JOVIAL computer language do not lend themselves to this method of calculation. Therefore, as stated earlier, only the floating-point algebraic math library functions have been implemented.

General Approach

The approach used during this thesis effort, is termed the "logicalized" model of a software system development cycle. This approach was considered a better alternative than the more commonly used "waterfall" method of software system development. The "waterfall" method is composed of a neat, concise and logical ordering of a series of steps, each of which must be accomplished in order to obtain a final software product. These steps are performed in order and include systems analysis, requirements definition, preliminary design, detailed design, coding, testing, and implementation.

The "logicalized" model is similar to the "waterfall" model just described, but it is more concerned with the problem definition part of the cycle (see Figure 1). This approach is more useful in eliminating errors that typically occur during the requirements definition and design phases of the "waterfall" method. Errors generated during these phases typically occur because designers have a tendency to shift between abstract high-level design issues and physical implementation considerations. Thayer (5: 335-41) and Boehm et al. (6: 125-33) made it clear that these problems exist, and that design errors not only outnumber other errors, but that they are also more persistent. For this reason, more attention was given to the top-down decomposition and abstract (logical) modeling of this particular software system. Such a structured approach recommends a dichotomy between the logical design issues, and implementation issues.

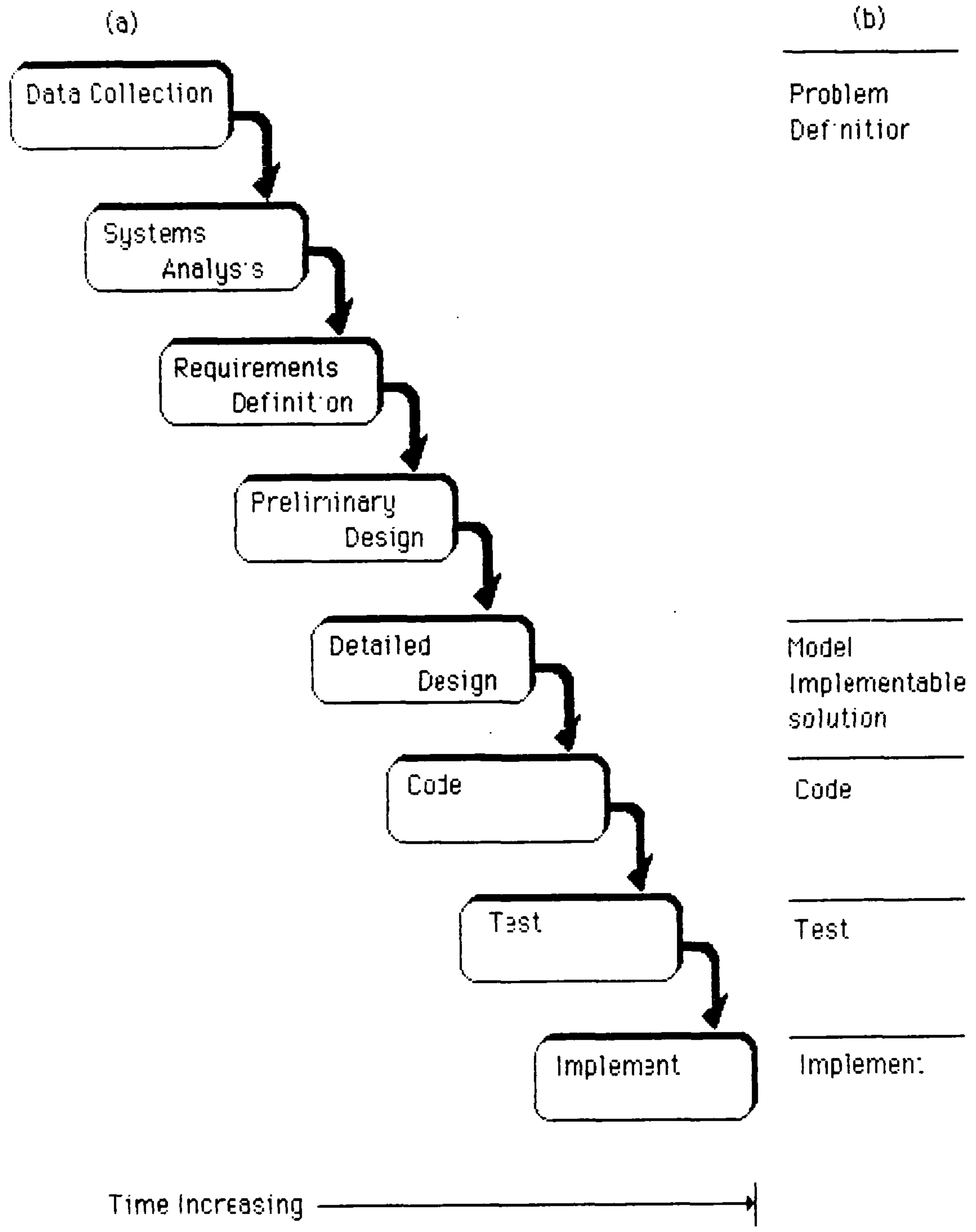


Figure 1 Chart of; (a) Waterfall; and (b) Logicalized Model of a Software Development Cycle

PHASE	INPUT	TASK	OUTPUT
ANALYSIS	Interviews, random data, and so on	Model problem and implied solution	Abstract model of implied solution
DESIGN	Abstract model of implied solution and environmental constraints	Model an implementable solution	Abstract model of implementable solution
CODE	Abstract model of an implementable solution	Implement solution	Executable solution

Table 1 Information Flow of the Logicalized Software Development Cycle

The information flow of a logicalized model is summarized in Table 1, and is "analogous to an artist's conception of a building, i.e. there is enough information to allow the customer and designer to communicate and to establish the buildings pluses and minuses, but not enough detail to begin construction. A series of reviews, refinements, and the imposition of local building ordinances, for example, are necessary before construction can start." (7:14)

Therefore, the approach taken for this project was similar to that just described. The ASD/Language Control Branch established the requirements for a MIL-STD-1750A run-time math library written in the D.O.D approved high-order language JOVIAL. During a design review and several other meetings, certain design considerations were refined. Then a "logical" model was established as a baseline. This was accomplished by using the refined problem statement, and researching the different methods for approximating the different algebraic functions.

The baseline model served as a reference from which all decisions regarding actual implementation could be made. Before proceding to the next phase of development, two such decisions had to be made. These decisions were to determine which testing methods and which performance evaluation techniques would be used after coding was complete. These decisions determined what sort of tests would catch all possible errors, and determined what techniques could be used to establish a confidence level for the final product.

Up until this point, the abstract model has been devoid of any implementation considerations. However, after it was clear that the abstract design was complete and consistent with the requirements, it became necessary to consider changes to fit the problem into the MIL-STD-1750A environment. Before any changes could be made, it was necessary to complete the following steps: study the architecture and ISA defined by MIL-STD-1750A; determine what resources were available, such as software support tools and hardware; and then to learn how to use the available resources. From there, it was possible to develop an abstract model of an implementable solution. This model took advantage of those environmental factors that affected the speed and accuracy of computation for each function approximation.

The major subset of the logicalized software engineering methodology just described is called structured programming. Structured programming can be understood as the decomposition of a problem in order to establish a manageable problem structure. The highest conceptual level represents a general description of the problem, and each level of decomposition provides more detail into the problem. This decomposition is carried out until the problem is almost in coded form, and is often called a stepwise refinement of the problem. All implementation considerations are left until the lowest levels of refinement.

The goals of structured programming must be to minimize: the number of errors that occur during the development process; the effort required to correct errors in sections of code found to be deficient; the effort required to upgrade sections when more reliable, functional, or efficient techniques are discovered; and the life-cycle costs of the software. (8: 32) It must also reduce the complexity of the problem.

Structured flowcharting is a technique used to support these structured programming concepts and goals, and is "designed to reduce labels and unstructured branching, encourage a single entry/single exit approach, aid in the use of top-down design techniques, and enhance modularization. The approach encourages the designer to conceive of the system in high-level constructs and not in terms of individual detailed statements." (7: 116) The structured flowcharting technique was used throughout the development of this project, not only because of the reasons just mentioned, but also for its simplicity and understandability from a reviewer standpoint.

Sequence of Presentation

This thesis addresses the design and performance evaluation of a run-time math library that is targeted to MIL-STD-1750A architectures. The requirements definition for this problem has already been discussed (Chapter 1 - Problem/Scope). The next topic discussed is the theoretical development of this thesis effort (Chapter 2). This is followed by a discussion of the detailed design considerations that were made during implementation of the library functions (Chapter 3). The last aspects covered in this report are the test and performance evaluation methods considered (Chapter 4) and conclusions (Chapter 5).

Appendices include algorithms useful for determining the pseudo-code operations used in the structured flowcharts (Appendix A), source listings for the implemented functions (Appendix B), support software developed in conjunction with this thesis (Appendix C), the VAX VMS command files required to compile, link, and run the developed product (Appendix D), and the coefficients for each of the functions (Appendix E).

II. Theoretical Development

General Discussion

The purpose of this thesis was to create and analyze algebraic functions developed for 1750A architectures. This chapter is concerned with the design theory of the algorithms used to approximate those functions. Within the given constraints, the emphasis for each of the designs is to compute results as quickly and as accurately as possible.

One way of computing a value quickly is to select an approximation that converges rapidly towards the value of the true function, $f(x)$. There were several methods of approximation that were considered; however, the polynomial and relational approximations described by Cody and Waite (4: 17-84) were found to be the best. The coefficients given by Cody and Waite were derived by using Chebyshev Economization of the Taylor series for each function for the approximation itself, or as a starting point for computing a rational approximation via the second algorithm of Remes. An excellent reference for Chebyshev Economization is Conte and de Boor (9: 265-273), and an excellent reference for the second algorithm of Remes is Ralston. (10: 301-306)

Another means of reducing the amount of processing time required to compute a result is to take advantage of certain aspects of the computer's architecture, as well as the different execution times for different instructions within the ISA. For example, incrementing the exponent field of a floating-point value is not only faster, but more accurate than the equivalent operation of multiplying by two, or examining the sign bit of a variable is faster than comparing the entire value to zero. These techniques have been used, and are referenced in the design descriptions as pseudo-operations. These operations are equivalent to those described by Cody and Waite (4: 9), and are listed in Appendix A.

The accuracy of an approximation may be dependent upon the domain over which the function is approximated. For example, if the domain of an approximation is halved, the error may be reduced by a factor of about $2^{-(n+1)}$ for all polynomials of degree n. (11: 59) This can be shown to be true for most functions, but not all of them. Domain reduction has no effect on accuracy in approximations of certain functions; however, it still serves as an excellent guide when designing an application. This is due to the way computer architectures perform operations and store mathematical values for floating-point numbers. The most significant bits of a number are always maintained, and since only a finite number of bits are available to represent the value, it is possible that bits from a fractional representation may be lost during operations on large numbers.

Approximation Techniques

The MIL-STD-1750A ISA doesn't call for the implementation of the elementary functions as standard instruction operators, so it is necessary to design software routines of optimum efficiency to replace them. The word "optimum" could be given a variety of precise definitions, but presumably it refers to an average execution time and storage space.

Unfortunately, there is no known way to derive or prove such an "optimal" design. For these reasons, the search for the appropriate approximation technique was limited to polynomial and rational approximations.

Some of the most popular methods of approximation used are called Chebyshev approximations. Chebyshev approximations are often referred to as "minimax" approximations because they are used to minimize the maximum "error" between the true function $f(x)$, and the approximation of $f(x)$. However, these methods of approximation are not without their problems, and there is a price, even though it is small, to be paid for using them. For example, the sum-of-squares of the errors in a Chebyshev approximation will be higher than if a least-squares method of approximation is used. However, since Chebyshev approximations assure that an error is never greater than a given amount, they were selected by this study.

Polynomials. The first class of approximations discussed are polynomials, and are the simplest of all the classes of approximations considered. The most important subclass of the polynomials is the class τ_n (Chebyshev), and are polynomials not exceeding degree n. The Chebyshev polynomials are especially important, and gave rise to the general concept of Chebyshev "approximations" discussed in the preceding paragraph.

The motivation for using Chebyshev polynomials over all other polynomials is their property of least maximum error, and their error behavior over the entire interval of the approximated function. Through the use of Theorem 1, the Alternation Theorem given below, Chebyshev was able to prove for all the polynomials of degree n with a leading coefficient of 1, that the Chebyshev polynomial divided by 2^{n-1} has the least maximum error in the interval [-1,1]. In other words, no other polynomial of the type mentioned will have a smaller error than $T_n(x)/2^{n-1}$. In order for a polynomial $P_n(x)$ to be considered a Chebyshev approximation of the function $f(x)$, the theorem requires that the maximum discrepancy between $f(x)$ and $P_n(x)$ occur with alternating signs at $n+2$ points over the interval [-1,1].

> <u>Alternation Theorem</u>: The polynomial P_n of degree $\leq n$ that (1) best approximates f is characterized by the existence of at least $n+2$ "points of alternation"

The other motivation for the use of Chebyshev polynomials is that its generated errors are more well behaved than the errors generated by other polynomials. For example, approximations, based on the Maclaurin series whose interval includes zero, have errors that are very nonuniform -- small near the middle, but very large at the end points. It is more desirable to use an approximation whose behavior is more uniform instead of powers of x. Since, as stated in Theorem 1, the Chebyshev polynomials spread the error over the entire interval, they provide this more desirable behavior.

<u>Definition of the Chebyshev Polynomials</u>. The Chebyshev polynomials form an orthogonal set, and are defined by the following equation.

$$T_n(x) = \cos(n\theta) \qquad \theta = \arccos(x), \quad n = 1, 2, \ldots \qquad (1)$$

From elementary trigonometry, $\cos(n\theta)$ is a polynomial of degree n in $\cos(\theta)$, and $\cos(\arccos(x)) = x$; therefore, it follows that the Chebyshev polynomials defined by $\tau_n(x) = \cos(n \arccos(x))$ are polynomials of degree n.

By substituting $\arccos(x)$ for θ and $\tau_n(x)$ for $\cos(n \arccos(x))$ in the identity function shown in equation (2), the recurrence relation defined in (3) is formed.

$$\cos((n + 1)\theta) + \cos((n - 1)\theta) = 2 \cos(\theta) \cos(n\theta) \qquad (2)$$

$$\tau_{n+1}(x) = 2x\, \tau_n(x) - \tau_{n-1}(x) \qquad (3)$$

Let $\tau_0 = 1$ and $\tau_1 = x$, then from the recurrence relation defined in (3), successive polynomials of greater degree can be generated as in column A of Table 2.

By using the results in column A of Table 2, the powers of the Chebyshev polynomials can be found. That is, it is possible to express the powers of x in terms of τ_n. An example of the powers of τ are shown in Table 2 column B Appendix A contains an algorithm that generates both the Chebyshev polynomials, and their powers.

$\mathcal{A}$	$\mathbf{B}$	
$\tau_0 = 1$	$x^0 = 1$	$= \tau_0$
$\tau_1 = x$	$x^1 = x$	$= \tau_1$
$\tau_2 = 2x\tau_1 - \tau_0 = 2x^2 - 1$	$x^2 = \frac{1}{2}(2x^2 - 1 + 1)$	$= \frac{1}{2}(\tau_2 + \tau_0)$
$\tau_3 = 2x\tau_2 - \tau_1 = 4x^3 - 3x$	$x^3 = \frac{1}{4}(4x^3 - 3x + 3x) =$	$\frac{1}{4}(\tau_3 + 3\tau_1)$

Table 2 ($\mathcal{A}$) Chebyshev Polynomials; ($\mathbf{B}$) Powers of Chebyshev Polynomials

Chebyshev Economization. As already mentioned, the Maclaurin series can be used to approximate many functions. In addition to the disadvantages that have already been mentioned for using this series as an approximation, the Maclaurin series also converges very slowly. That is, it takes several multiplications and additions to obtain a desired accuracy. One way of obtaining a lower degree polynomial, and still maintain the desired accuracy, is to use a technique that is called "telescoping" or "Chebyshev Economization". In other words, the polynomial can be expressed in a manner similar to that shown in (4).

$$P_n(x) = d_0\tau_0(x) + \ldots + d_n\tau_n \tag{4}$$

To compute the economized polynomial approximation to the function $f(x)$ of absolute accuracy ε on the interval $[-1, 1]$, use the following procedure as outlined by Conte et.al. (9: 271-272)

Step 1. Get a power series expansion for $f(x)$ valid on $[-1, 1]$; typically, calculate the Maclaurin or Taylor series expansion for $f(x)$ around $x = 0$.

Step 2. Truncate the power series to obtain a polynomial as in (5), which approximates $f(x)$ on $[-1, 1]$ within an error ε_n, where ε_n is smaller than ε, and ε_n is defined as in (6). The result of ε_n is the maximum absolute value, within the interval $[-1, 1]$, of the product of the first truncated coefficient, x to the power of $n + 1$, and the $n + 1$ derivative of the function $f(x)$.

$$P_n(x) = a_0 + a_1 x + \ldots + a_n x^n \tag{5}$$

$$\varepsilon_n = R_n(x) = a_{n+1} x^{n+1} f^{(n+1)}(x) \tag{6}$$

Step 3. By making use of a table similar to that shown in Table 2 column **B**, expand the polynomial $P_n(x)$ into a Chebyshev series as defined in (4). In other words, substitute the far right-hand-side of the equations in Table 2 column **B**, with the appropriate powers of x contained in the polynomial formed by Step 2 of this algorithm. The result is similar to that shown in (7), but of a greater degree.

Step 4. Retain the first $k + 1$ terms in this series, i.e. find equation (7), choosing k as the smallest possible integer such that equation (8) holds true.

$$P^{*}_{k}(x) = d_0 \tau_0(x) + \ldots + d_k \tau_k \tag{7}$$

$$\varepsilon_n + d_{k+1} + \ldots + d_n \leq \varepsilon \tag{8}$$

Step 5. Convert the result of Step 4 into a power series polynomial similar to (5), by making use of a table similar to that in Table 2 column $\mathcal{A}$. In other words, substitute the right-hand-side values of Table 2 column $\mathcal{A}$, into the equation formed by Step 4. Simplify the result.

Rational Approximations. In most instances, rational approximations will generate a least maximum error that is as small or smaller than a Chebyshev polynomial, and will also cost less in terms of the number of multiplications and additions required to compute them. Therefore, they deserved attention in this study.

As stated earlier, the approximation techniques considered by this thesis are classified as Chebyshev approximations. These methods, through their exploitation of Theorem 1, provide approximations whose maximum error is less than those generated by other techniques. There are several algorithms that generate rational approximations that can be considered Chebyshev approximations; however, the ones that generate the

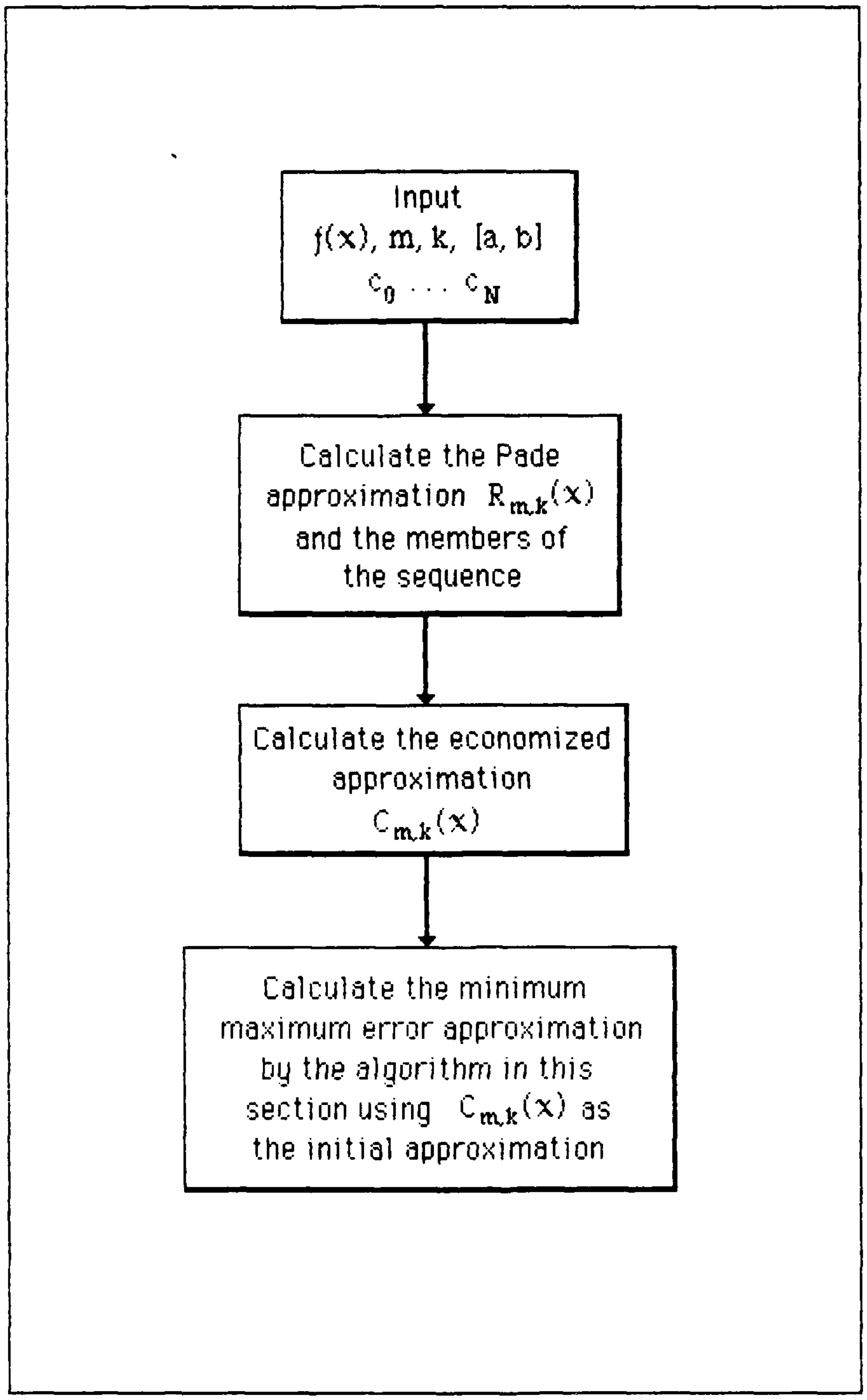

Input
$f(x)$, m, k, $[a, b]$
$c_0 \ldots c_N$
Calculate the Pade approximation $R_{m,k}(x)$ and the members of the sequence
Calculate the economized approximation $C_{m,k}(x)$
Calculate the minimum maximum error approximation by the algorithm in this section using $C_{m,k}(x)$ as the initial approximation

Figure 2 Calculation of Remes Rational Approximations

most uniform approximations are those generated by the second algorithm of Remes. This algorithm is easily automated, and is described in detail by the following subsection.

The Second Algorithm of Remes. The method used in this description is similar to that outlined by Ralston (10: 301-305), and is summarized in Figure 2.

Let $f(x)$ be a continuous function that is to be approximated over the the interval [a, b], and let the interval include the point 0.0. Furthermore, let (9) equal the error of any rational approximation of the form shown in (10).

$$r_{m,k}(x) = \max \mid f(x) - R_{m,k}(x) \mid \tag{9}$$

$$R_{m,k} = \frac{P_m(x)}{Q_k(x)} = \frac{\sum_{j=0}^{m} a_j x^j}{\sum_{j=0}^{k} b_j x^j} \tag{10}$$

Step 1 of the algorithm names the input required for this algorithm. The input value $f(x)$ is the function being approximated. If the algorithm is being run on a machine with higher precision than the error for which the function is being approximated, then the built-in functions of the machine can be used for $f(x)$. If the machine that the algorithm is to run on is of the same precision for which the approximation is to be made, then a reasonable substitute, such as a truncated power series that is of equal or greater precision than what is being approximated, can be used.

The other inputs include: m, k, $[a, b]$, and $C_0 \ldots C_N$. The values m, and k represent the degree of the polynomials found in the numerator and denominator, respectively. The interval $[a, b]$ is the interval for which the approximation is valid, and should include the point zero, as it will allow the coefficient b_0, of the denominator, to always be one. The values $C_0 \ldots C_N$ represent the first $N + 1$ coefficients of the power series polynomial that is being converted to a rational approximation. The value N represents the sum of the degree of the polynomials used in the numerator and the denominator $(m + k)$.

The second step of the algorithm is to compute a series of Pade approximations and their error coefficients. The Pade approximations are of the form depicted in (11), with the restrictions that $0 \leq i \leq m$ and $0 \leq j - i \leq k$. For example, the sequence of Pade approximations computed for an $R_{2,2}$ approximation would only include $R^{(0)}{}_{0,0}(x)$, $R^{(1)}{}_{1,0}(x)$, $R^{(2)}{}_{1,1}(x)$, $R^{(3)}{}_{2,1}(x)$, and $R^{(4)}{}_{2,2}(x)$. The error of the approximations is equal to the first power of x truncated from the power series, multiplied by the error coefficients shown in (12). The error calculations used would only include: $d^{(0,0)}{}_1$, $d^{(1,0)}{}_2$, $d^{(1,1)}{}_3$, $d^{(2,1)}{}_4$, and $d^{(2,2)}{}_5$.

$$R^{(j)}_{i,j-i}(x) = \frac{P^{(j)}_i(x)}{Q^{(j)}_{j-i}(x)} \qquad j = 0, \ldots, N-1 \tag{11}$$

$$d^{(m,k)}_{N+1} = \sum_{j=0}^{k} C_{N+1-j}\, b_j \tag{12}$$

The coefficients for each of the sequence of Pade approximations are computed using (13) and (14). Equation (13) forms a set of **m** linear equations, which when solved, determines the value of the coefficients used in the denominator. Those values can then be directly substituted into the set of equations formed by (14), and will determine the value of the coefficients for the numerator.

$$\sum_{j=0}^{k} C_{N-s-j}\, b_j = 0 \qquad s = 0, 1, \ldots, N-m-1 \quad (C_j = 0 \ \text{ if } j < 0, \ b_0 = 1) \tag{13}$$

$$a_r = \sum_{j=0}^{r} C_{r-j}\, b_j \qquad r = 0, 1, \ldots, m \quad (b_j = 0 \ \text{ if } j > k) \tag{14}$$

The third step of the Remes algorithm is to compute the economized approximation $C_{m,k}(x)$. To complete this step, it is necessary to compute the Chebyshev polynomial τ_{N+1}. This polynomial can be determined by using equation (3) of the previous subsection. Once the coefficients of τ_{N+1} are found, then the values γ from (15) can be directly substituted into (16), and thus solve $C_{m,k}(x)$. The value t_j in (15) is the coefficient for u^j in $\tau_{N+1}(u)$. The rational approximation must also be normalized, that is, the numerator and denominator must be divided by b_0, such that b_0 will remain equal to 1.

$$\gamma_0 = -d_{N+1}^{(m,k)} t_0/2^N \qquad j = 0, \ldots, N-1 \qquad (15)$$

$$\gamma_{j+1} = \frac{d_{N+1}^{(m,k)} \; t_{j+1}}{d_{j+1}^{(i,j-1)} \; 2^N}$$

$$C_{m,k}(x) = \frac{P_m(x) + \sum_{j=0}^{N-1} \gamma_{j+1} P_i^{(j)}(x) + \gamma_0}{Q_k(x) + \sum_{j=0}^{N-1} \gamma_{j+1} Q_{j-1}^{(j)}(x)} \qquad (16)$$

The final step of the Remes algorithm is an iterative one. Now that the initial approximation to the function has been found, it becomes necessary to find the N + 2 points of alternation. This can be done through interpolation, or by dividing the interval into several small pieces and solving for each point on a division. This method works, and all that is necessary is a little bookkeeping to maintain a list of the N + 2 points of alternation. This step consists of the following three procedures.

Procedure 1. Solve the system of $N+2$ equations for the $N+2$ unknowns $a_0^{(0)}, \ldots, a_m^{(0)}, b_1^{(0)}, \ldots, b_k^{(0)}$, and $E^{(0)}$ as shown in expression (17). Note that $E^{(0)}$ is the magnitude of error in the approximation at each of the points $x_i^{(0)}$, and for the first iteration can be assumed to be 0.

$$f[x_i^{(0)}] - \frac{\sum_{j=0}^{m} a_j [x_i^{(0)}]^j}{\sum_{j=0}^{k} b_j [x_i^{(0)}]^j} = (-1)^i E \tag{17}$$

Procedure 2. Find $h_0(x)$ as shown in (18). The function $h_0(x)$ then has a magnitude of $|E^{(0)}|$ with alternating signs at x_i, $i=0, \ldots, N+1$. In the neighborhood of each $x_i^{(0)}$, there is a point $x_i^{(1)}$ at which $h_0(x)$ has an extremum of the same sign as that of $f(x) - R^{(0)}_{mk}(x)$ at $x_i^{(0)}$. Replace each $x_i^{(0)}$ by the corresponding $x_i^{(1)}$. If x', the point at which $h_0(x)$ has its maximum magnitude, is one of the points $x_i^{(1)}$, do not perform procedure 3. If not, replace one of the points $x_i^{(1)}$ by x' in such a way that $h_0(x)$ still alternates in sign on the points $x_i^{(1)}$.

$$h_0(x) = f(x) - \frac{\sum_{j=0}^{m} a_j^{(0)} x^j}{\sum_{j=0}^{k} b_j^{(0)} x^j} \qquad [\, b_0^{(0)} = 1 \,] \tag{18}$$

Procedure 3. Repeat procedures 1 and 2 using the points $x_0^{(1)}, \ldots, x_{N+1}^{(1)}$ in (17). This process generates a sequence of rational approximations which will converge to an optimum if the initial extrema were sufficiently close.

III. Development and Design of the Functions

General Discussion

This chapter deals with the detailed design of each of the specific functions. Each design has an associated structured flowchart, and each box within the flowchart has been numbered for ease of reference. Pseudo-operations are used throughout each of the flowcharts, and includes those defined by Cody and Waite (4: 9-10). Furthermore, a few additional pseudo-operations have been introduced. (see Appendix A)

Although the approximation methods used are those suggested by Cody and Waite, the actual design implementations are significantly different. The designs proposed by Cody and Waite are guidelines for a broad class of computer, and weren't specifically targeted towards a 1750A architecture. Therefore, the designs have been tailored somewhat.

The coefficients for each of the functions were either taken from Cody and Waite, or are modifications of those provided by Cody and Waite. These modifications are discussed in their appropriate subsection.

Square Root Implementation

The square root of every non-negative floating point number "X" can be computed. Computation is composed of three steps: the reduction of

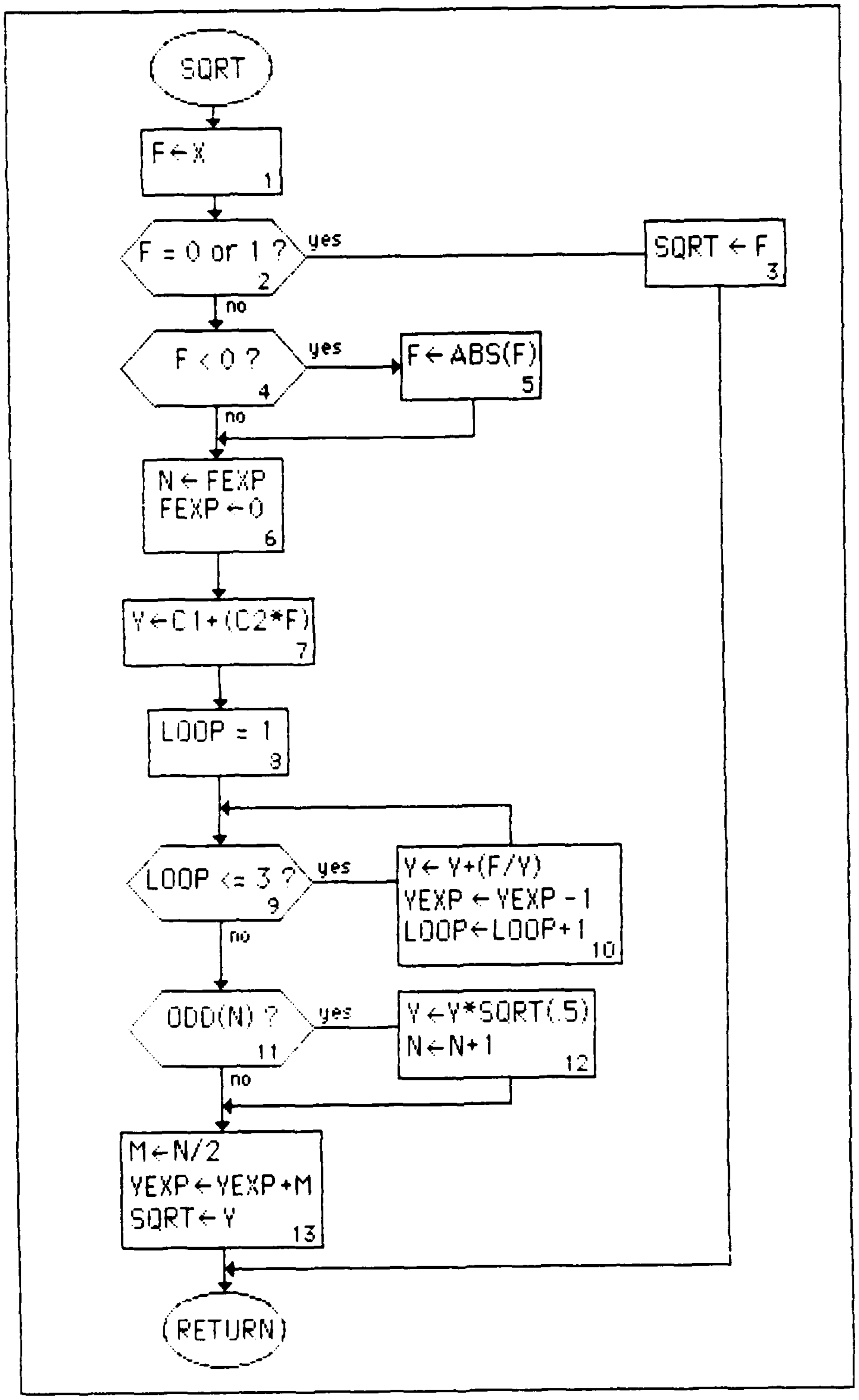


Figure 3 Square Root Structured Flowchart

the given argument "X" into the parameters "f" and "e" using base 2,

$$X = f * 2^{e}, \qquad 1/2 \le f < 1 \tag{19}$$

$$\mathrm{sqrt}(X) = \mathrm{sqrt}(f) * 2^{e/2}, \qquad \text{if } e \text{ is even} \tag{20}$$

$$\mathrm{sqrt}(X) = [\, \mathrm{sqrt}(f) / \mathrm{sqrt}(2) \,] * 2^{(e+1)/2}, \qquad \text{if } e \text{ is odd} \tag{21}$$

the computation of **sqrt** (f), and the reconstruction of **sqrt** (X) from the results.

The variable "X" is the argument passed to the square root function. Since JOVIAL treats formal arguments as read only, upon program entry, the value of "X" is assigned to "F". (step 1 of Figure 3) The variable " F" is then used throughout the remainder of the procedure.

The next step is to check if "F" is either zero or one. (step 2 of Figure 3) If it is, then "F" is its own square root; therefore, this value is returned by the procedure. (step 3 of Figure 3). If "F" was neither a zero or a one, then it is checked to see whether it is negative. If it is negative, under normal circumstances, an error would be assumed, and the procedure would terminate rather than evaluating for a complex number. Due to the nature on embedded avionics systems, an error should not be fatal. In this light, rather than attempting to evaluate a complex number, the absolute value of the input argument is formed. (steps 4 and 5 of

Figure 3) The built-in function for absolute value was found to give inconsistent results, so the absolute value is found by: **F** = - **F**. If this method of error correction proves inappropriate at a later date, it would not be difficult to modify. Perhaps a different default value should be assumed, or an error indicator could be established.

The next step of the algorithm is to obtain the exponent portion of the input argument. (step 6 of Figure 3) JOVIAL's specified tables make this an easy conversion. When the input argument was placed into "**F**", "**F**" had previously been established as a table whose elements identify the components of the floating-point number. Therefore, the item "**Fexp**" is actually the exponent portion of the floating-point number. Immediately following the extraction of the exponent, this same exponent portion of the floating-point number "**F**" is cleared or set to zero.

The next few steps (steps 7 through 10 of Figure 3) are to compute a polynomial approximation for **sqrt** (**F**). Specifically, the computation begins with an initial approximation of "y_0" shown in (22) with successively more accurate approximations being obtained through the use of Newton's iteration in the form of Heron's formula.

$$y_0 = .41731 + .59016 * f \tag{22}$$

$$y_i = (y_{i-1} + f/y_{i-1}) / 2 \tag{23}$$

The coefficients used in this algorithm are those presented by Cody and Waite. (4: 23) The approximation described by Cody and Waite is in the form shown in (23). Aside from the original calculation of "y_0", by examining steps 8 through 10 of Figure 3, note that the Newton iteration is performed three times. Since each iteration doubles the number of correct significant digits in the square root, this assures an accuracy of 63.32 bits. (4: 23) The next step is to determine whether or not the exponent field from the floating-point number originally input was odd or even. (step 11 of Figure 3) Depending upon the result of this evaluation, different actions are taken. If the number is odd, additional calculations are necessary as shown in equation (21). The instruction for determining whether the number is odd is not a separate function. The power of the JOVIAL language permit testing of specific bits. Using this tool, the low-order bit can be checked to determine if it is zero or one. A zero signifies an even number and a one signifies an odd number which is just what the procedure checks. Given that multiplication is preferred multiplications are more efficient than division , the calculation **sqrt(f) / sqrt(2)** is represented by y_j * **sqrt(.5)**, where the decimal representation of **sqrt(.5)** is the constant .7071067811865.

The final step prior to returning with the result is to form the exponent portion of the result. (step 13 of Figure 3)

Exponential Implementation

There are three steps in calculating the exponential of a floating-point number. The first step is the reduction of the given argument to a related argument in a small interval symmetric about the origin. The second step is the computation of the exponential for the reduced argument, and the final step is the reconstruction of the desired function from its components.

The exponential is formed using the following general procedure. Let

$$X = N * \ln(2) + g, \text{ with } |g| \leq \ln(2)/2 \quad \text{then}$$

$$\exp(X) = \exp(g) * C^N \qquad (24)$$

The accuracy of g is the basis for the accuracy of the function value. Let $y = \exp(g)$, then $dy / y = dg$. This means that the relative error in $\exp(g)$ is approximately the absolute error in g. This error is proportional to the magnitude of X when X is exact because of the finite word length of the computer. The only way to achieve small absolute error in g is to extend the effective precision of the computer during the computation of g. In most cases, the following computation is used.

$g = [(X_1 - N * C_1) + X_2] - N * C_2$ where

$X_1 + X_2 = X$,

X_1 **is the integer part of** X ,

$C_1 + C_2$ **represents ln (C) to more than working precision**

This method gives extra digits of precision equivalent to the number of extra digits in the representation of **ln** (C) when N is small enough that $N * C_1$ is representable exactly in the machine. If this exact representation cannot be accomplished, the computation is equivalent to not using extra precision. Therefore, the magnitude of N has a practical limit which results in a limit on the magnitude of X.

There is also a largest and smallest X such the **exp**(X) can be represented in the machine. For example, if SMALLX is the smallest positive floating-point number and BIGX is the largest without causing overflow, then **exp** (X) can be represented only for those values of X that between **ln** (SMALLX) and **ln** (BIGX). The value $N * C_1$ will be representable exactly in a machine for any X within the specified bounds, because a C_1 can always be chosen to fit the bound. Obviously, careful argument reduction cannot compensate for inaccuracies in X. (4: 61)

BIGX	88.02969193111
SMALLX	-89.41598629223
EPS	9.094947017729E-13
ONEOVERLN	1.4426950408890
LN2	0.6931471805599

Table 3 Constants for Exponential Determination

The variable "**Arg**" is the argument passed to the square root function. Since JOVIAL treats formal arguments as read only, upon program entry, the value of "**Arg**" is assigned to "X". (step 1 of Figure 4) The variable "X" is then used throughout the remainder of the procedure.

The constant "**BIGX**" (see Table 3), which has been assigned a value that is slightly less than the natural logarithm of the largest positive finite floating-point number (step 2 of Figure 4), is compared with the input argument. If the argument is larger that this value, an error would occur during calculating its exponential. Since this application is destined for embedded avionics systems, a solution to this error situation must be found that does not result in a degradation of the system. The selected solution involves replacing the input argument with the constant "**BIGX**". Obviously, other possible options are available to resolve the error condition, and another solution can easily replace the existing methodology.

The constant "**SMALLX**" (see Table 3), which has been assigned a value that is slightly greater than the natural logarithm of the smallest positive finite floating-point number (step 4 of Figure 4), is compared

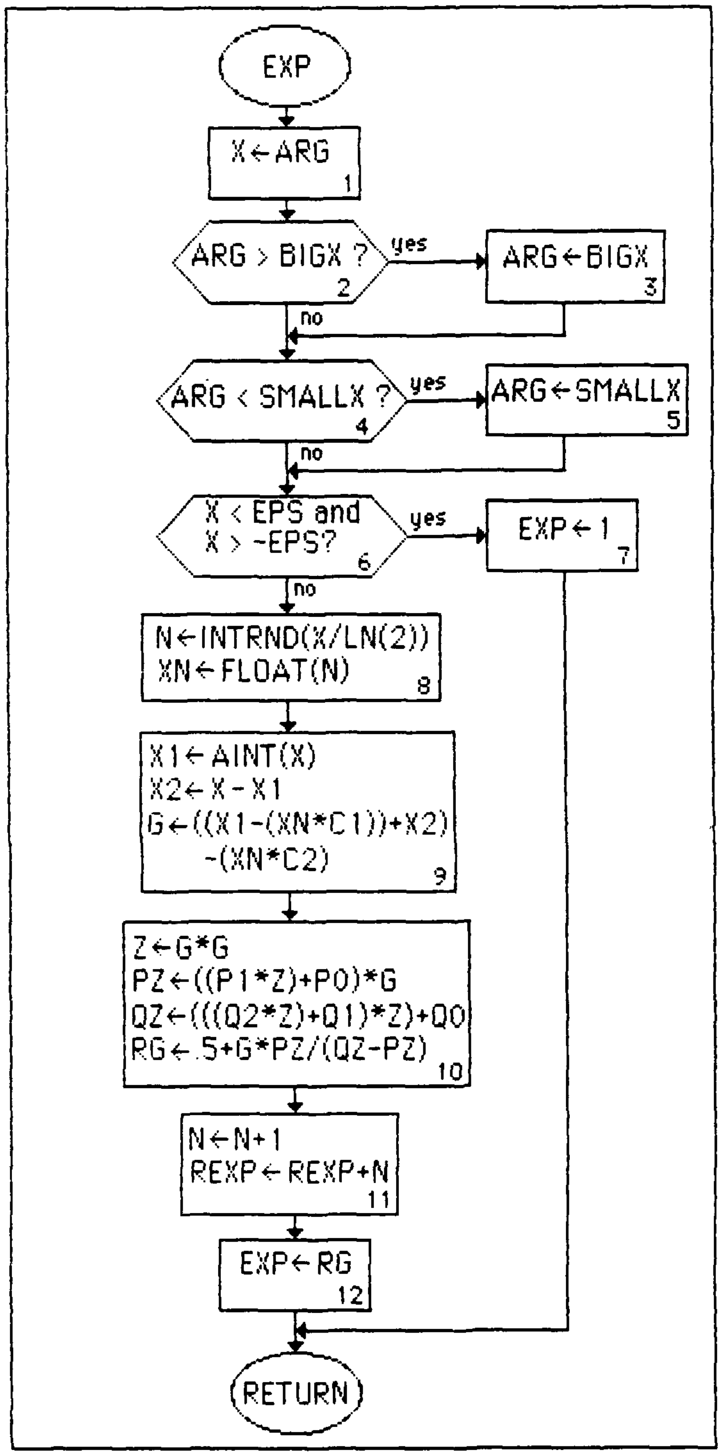

EXP
X←ARG 1
ARG > BIGX ? 2
yes
ARG←BIGX 3
no
ARG < SMALLX ? 4
yes
ARG←SMALLX 5
no
X < EPS and X > -EPS? 6
yes
EXP←1 7
no
N←INTRND(X/LN(2)) XN←FLOAT(N) 8
X1←AINT(X) X2←X-X1 G←((X1-(XN*C1))+X2) -(XN*C2) 9
Z←G*G PZ←((P1*Z)+P0)*G QZ←(((Q2*Z)+Q1)*Z)+Q0 RG←.5+G*PZ/(QZ-PZ) 10
N←N+1 REXP←REXP+N 11
EXP←RG 12
RETURN

Figure 4 Exponential Structured Flowchart

with the input argument. If the argument is smaller that this value, an error would occur during calculating its exponential. Again the discussion in the previous paragraph concerning the resolution of an error condition in an embedded avionics application still holds true. The selected solution involves replacing the input argument with the constant "**SMALLX**". Obviously, other possible options are available to resolve the error condition, and another solution can easily replace the existing methodology.

The next step is to check if "X" is either larger than a positive **eps** or smaller than a negative **eps**. (step 6 of Figure 4) In either case, if it is, the exponential function returns a value of 1 and terminates processing. The value of **eps** (see Table 3) is selected with **exp** (X) = 1.0 to machine precision such that $|X|$ < **eps** and $p_1 * X^2$ will not underflow for $|X| \leq$ **eps**. Cody and Waite have suggested that **eps** = $2^{-t}/2$ where there are t base-2 digits in the significand.

The next step (step 8 in Figure 4) involves extracting the integer portion of the floating-point number that results from the following the calculation: X * **[1 / ln (2)]**. As noted in the description of the square root function, multiplication is not as costly as division. Therefore, the value of **1 / ln (2)** has been calculated and used as a constant. (see Table 3) This newly formed integer is then transformed into a floating-point number. The JOVIAL specified table construct is put to use here rather than calling the two functions INTRND and FLOAT. Extraction of a specific portion of a floating-point number simply involves naming its component parts and using these names to access the needed part.

This particular construct is an extremely efficient method for doing this type of accessing, and is not confined to JOVIAL. It is also available in the Ada language.

The computation provided by Cody and Waite that is specified for no guard digits is use to create a new, more precise number.

$$g = [(X_1 - XN * C_1) + X_2] - XN * C_2, \quad \text{where} \tag{25}$$

X_1 = the floating-point value of the integer portion of X,

$X_2 = X - X_1$,

$C_1 = 0.693359375$,

$C_2 = -2.1219444005470E-4$

Now, that the value of the values of the X's and N's are known, equation (25) can be evaluated for **g**. (step 9 in Figure 4) This is followed by the determination of the rational functions **R (g)** which approximate **exp (g) / 2**. The factor of **0.5** is inserted to counteract wobbling precision. The calculation of the coefficients for the approximation are determined by the number of bits in the significand. For this architecture, the number of bits selected are between 30 and 42 inclusively. This results in the coefficient list of Table 4 on the next page.

p_0	0.24999 99999 999 E+0
p_1	0.59504 25497 759 E-2
q_0	0.50000 00000 000 E+0
q_1	0.53567 51764 522 E-1
q_2	0.29729 36368 224 E-3

Table 4 Coefficients for Polynomial Approximation to Exp

The first step in calculating **Rg** (step 10 in Figure 4), requires the formation of g^2. This value is then used to form $g * P(z)$ and $Q(z)$ using nested multiplication. These values are then used to form **Rg**. Just prior to returning the value generated through all these calculations, an additional step is performed to rescale the number. (step 11 in Figure 4)

Natural Logarithm Implementation

The calculation of the logarithm required three steps. First, the given argument is reduced to a related argument in a small, logarithmically symmetric interval about one. The second step involves the computation of the logarithm for this reduced argument. Finally, the desired logarithm must be reconstructed from its components.

Upon entry into this routine, the value of the input is checked to see if it is either zero or less than zero. (step 1 of Figure 5) If it is either zero or negative, under normal circumstances, an error would be assumed, and the procedure would terminate. In this function, the negative value of the largest floating-point number is returned. (step 2 of Figure 5) As previously mentioned, due to the critical nature of embedded avionics systems, an error should not be fatal. It should provide an alternate path to a graceful completion of the function.

Many methods exist for calculating the logarithm of a reduced argument. Cody and Waite have chosen the following method. (4, 42) The initial assumption is made that the argument is in the following form

$$X = \pm f * 2^{e}, \quad \text{where} \quad .5 \leq f < 1$$

Determine the value of N and the scaled value of f such that

$$X = f * 2N, \quad \text{where} \quad .5 \leq f < 1$$

Initially, f is assigned the value of the input argument. This allows for modification of the input floating-point number. Then an estimate for N is made. N is given the value of the exponent of the input floating-point number, and then this same exponent field is erased. (step 3 in Figure 5)

The value of **sqrt(.5)** has been previously determined and stored as a constant for use by this routine. Depending on the value of f, one of two

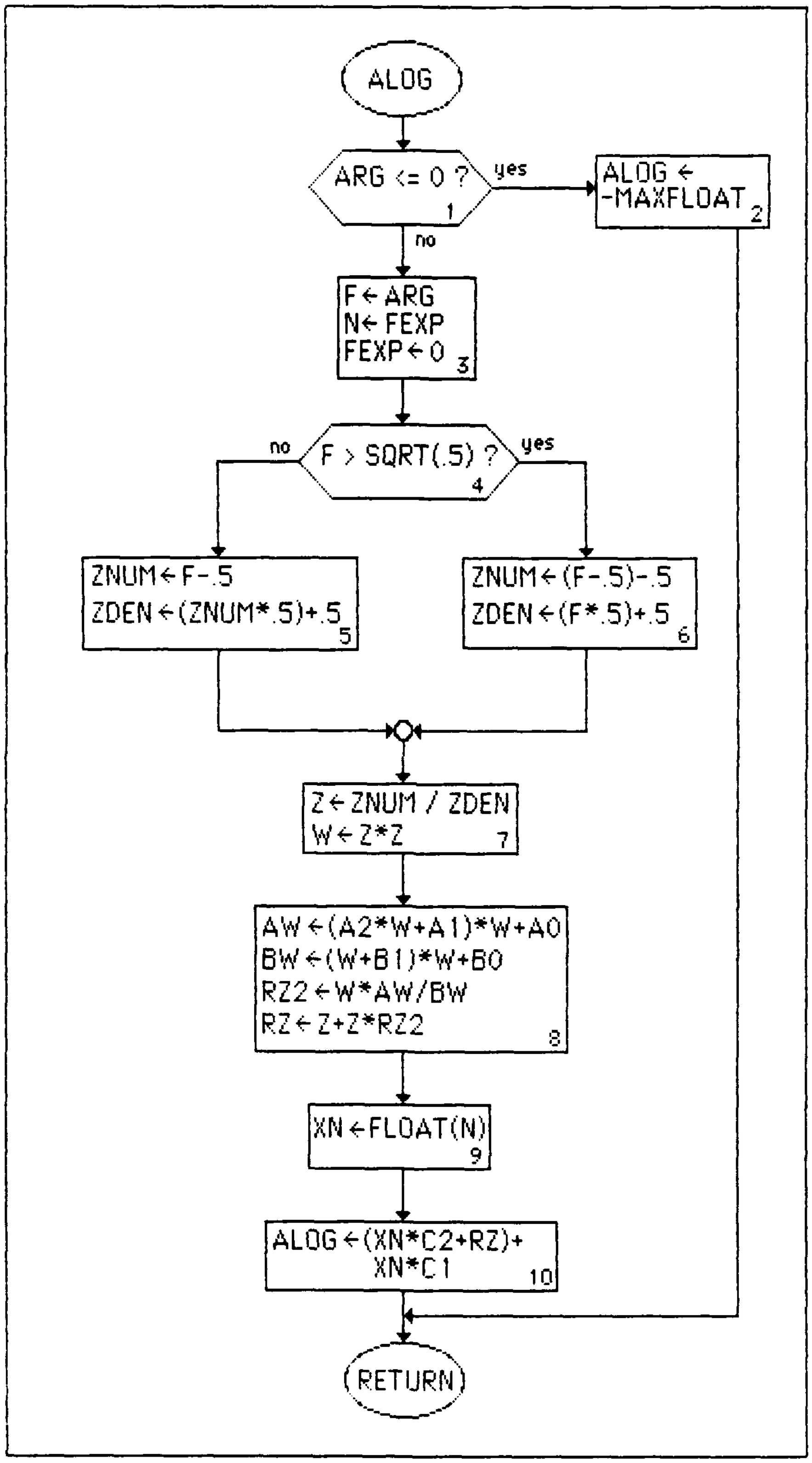


Figure 5 Natural Logarithm Structured Flowchart

a0	0.3733916896316E+1
a1	-0.6326086623386E+0
a2	0.4444551510980E-2
b0	0.4480700275574E+2
b1	-0.1431235435589E+2
b2	0.1000000000000E+1

Table 5 Coefficients for Polynomial Approximation to Alog

distinct paths may be taken. The value of **f** is compared to the **sqrt(.5)**, and **znum** and **zdem** will vary accordingly.

After forming **z = znum / zdem and w = z^2**, evaluate **r (z^2) = w * A(w) / B(w)**. Both A(w) and B(w) are polynomials in the w coefficients given in Table 5. (step 8 in Figure 5)

<u>Common Logarithm Implementation</u>

Obviously, from the structure chart for this function (Figure 6), all the work is done by the Alog function. Since the JOVIAL language does not support multiple entry points, the common logarithm function had to be formed in this manner. The result of this function is generated through the multiplication of the natural logarithm of the input argument with the natural logarithm of "**e**". This latter item is encoded as a constant to avoid wasted effort to recalculate for every use of this function. All the

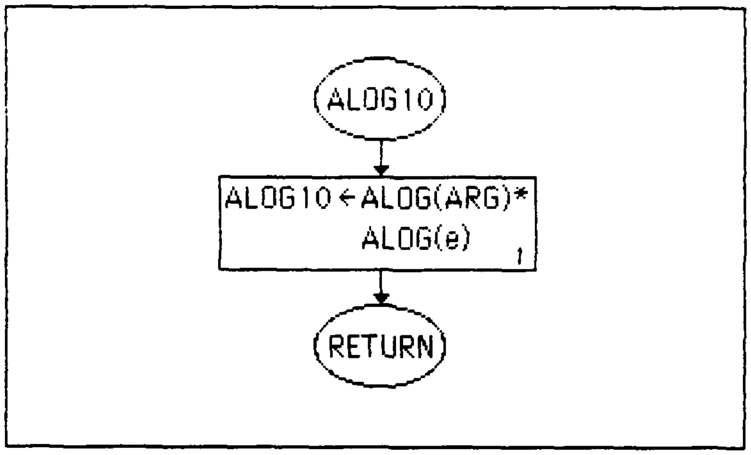


Figure 6 Common Logarithm Structured Flowchart

restrictions that were imposed on the natural logarithm of a number also apply here.

IV Validation Verification and Performance Evaluation

General Discussion

This chapter is concerned with describing the methodology used for determining the correctness and performance qualities of the implemented functions. Due to problems in the availability of hardware and the associated support software, the testing and performance evaluations are somewhat limited. Hardware became available towards the middle of the thesis effort, but software tools used for development were incompatible with those required by the available 1750A. The loader used by the available 1750A equipment, expects files of a different format than what is created by the software development tools. Rather than developing a new loader, a routine was written that converts load modules into a format required by the 1750A loader. The reformatting procedure is listed in Appendix C.

Another problem that had to be overcome before testing and evaluation could be considered, was the availability of input/output (I/O) routines. Without I/O routines, further considerations for testing would be fruitless. No I/O packages were available, and as a consequence, had to be created. This delayed testing efforts considerably, as an I/O routine had to be developed with the use of the MIL-STD-1750A standard ISA, rather than with a high-order language. The I/O package developed is listed in Appendix B, and is only capable of writing to a user console.

Performance analysis requires the comparison of 1750A results, with those generated on a machine of higher precision. Unfortunately, this requirement made the newly created I/O routine insufficient for this task. An available console driver has a routine that writes user specified areas of 1750A memory to magnetic disk. By storing a function's results in a specified area of 1750A memory, the test results can then be dumped to disk for an eventual upload to a VAX 11/780A. The results are then available for input to the different software test packages. However, the record format of the 1750A memory dump is not in a friendly format, and must be converted to a readable form. At the time of this writing, a routine for making the disk file readable is not completely debugged. However, it is at a point where it could be completed by another programmer.

The aforementioned problems have limited the amount of time available for designing extensive test procedures. Therefore, validation, verification, and performance analysis is confined to: manual static analysis methods, critical value testing, and measurement of each algorithms generated error.

Manual Static Analysis Methods

To most people, manual static analysis is called "desk checking" . Static analysis involves the search for any inconsistencies between design tools (i.e. flowcharts), design details (chapter 3), program headers, and program comments. This method is useful for finding errors caused by the translation of design into code, as well as possible design errors. An inconsistency may indicate potential problems. This methodology was used, and all inconsistencies that were found were resolved.

Critical Value Testing

Critical value testing is an attempt to "break" the software, and requires the selection of specific arguments that could possibly cause problems. A knowledge of each of the algorithms is required to select proper arguments. Individual test cases are not listed here, but the reader may find specific information by examining the test procedures listed in Appendix B.

It is possible to generalize the tests performed without listing the specific test cases. Potential test arguments are those whose intermediate results could generate an overflow or underflow, or are arguments lying in the fringe of computational abnormality. These arguments will help detect problem areas, and will give an indication as to how robust each function is.

In addition, arguments that test each path of the algorithm have been selected. Path testing is limited to insuring that every path of an algorithm is tested, and does not imply that every possible path combination is taken.

Performance Evaluation

As was mentioned in the introduction of this chapter, screen output to the user console and hard copies of computed results are insufficient for performance evaluation. Their use would imply a visual comparison of generated results against published tables. Such a technique limits the number of comparisons that could be made, and would cause doubt as to the credibility of the comparisons. At best, it would provide a good feeling for the quality of each function's performance. Therefore, it is better to automate the process completely, and compare the generated results against another machine generated standard.

The performance evaluation of the functions involves the computation of two important statistics: the maximum relative error (**MRE**), and the root-mean-square relative error (**RE**). Their values are determined through the use of (43) and (44), where $F(x)$ is the test result and $f(x)$ is the comparison value generated by the same extended-precision function call written for the VAX 11/780.

$$\text{MRE} = \max_{x_i} \left| \frac{F(x_i) + f(x_i)}{f(x_i)} \right| \tag{43}$$

$$\text{RE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(\frac{F(x_i) + f(x_i)}{f(x_i)} \right)^2} \tag{44}$$

This method of error checking is an automatic tabular comparison, where the VAX routines serve as the accepted standard. The test routine tests densely packed samples of evenly spaced arguments spread throughout $[-3\pi, 3\pi]$ for floating-point algorithms, and $[-1, 1]$ for fixed-point algorithms. When regenerating arguments within the test modules, it is important not to introduce unnecessary errors. This means that arguments in the VAX should have its lower order bits padded with zeros. The most-significant bits must be equivalent to the number of bits in the 1750A argument, and no extra precision should be introduced.

The method of argument generation just described is recommended by Cody (12: 762), and is the method used at the NASA Lewis Research Center. This method is preferred to a random-number test because it measures the relative error throughout an entire interval. Using densely packed arguments also gives valuable insight to problems of different argument ranges. If the evenly spaced interval is set to a power of two (representable on both machines), and is not less than the least-significant bit of the 1750A argument, then an initial argument can be chosen, such that, zero padding will only have to be performed once. For example, if an initial floating-point argument is -3.1415 and the chosen interval is 2^{-2}, the second argument will be $-3.1415 + 2^{-2}$. Additional padding is not necessary, because "carries" are cascaded forward and do not increase the number of most-significant bits in the next argument. Arguments used in the function calls on both machines must be the same, and must be generated in the same order.

Extra care is needed while reading the 1750A results from disk. Each of the 1750A results are stored in an unformatted file, and must be read into a binary record. This record is moved, bit-by-bit, to a variable of the appropriate type (VAX 11/780 fixed-point or floating-point). The bit-by-bit manipulation is accomplished through the use of JOVIAL specified tables, and prevents conversion errors associated with formatted input.

Before a comparison of the two results (one from the 1750A, and the other from the VAX) can be made, the results generated within the test module must be reduced to the same precision (same number of most-significant bits) as those from the 1750A. The precision reduction gives a rounded result that can be used to determine the **MRE** and **RE**, and will give a meaningful interpretation to the inherited error of the 1750A functions.

V Conclusions and Recommendations

Conclusions

The purpose of this thesis was to develop and to do performance evaluation on a run-time math library developed specifically for MIL-STD-1750A architectures. The library consists of the floating-point implementation of several algebraic functions. Performance evaluation was the major effort of this thesis, but not in the manner intended.

Function approximations are accomplished through the use of either Chebyshef or rational approximations. The two different approximation methods were discussed in chapter two, and are useful in understanding certain design considerations. The values of each polynomial's coefficients were derived by (or were modifications of those derived by) Cody and Waite. (4: 17-84) However, the implementation designs are significantly different from those suggested by Cody and Waite. The primary difference between the implemented designs and those suggested by Cody and Waite, are the methods of argument reduction required of each function.

Performance evaluation turned out to be the major effort, but not because of extensive or elaborate testing of the library functions. Most of the effort involved overcoming the following problems:

1.) There were several compiler bugs in the original 1750A compiler used. Assembly listings had to be reviewed, in order to verify each compilation of the source code.

2.) The use of a simulator for performance evaluation was ruled out because of the limited number instructions that could be simulated, its inability to simulate the use of floating-point data, and the relative speed at which results were calculated. The simulator also lacked a facility for writing results to mass storage. Storage of results on an external device is necessary for input to software test packages.

3.) A new compiler and linker was introduced near the midpoint of the thesis effort, and required a long learning curve in order to use them.

4.) Once a 1750A machine became available, it was determined that all its support software was intended for use with files created by the old compiler and linker.

5.) Rather than use a compiler and linker that had several deficiencies, or write a new loader routine, it was decided to write a support tool that would convert load modules into a format expected by the available loader.

6.) The reformatting program required the use of JOVIAL and its specified table features. It also required the use of FORTRAN routines to perform the I/O of source and target files. The FORTRAN and JOVIAL interfaces did not operate as expected, and the use of COMMON/COMPOOL areas wouldn't work. This required parameter passing between the routines, and the documentation for this type of interface was very inadequate; however, the problems were eventually resolved.

7.) The reformatting tool was written for use on a VAX 11/780. It was assumed that the JOVIAL compiler was free of bugs for a VAX target. However, when the reformat routine was being debugged, it was discovered that JOVIAL table names could be overlayed, but corresponding table items weren't overlayed with them. This problem took a long time to discover, and an additional amount of time to design around.

8.) I/O routines have not been written for the 1750A, and had to be developed. These routines are only capable of writing to a console screen.

9.) Screen output is insufficient for generating the thousands of results that would be needed during testing and evaluation, so another means of capturing the data had to be developed. Due to the lack of time and inexperience in the internal I/O communications techniques of the 1750A hardware, development of a disk I/O routine was not a feasible alternative. It was determined that results could be stored in specific locations of memory, and then an available console routine could be used to write the information to disk. An additional problem was encountered when it was discovered that the record format of the disk files is not in a VAX friendly format, and another routine had to be written to unpack the stored results.

These problems limited the scope of this thesis effort to developing the following: designs; code that is free of syntax errors; the development of command files for compiling, assembling, and linking routines written for the 1750A; tools for formatting load modules that are capable of being loaded into a Sperry 1631 implementation of the MIL-STD-1750A; and tools that unpack test results stored on an RT/11 formatted floppy disk. Generic test algorithms are provided, but are not written in a high-order-language. They provide the basic structure for critical range testing, and a means of evaluating and measuring each functions performance.

Recommendations

The products produced by this thesis effort are at point where design of the intended performance evaluation can begin. All the groundwork has been provided, and should be adequate for someone to continue the effort. Many of the aforementioned problems have been resolved, and support tools and command files are provided to shorten the learning curve that follow-on programmers will have to experience.

The following recommendations should be considered if this effort is continued.

1.) If the effort is limited to the use of JOVIAL, an analysis should be made for determining how to handle exceptions detected at run time. Exceptions include arguments outside legally defined limits.

2.) Since Ada has features for exception handling, all the library functions should also be developed and implemented in Ada.

3.) Another point may be in favor or using Ada is that it also allows the creation of generic packages and subprograms. The generic subprograms define a template, and generic parameters provide the facility for tailoring the template to fit a particular need at translation time. In other words, one subprogram could provide calculations for both fixed-point or floating-point arguments, based on how it is used at compile time. Because a generic package would not be able to take advantage of the specific hardware functions unique to floating-point and fixed point routines, this may result in a degradation of performance.

4.) Initially, it was discussed that all the math library routines should be written in both JOVIAL and Ada with the intent that a comparative evaluation could be done on the two languages. Unfortunately, an Ada compiler targeted to the 1750A is not yet available. When a compiler does become available, it is recommended that a new Ada math library be developed and this comparative evaluation be performed.

5.) The compiler problems, mentioned above, should be corrected, and 1750A architectures and associated support software should be acquired before more time is allocated to the effort.

Appendix A

The following pseudo-operations were used in describing the implementation designs of the different mathematic functions.

ADX(X,N): augments the integer exponent of a floating-point representation of X by N. This scales the argument X by 2^N. For example,

$$ADX(1.0,2) = 4.0$$

FIX(X): returns the fixed-point representation of the floating-point value X . This operation requires explicit conversion in JOVIAL.

FLOAT(X): returns the floating-point representation of the fixed-point argument X. This operation requires explicit conversion in JOVIAL.

ODD(X): determines whether the argument X is odd. For an integer, the least-significant bit is checked directly. For a floating-point number, the integer portion is checked. A description of the floating-point process for this determination is given below.

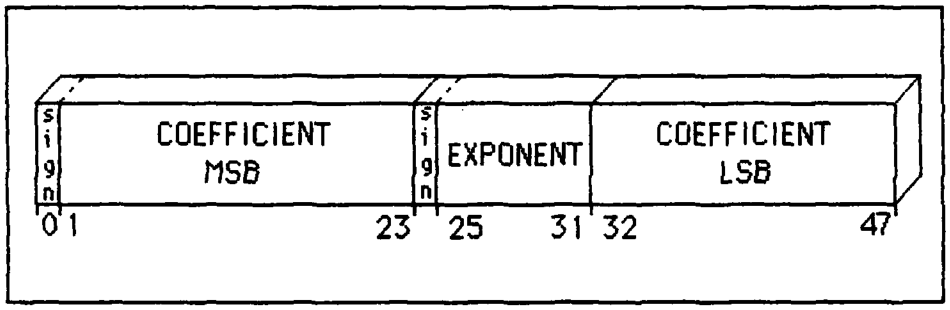


Figure 7 Bit Layout of 1750A Floating-Point Number

To determine whether the integer portion is odd, knowledge of the internal representation of the 1750A floating-point number is necessary. The argument X is a JOVIAL specified table item that makes the components shown in Figure 7 easily accessible. Within this table is an integer item that overlays the exponent field of X. This exponent field is the tool needed to check whether the integer portion is odd or even. Since X has a value of one or greater, and all floating-point values are normalized, the exponent can be used to point to the least significant bit of the integer field. Because X is positive, a one in the least significant bit would indicate the integer portion is odd. A limit on the maximum value of the coefficient has been imposed by the functions that use this routine. This limit prevents the least-significant bit of the integer portion from falling in the exponent or "LSB" area of the floating-point coefficient (see Figure 7).

Since the 1750A architecture requires that all floating-point values be normalized, the most-significant bit is in the first bit position following the sign bit. The decimal-point is assumed to be positioned immediately behind the sign bit, but immediately in front of the most-significant bit. The exponent represents a power of two; therefore, if ε represents the value of the exponent field, the value of the floating-point number is: coefficient * 2^{ε} . Equivalently, it is obvious that the decimal-point floats ε places to the left if ε is negative, or ε places to the right if positive.

Knowledge of how floating-point numbers are stored can be used to determine whether the integer portion of a number is odd. The following example gives an explanation of the process.

> Given the following machine representation of a floating-point number, determine whether its integer portion is odd. In the example below, the decimal-point was inserted only for clarity.
>
> 0.110000000000000000000000**0000000**10000000000000000
>
> Since the sign bit of the exponent is zero, the value of the coefficient is positive. The following two numbers are summed together to determine the value represented by this coefficient:

$1 * 2^{-1} = .5$

$1 * 2^{-1} = .25$

The exponent field is in bold text, and has the value one. Therefore, the value of this floating-point representation is, the coefficient (.75) multiplied by two to-the-power-of the exponent (1), or 1.5.

$.75 * 2^{1} = 1.5$

Another way to compute the result is to shift the decimal-point in a direction as indicated by the exponent. The exponent in this case is +1 , so the decimal-point is shifted one position to the right. The number can then be computed in a similar manner as described above.

This last method demonstrates how to determine whether this example is even or odd. If the decimal-point is shifted 1 position to the right, this number will have 1 integer bit and 38 fractional bits. The integer bits always occupy the left-most position of the number. If the exponent is thought of as a pointer from the left-most side of the number, the least-significant integer bit can be found. The exponent in this example points to bit position one. Since the bit is set to 1, this example's integer value is odd. ∎

INT(X): return the integer portion of the floating-point argument X. The description ODD(X) given above determines the least-significant bit of the integer portion of the floating-point argument. This is used to extract the entire integer portion of the argument (bits 0 through the least significant bit).

Appendix B

```
"*****************************************************************
*                                                                 *
*  DATE:          30 August 1985                                  *
*  VERSION:       1.0                                             *
*  NAME:          ALog                                            *
*  MODULE NUMBER: 1.0                                             *
*  DESCRIPTION:                                                   *
*                 This function is called to compute the natural log of   *
*                 of the argument 'Arg'.  Since                   *
*                      ALog10(X) = ALog(X) * ln(e)                *
*                 ALog is also called by ALog10 to do its computations.   *
*  PASSED VARIABLES:  Arg - an extended precision floating-point variable *
*  RETURNS:           The natural log of arg in extended precision float  *
*  MODULES CALLED:   None                                         *
*  AUTHOR:           Capt. Jennifer Fried                         *
*  HISTORY:          This project was undertaken as a thesis project for  *
*                    partial fulfillment of requirements for an MS degree *
*                    in Information Science from the Air Force Institute  *
*                    of Technology. Sponsoring organization is the ASD    *
*                    Language Control Branch, Wright Patterson AFB,Oh.    *
*                                                                 *
*****************************************************************"

START

   DEF PROC  ALog  RENT( Arg )   F   39;
     BEGIN

     ITEM  Arg          F   39;
     ITEM  Nn           S    7;

     ITEM  Xn           F   39;
     ITEM  Znum         F   39;
     ITEM  Zden         F   39;
     ITEM  Zz           F   39;
     ITEM  Rz           F   39;
     ITEM  Rz2          F   39;
     ITEM  Ww           F   39;
     ITEM  Aw           F   39;
     ITEM  Bw           F   39;

     TABLE Overlays (0) W 3;
        BEGIN
        ITEM  Ff     F  39  POS(0,0);
        ITEM  Fexp   S   7  POS(8,1);
        END

     CONSTANT ITEM  Zero         F   39 =  0.0;
     CONSTANT ITEM  PtFive       F   30 =  0.5;
     CONSTANT ITEM  SqrtPtFive   F   39 =  0.7071067811865;

     CONSTANT ITEM  A0           F   39 =  0.37339168963168E+1;
     CONSTANT ITEM  A1           F   39 = -0.63260866233386E+0;
     CONSTANT ITEM  A2           F   39 =  0.44445515109980E-2;

     CONSTANT ITEM  B0           F   39 =  0.4480700275574E+2;
     CONSTANT ITEM  B1           F   39 = -0.143123543558 9E+2;
     CONSTANT ITEM  B2           F   39 =  0.1000000000000E+1;
```

```
      CONSTANT ITEM  C1          F   39 =  0.6933593750000;
      CONSTANT ITEM  C2          F   39 = -2.1219444005469E-4;

      IF Arg <= Zero;
         ALog = -MAXFLOAT(39);
      ELSE
         BEGIN

         Ff(0)   = Arg;
         Nn      = Fexp(0);
         Fexp(0) = 0;


         IF Ff(0) > SqrtPtFive;
            BEGIN
            Znum = (Ff(0) - PtFive) - PtFive;
            Zden = (Ff(0) * PtFive) + PtFive;
            END
         ELSE
            BEGIN
            Znum = Ff(0) - PtFive;
            Zden = (Znum * PtFive) + PtFive;
            END

         Zz = Znum / Zden;
         Ww = Zz * Zz;

         Aw  = (A2 * Ww + A1) * Ww + A0;
         Bw  = (Ww + B1) * Ww + B0;
         Rz2 = Ww * Aw / Bw;
         Rz  = Zz + Zz * Rz2;

         Xn = (* F 39 *)( Nn );
         ALog = (Xn * C2 + Rz) + Xn * C1;

         END

      RETURN;
   END
TERM
```

```
*****************************************************************************
*                                                                           *
*  DATE:          30 August 1985                                            *
*  VERSION:       1.0                                                       *
*  NAME:          ALog10                                                    *
*  MODULE NUMBER: 1.0                                                       *
*  DESCRIPTION:                                                             *
*                   This subroutine is called to compute the base 10 log   *
*                   of the passed argument. Since                           *
*                            ALog10 = ALog * log(e)                         *
*                   It makes a call to ALog                                 *
*  PASSED VARIABLES:  Arg - an extended-precision floating-point variable  *
*  RETURNS:             The floating-point rep of log(Arg)                  *
*  MODULES CALLED:  ALog                                                    *
*  AUTHOR:          Capt. Jennifer Fried                                    *
*  HISTORY:         This project was undertaken as a thesis project for    *
*                   partial fulfillment of requirements for an MS degree   *
*                   in Information Science from the Air Force Institute    *
*                   of Technology. Sponsoring organization is the ASD      *
*                   Language Control Branch, Wright Patterson AFB,Oh.      *
*                                                                           *
*****************************************************************************

START

   REF PROC  ALog  RENT ( Arg )  F   39;
      BEGIN
      ITEM  Arg  F   39;
      END

   DEF PROC ALog10 RENT (Arg)  F   39;

      BEGIN

      ITEM  Arg                   F   39;

      CONSTANT  ITEM  Log'e'   F   39 = 0.4342944819033;


      ALog10 = ALog(Arg) * Log'e';

      RETURN;
   END
TERM
```

```
"*********************************************************************
*                                                                   *
*  DATE:          30 August 1985                                    *
*  VERSION:       1.0                                               *
*  NAME:          Exp                                               *
*  MODULE NUMBER: 1.0                                               *
*  DESCRIPTION:                                                     *
*                 Returns the extended-precision floating-point value *
*                 for  e**Arg                                       *
*  PASSED VARIABLES:  Arg - an extended precision floating-point variable *
*  RETURNS:       e**x                                              *
*  MODULES CALLED:  none                                            *
*  AUTHOR:          Capt. Jennifer Fried                            *
*  HISTORY:         This project was undertaken as a thesis project for *
*                   partial fulfillment of requirements for an MS degree *
*                   in Information Science from the Air Force Institute *
*                   of Technology. Sponsoring organization is the ASD *
*                   Language Control Branch, Wright Patterson AFB,Oh. *
*                                                                   *
*********************************************************************"

START

  DEF PROC  Exp  RENT (Arg)  F   39;

    BEGIN

    ITEM  ARG                    F   39;
    ITEM  Xx                     F   39;
    ITEM  Xn                     F   39;
    ITEM  Gg                     F   39;
    ITEM  X1                     F   39;
    ITEM  X2                     F   39;
    ITEM  Zz                     F   39;
    ITEM  Pz                     F   39;
    ITEM  Qz                     F   39;

    ITEM  Nn                     S    7;

    TABLE Overlays (0) W 3;
      BEGIN
      ITEM  Ag     F  39  POS(0,0);
      ITEM  Aexp   S   7  POS(8,1);
      END

    CONSTANT ITEM  Xmax          F   39 =  1.70141183459E+38;
    CONSTANT ITEM  Xmin          F   39 =  1.469367938527E-39;

    CONSTANT ITEM  Xbig          F   39 =  88.02969193111;
    CONSTANT ITEM  Xsmall        F   39 = -89.41598629223;
    CONSTANT ITEM  Eps           F   39 =  9.094947017729E-13;

    CONSTANT ITEM  P0            F   39 =  0.24999999999999E+0;
    CONSTANT ITEM  P1            F   39 =  0.5950425497759E-2;

    CONSTANT ITEM  Q0            F   39 =  0.50000000000000E+0;
    CONSTANT ITEM  Q1            F   39 =  0.5356751764522E-1;
    CONSTANT ITEM  Q2            F   39 =  0.2972936368224E-3;
```

```
        CONSTANT ITEM  C1          F   39 =  0.693359375;
        CONSTANT ITEM  C2          F   39 = -2.1219444005470E-4;

        CONSTANT ITEM  Ln2         F   39 =  0.6931471805599;
        CONSTANT ITEM  OneOverLn2  F   39 =  1.4426950408890;
        CONSTANT ITEM  One         F   39 =  1.0;
        CONSTANT ITEM  PtFive      F   39 =  0.5;


        Xx = Arg;

        IF Arg > Xbig;
           Xx = Xbig;

        IF Xx < Xsmall;
           Xx = Xsmall;

        IF (Xx < Eps) AND (Xx > -Eps);
           Exp = One;
        ELSE
           BEGIN

           Nn = (* S, R 7 *)( Xx * OneOverLn2 );
           Xn = (* F 39 *)( Nn );

           X1 = (* F 39 *)((* S 31 *)( Xx ));
           X2 = Xx - X1;
           Gg = ((X1 - Xn * C1) + X2) - Xn * C2;

           Zz    = Gg * Gg;
           Pz    = (P1 * Zz + P0) * Gg;
           Qz    = (Q2 * Zz + Q1) * Zz + Q0;
           Rg(0) = PtFive + Gg * Pz / (Qz - Pz);

           Rexp(0) = Rexp(0) + Nn + (* S 7 *)( 1 );

           Exp = Rg(0);

           END

        RETURN;
     END
  TERM
```

```
"********************************************************************
*                                                                   *
*  DATE:          30 August 1985                                    *
*  VERSION:       1.0                                               *
*  NAME:          Sqrt                                              *
*  MODULE NUMBER: 1.0                                               *
*  DESCRIPTION:                                                     *
*                 Approximates the square root of th argument 'Arg' *
*  PASSED VARIABLES:  Arg - an extended precision floating-point variable *
*  RETURNS:       An extended precision float representation of the sqrt *
*                 of 'Arg'                                          *
*  MODULES CALLED:  none                                            *
*  AUTHOR:          Capt. Jennifer Fried                            *
*  HISTORY:         This project was undertaken as a thesis project for *
*                   partial fulfillment of requirements for an MS degree *
*                   in Information Science from the Air Force Institute *
*                   of Technology. Sponsoring organization is the ASD *
*                   Language Control Branch, Wright Patterson AFB,Oh. *
*                                                                   *
********************************************************************"

START

   DEF PROC  Sqrt  RENT(Xx)  F 39;
      BEGIN

      TABLE OverLays (0) W 6;
         BEGIN
         ITEM  Ff    F  39  POS(0,0);
         ITEM  Fexp  S   7  POS(8,1);
         ITEM  Yy    F  39  POS(0,3);
         ITEM  Yexp  S   7  POS(8,4);
         END

      ITEM  Xx    F  39;
      ITEM  Mm    S   7;
      ITEM  Ix    S   8;

      ITEM  Nn    S  15;
      ITEM  Nbit  B  16;
      OVERLAY Nn : Nbit;

      CONSTANT  ITEM  SqrtOneHalf  F  39 = 0.7071067811865;
      CONSTANT  ITEM  C1           F  39 = 0.41731;
      CONSTANT  ITEM  C2           F  39 = 0.59016;
      CONSTANT  ITEM  One          F  39 = 1.0;
      CONSTANT  ITEM  Zero         F  39 = 0.0;
      CONSTANT  ITEM  OneInt       S   7 = 1;

      Ff(0) = Xx;

      IF (Ff(0) = Zero) OR (Ff(0) = One);
         Sqrt = Ff(0);
      ELSE
         BEGIN

         IF Ff(0) < Zero;
            Ff(0) = -Ff(0);
            Nn = Fexp(0);
            Fexp(0) = 0;

            Yy(0) = C1 + C2 * Ff(0);

            FOR  Ix : 1 BY 1 WHILE Ix <= 3;
               BEGIN
               Yy(0) = Yy(0) + Ff(0) / Yy(0);
               Yexp(0) = Yexp(0) - OneInt;
               END

            IF BIT(Nbit,15,1) = 1B'1';
               BEGIN
               Yy(0) = Yy(0) * SqrtOneHalf;
               Nn = Nn + 1;
               END

            Mm = Nn / 2;
            Yexp(0) = Yexp(0) + Mm;
            Sqrt = Yy(0);
            END;
         RETURN;
      END
   TERM
```

```
"***************************************************************************
*                                                                          *
*  DATE:           19 JULY 1985                                            *
*  VERSION:        1.0                                                     *
*  NAME:           MathLib                                                 *
*  MODULE NUMBER: 1.0                                                      *
*  DESCRIPTION:                                                            *
*         This compool is required by any JOVIAL program that needs to    *
*         reference any of the math functions written for floating-point  *
*         or fixed-point computations                                     *
*  PASSED VARIABLES:  N/A                                                  *
*  RETURNS:           N/A                                                  *
*  MODULES CALLED:    N/A                                                  *
*  AUTHOR:            Capt. Steven A. Hotchkiss and                        *
*                     Capt Jennifer Fried                                  *
*  HISTORY:           This project was undertaken as a thesis project for  *
*          .          partial fulfillment of requirements for an MS degree *
*                     in Information Science from the Air Force Institute  *
*                     of Technology. Sponsoring organization is the ASD    *
*                     Language Control Branch, Wright Patterson AFB,Oh.    *
*                                                                          *
***************************************************************************"

START

  COMPOOL MathLib;

    REF  PROC  Exp    Rent(Arg) F     39;
      BEGIN
      ITEM  Arg  F    39;
      END

    REF  PROC  ALog   Rent(Arg) F     39;
      BEGIN
      ITEM  Arg  F    39;
      END

    REF  PROC  ALog10 Rent(Arg) F     39;
      BEGIN
      ITEM  Arg  F    39;
      END

    REF  PROC  Sqrt   RENT(Arg)  F     39;
      BEGIN
      ITEM  Arg  F    39;
      END

    REF  PROC  Sin    RENT(Xx)  A   1,30;
      BEGIN
      ITEM  Xx   A  1,30;
      END

    REF  PROC  Cos    RENT(Xx)  A   1,30;
      BEGIN
      ITEM  Xx   A  1,30;
      END
```

```
REF  PROC  Tan    RENT(Xx)  A  12,18;
   BEGIN
   ITEM  Xx   A  1,30;
   END

REF  PROC  Cot    RENT(Xx)  A  12,18;
   BEGIN
   ITEM  Xx   A  1,30;
   END

REF  PROC  ASin   RENT(Xx)  A   1,30;
   BEGIN
   ITEM  Xx   A  1,30;
   END

REF  PROC  ACos   RENT(Xx)  A   1,30;
   BEGIN
   ITEM  Xx   A  1,30;
   END

REF  PROC  ATan   RENT(Xx)  A   1,30;
   BEGIN
   ITEM  Xx   A  1,30;
   END

REF  PROC  Sinf   RENT(Xx)  F      39;
   BEGIN
   ITEM  Xx   F    39;
   END

REF  PROC  Cosf   RENT(Xx)  F      39;
   BEGIN
   ITEM  Xx   F    39;
   END

REF  PROC  Tanf   RENT(Xx)  F      39;
   BEGIN
   ITEM  Xx   F    39;
   END

REF  PROC  Cotf   RENT(Xx)  F      39;
   BEGIN
   ITEM  Xx   F    39;
   END

REF  PROC  ASinf  RENT(Xx)  F      39;
   BEGIN
   ITEM  Xx   F    39;
   END

REF  PROC  ACosf  RENT(Xx)  F      39;
   BEGIN
   ITEM  Xx   F    39;
   END
```

```
      REF  PROC  ATanf  RENT(Xx)  F      39;
         BEGIN
         ITEM  Xx   F    39;
         END

TERM
```

```
"*********************************************************************
*                                                                    *
*  DATE:              29 August 1985                                 *
*  VERSION:           1.0                                            *
*  NAME:              IoRefs                                         *
*  MODULE NUMBER:     1.0                                            *
*  DESCRIPTION:                                                      *
*       This Compool is necessary to reference routines that were    *
*       necessary for testing and performance evaluation of all math *
*       functions developed for the 1750.                            *
*  PASSED VARIABLES:  N/A                                            *
*  RETURNS:           N/A                                            *
*  MODULES CALLED:    N/A                                            *
*  AUTHOR:          Capt. Steven A. Hotchkiss  and                   *
*                   Capt. Jennifer Fried                             *
*  HISTORY:         This project was undertaken as a thesis project for *
*                   partial fulfillment of requirements for an MS degree *
*                   in Information Science from the Air Force Institute *
*                   of Technology. Sponsoring organization is the ASD  *
*                   Language Control Branch, Wright Patterson AFB,Oh.  *
*                                                                    *
*********************************************************************"

  START

    COMPOOL IoRefs;

    "
    ' The following ITEMs are required to print a carriage return and
    ' line feed on a terminal connected to a MIL-STD-1750 computer
    "

    DEF ITEM  Carriage  STATIC  U  16 = 2573;
    DEF ITEM  CRLF      STATIC  C   2;
    OVERLAY  Carriage: CRLF;



    "
    ' The following referenced subroutine is written in 1750 Assembly language
    ' and is used to print character strings only. Noncharacter types will
    ' have to be converted before calling this routine. The following DEFINE is
    ' recommended for all routines calling ObcSim:
    '
    '        DEFINE WRITE'STRING(A)  ''Printc(WORDSIZE(!A),LOC(!A))'';
    '
    ' An example of a typical call follows:
    '
    '       ITEM Example  C  2;
    '              :
    '              :
    '              :
    '       WRITE'STRING(Example);
    "


  REF PROC  Printc RENT(Length, Message);
```

```
      BEGIN
      ITEM  Length   U  (BITSINWORD-1);
      ITEM  Message  P;
      END


   "
   ' The following referenced routine is necessary for routines wishing
   ' to convert floating-point values to a character string
   "

   REF PROC  FltToChar (Arg)  C 20;
      BEGIN
      ITEM  Arg      F  39;
      END


   "
   ' The following referenced routine is necessary for routines wishing
   ' to convert fixed-point values to a character string. The variable
   ' IntOverlay must be overlayed on top of a fixed-point variable and
   ' BitsInFrac is an integer value indicating the number of fractional
   ' bits in the fixed-point value.
   "

   REF PROC FixToChar (IntOverlay, BitsInFrac) C  20;
      BEGIN
      ITEM  IntOverlay        S  31;
      ITEM  BitsInFrac        U   8;
      END

TERM
```

```
   "*****************************************************************
   *                                                               *
   *  DATE:              29 August 1985                            *
   *  VERSION:           1.0                                       *
   *  NAME:              FixToChar                                 *
   *  MODULE NUMBER:     1.0                                       *
   *  DESCRIPTION:                                                 *
   *        This routine is used to convert fixed-point values into*
   *        character representation. This routine was necessary for*
   *        testing and performance evaluation of math routines developed *
   *        for the 1750                                           *
   *  PASSED VARIABLES:  IntOverlay - An Integer Variable Overlayed on top *
   *                              of a fixed-point value           *
   *                     BitsInFrac - the number of fractional bits of *
   *                              the fixed-point argument         *
   *  RETURNS:           a 20 character representation of the argument *
   *  MODULES CALLED:    FltToChar                                 *
   *  AUTHOR:         Capt. Steven A. Hotchkiss  and               *
   *                  Capt. Jennifer Fried                         *
   *  HISTORY:        This project was undertaken as a thesis project for *
   *                  partial fulfillment of requirements for an MS degree *
   *                  in Information Science from the Air Force Institute *
   *                  of Technology. Sponsoring organization is the ASD *
   *                  Language Control Branch, Wright Patterson AFB,Oh. *
   *                                                               *
   *****************************************************************"

START

   REF PROC  FltToChar (Arg)  C 20;
      BEGIN
      ITEM  Arg   F 39;
      END


            "***************   FixToChar Procedure   ***************"


   DEF PROC  FixToChar  (IntOverlay, BitsInFrac)  C  20;

      BEGIN

      ITEM  IntOverlay     S  31;
      ITEM  BitsInFrac     U   8;

      TABLE Overlays (0) W 3;
         BEGIN
         ITEM Arg       F 39 POS(0,0);
         ITEM ArgExp    S  7 POS(8,1);
         END

      Arg(0)    = (* F 39 *)( IntOverlay );
      ArgExp(0) = ArgExp(0) - (* S 7 *)( BitsInFrac );

      FixToChar = FltToChar(Arg(0))
      RETURN;
   END
TERM
```

```
"*****************************************************************
*                                                               *
*  DATE:              29 August 1985                            *
*  VERSION:           1.0                                       *
*  NAME:              FltToChar                                 *
*  MODULE NUMBER:     1.0                                       *
*  DESCRIPTION:                                                 *
*         This routine is used to convert floating-point values into *
*         character representation. This routine was necessary for   *
*         testing and performance evaluation of math routines developed *
*         for the 1750                                          *
*  PASSED VARIABLES:  Arg - the value to be converted           *
*  RETURNS:           a 20 character representation of the argument *
*  MODULES CALLED:    none                                      *
*  AUTHOR:         Capt. Steven A. Hotchkiss  and               *
*                  Capt. Jennifer Fried                         *
*  HISTORY:        This project was undertaken as a thesis project for *
*                  partial fulfillment of requirements for an MS degree *
*                  in Information Science from the Air Force Institute *
*                  of Technology. Sponsoring organization is the ASD *
*                  Language Control Branch, Wright Patterson AFB,Oh. *
*                                                               *
*****************************************************************"

START

  DEF  PROC  FltToChar (Arg)  C  20;

     BEGIN

     DEFINE  Yes          "1B'1'";
     DEFINE  No           "1B'0'";

     ITEM  Arg          F  39;
     ITEM  Fraction     F  39;
     ITEM  Temp         F  39;
     ITEM  Result       C  20;

     ITEM  Ix           U   8;
     ITEM  Iy           U   8;
     ITEM  ExpCnt       U   8;

     ITEM  NegExp       B;

     ITEM  CharVal      U   8;
     ITEM  CharRep      C   1;
     OVERLAY  CharRep: CharVal;

     ITEM  ZeroRep      STATIC   C    1 = '0';
     ITEM  ZeroVal      STATIC   U    8;
     OVERLAY  ZeroRep: ZeroVal;

     CONSTANT  ITEM  Zero       F  39 =  0.0;
     CONSTANT  ITEM  One        F  39 = 1.0;
     CONSTANT  ITEM  TenFloat   F  39 = 10.0;
     CONSTANT  ITEM  PtFive     F  39 =  0.5;
     CONSTANT  ITEM  PtOne      F  39 =  0.1;
```

```
        Result = ' 0.0000000000000E+00';

        IF Arg < Zero;
           BEGIN
           Fraction          =  -Arg;
           BYTE(Result,0,1) = '-';
           END
        ELSE
           Fraction = Arg;

        IF Fraction < PtOne;
           NegExp = Yes;
        ELSE
           NegExp = No;

        ExpCnt = 0;
        WHILE (Fraction > One);
           BEGIN
           ExpCnt   = ExpCnt + 1;
           Fraction = Fraction / TenFloat;
           END

        IF (NegExp = Yes) AND ( Fraction <> Zero);
           BEGIN
           BYTE(Result,17,1) = '-';

           WHILE (Fraction < PtOne);
              BEGIN
              ExpCnt   = ExpCnt + 1;
              Fraction = Fraction * TenFloat;
              END

           END

        Iy = 0;
        WHILE ((Fraction <> Zero) AND (Iy < 13));
           BEGIN
           Temp = Fraction * TenFloat;
           IF Iy = 12;
              Temp   = Temp + PtFive;
           CharVal  = (* U 8 *)( Temp );
           Fraction = Temp - (* F 39 *)( CharVal );
           CharVal  = CharVal + ZeroVal;
           BYTE(Result,Iy+3,1) = CharRep;
           Iy = Iy + 1;
           END

        CharVal           = (* U 8 *)(ExpCnt MOD 10) + ZeroVal;
        BYTE(Result,19,1) = CharRep;
        CharVal           = (* U 8 *)(ExpCnt / 10)   + ZeroVal;
        BYTE(Result,18,1) = CharRep;

        FltToChar = Result;

        RETURN;
     END
TERM
```

```
               TITLE    HOL(PRINTC)
               MODULE   PRINTC
*
*
*
*  DATE:          4 September 1985
*  VERSION:       1.0
*  NAME:          Printc
*  MODULE NUMBER: 1.0
*  DESCRIPTION:
*                 This module is called to print a character string onto
*                 a console that is connected to a Mil-Std-1750 computer
*  PASSED VARIABLES:
*                 LENGTH_3  - this variable contains a count of the number
*                             characters to print
*                 MESSAGE_3 - this is a location pointer for the string to be
*                             printed
*  RETURNS:
*                 prints messages on user console
*  MODULES CALLED:
*  AUTHOR:        Capt. Steven A. Hotchkiss and
*                 Capt. Jennifer Fried
*  HISTORY:       This project was undertaken as a thesis project for
*                 partial fulfillment of requirements for an MS degree
*                 in Information Science from the Air Force Institute
*                 of Technology. Sponsoring organization is the ASD
*                 Language Control Branch, Wright Patterson AFB,Oh.
*
*
*
* $  4-SEP-85/16:09:29 $
               PRINTOFF                        . DO NOT LIST METAS
*
* START OF META DEFINITIONS
*
DATAS          META     3             . REPEATED PRESET META
LF(0)          EQU      $
_              LOOP     2,1,NUM(GF)-1
               VOID     GF(_,1),NORMA,__DATAS
               GOTO     TEST
NORMA          LABEL
               DATA     GF(_)
TEST           LOOPTEST
               MEND
__DATAS        META     3
__             LOOP     1,1,GF(_,1)
               DATA     GF(_)
               LOOPTEST
               MEND
               LENGTH   25,9999
*
SECTION        META     0             . CSECT META
_              LOOP     2,1,31
SC(DL)N*(_)    CSECT
               LOOPTEST
               MEND
```

```
*
* GENERATE REG EQUATES META
*
REG             META
XNC             LOOP      0,1,15
NC(R)           EQU       XNC
                LOOPTEST
                MEND
*
* END OF META DEFINITIONS
*
*
* BASE REG EQUATES
*
B12             EQU       12
B13             EQU       13
B14             EQU       14
B15             EQU       15
*
* CONDITION CODE EQUATES
*
_NOP            EQU       0
_LT             EQU       1
_EQ             EQU       2
_LE             EQU       3
_GT             EQU       4
_NE             EQU       5
_GE             EQU       6
_CY             EQU       8
_CLT            EQU       9
_CEQ            EQU       10
_CLE            EQU       11
_CGT            EQU       12
_CNE            EQU       13
_CGE            EQU       14
_UN             EQU       15
*
* END OF EQUATES
*
                REG
                SECTION
                PRINT
                DEFINE    PRINTC
PSSDATA$        EQU       3
PSSCONS$        EQU       4
PSSCODE$        EQU       2
* NO REF DATA DECLARATIONS
* NO BYREF/TYPE/ABSOLUTE DECLARATIONS
* LOCAL AUTOMATIC DATA  *** SIZE IN WORDS -- 2 DECIMAL : 2  HEX ***
*    LOCAL AUTOMATIC DATA FOR PROC      PRINTC
* STACK FRAME  *** SIZE IN WORDS -- 2 DECIMAL : 2  HEX ***
BK__003EF       EQU       HEX(0)                . SIZE =     2
LENGTH_3        EQU       HEX(0)                . SIZE =     1
MESSAGE_3       EQU       HEX(1)                . SIZE =     1
* END OF LOCAL AUTOMATIC DECLARATIONS
* PSECT $DATA IS EMPTY
```

```
*
*
*                 R2 = NUMBER OF CHARACTERS IN STRING
*                 R3 = LOCATION OF CHARACTER STRING
*
*
PSSCODE           ORIGIN    HEX(0)
PRINTC            EQU       $
                  ORIGIN    HEX(0002)
*
                  AISP      R2,1          . ADJUST CHARACTER COUNT
                  SRA       R2,1          . 1ST TWO COMMANDS EQUIVALENT TO
*                                         .      R2 = ROUND(R2/2)
                  BLE       LB__0002      . BRANCH OUT IF ILLEGAL CHAR COUNT
OUTPUT            EQU       $
                  XIO       R5,RCS        . READ CONSOLE STATUS
                  TBR       1,R5          . CHECK STATUS BIT 1
                  BEZ       OUTPUT        . IF OFF, LOOP BACK UNTIL CONSOLE READY
                  L         R5,0,R3       . GET NEXT TWO CHARACTERS OF MESSAGE
                  XIO       R5,CO         . PRINT BOTH CHARACTERS
                  AISP      R3,1          . POINT TO NEXT TWO CHARACTERS
                  SOJ       R2,OUTPUT     . DECREMENT LOOP COUNT, GO BACK IF MORE
*
LB__0002          EQU       $
                  AISP      R15,2
                  POPM      R2,R3
                  URS       R15
                  ORIGIN    HEX(0000)
                  PSHM      R2,R3
                  SISP      R15,2
                  ORIGIN    HEX(0013)
                  END
```

Appendix C

```
"*****************************************************************************
*                                                                           *
*  DATE:              10 October 1985                                       *
*  VERSION:           1.0                                                   *
*  NAME:              RefMat                                                *
*  MODULE NUMBER:     1                                                     *
*  DESCRIPTION:                                                             *
*                     This routine is used to convert ITS LINK files into   *
*                     a format that can be loaded into the SPERRY 1631      *
*                     computer (1750A architecture). The ITS files are      *
*                     '.SO' files and must be in the 80 column record format *
*                     described in the EMAD ITS Load Module ICD (CDRL #1005  *
*                     contract #F33657-83-C-0244). Use of the command file   *
*                     LINK1750.COM to link all compiled modules will insure  *
*                     that these records are of the right format. The format *
*                     of the SPERRY loader records are defined in Appendix B *
*                     of its programmer reference manual. The bytes of all   *
*                     binary data fields must be swapped (i.e. the high      *
*                     order bits of a word are swapped with the low order 8) *
*                     The only type ITS records converted are binary and     *
*                     end record types. It also ignores all protection       *
*                     indicators, and can not handle expanded memory jobs.   *
*                     When all object files are copied into a single object  *
*                     for linking by the ITS LINKER, the main procedure must *
*                     be copied into the file first!!!!!!!! Otherwise, this  *
*                     application will have no way of determining the point  *
*                     that execution is to begin. The 'end' record created   *
*                     by the ITS linker contains the lowest addrress of the  *
*                     load module, and this application assumes that the     *
*                     routine begins at that point. The ITS file contains    *
*                     datafields that are in HEX character representation,   *
*                     and the SPERRY 1631 expects binary data fields;        *
*                     therefore the ITS data must also be converted to       *
*                     binary                                                 *
*  PASSED VARIABLES:   N/A                                                   *
*  RETURNS:            N/A                                                   *
*  MODULES CALLED:     GetHdr                                                *
*                      Readf                                                 *
*                      Printf                                                *
*                      ClnUp                                                 *
*                      IntFil                                                *
*                      WriteRcd                                              *
*  AUTHOR:            Capt. Steven A. Hotchkiss and                          *
*                     Capt. Jennifer Fried                                   *
*  HISTORY:           This project was undertaken as a thesis project for    *
*                     partial fulfillment of requirements for an MS degree   *
*                     in Information Science from the Air Force Institute    *
*                     of Technology. Sponsoring organization is the ASD      *
*                     Language Control Branch, Wright Patterson AFB,Oh.      *
*                                                                           *
*****************************************************************************"


START

   !COMPOOL ('IoData');
   !COMPOOL ('IoCalls');
   !COMPOOL ('RfMtCpl');
```

```
PROGRAM RefMat;

   BEGIN
   ChkSum = 4B'0000';
   FirstPass = True;
   LdPt = -1;
   Eof = False;
   Buff = 0;
   BufPtr(0) = 1;
   BufPtr(1) = 1;

   " Initialize IO Files "
   IntFil;

   " Get Header Info for Loader File "
   GetHdr;

   WHILE NOT Eof;
      BEGIN

      " Read the first 80 column record "
      Readf(:ItsRcd,Eof);

      " Put Loader Info into Contiguous Memory Locations "
      WdsInRcd = Cntl(0) - Ascii0;
      AddrC(0) = Addr(0);
      Wd1C(0)  = Wd1(0);
      Wd2C(0)  = Wd2(0);
      Wd3C(0)  = Wd3(0);
      Wd4C(0)  = Wd4(0);
      Wd5C(0)  = Wd5(0);
      Wd6C(0)  = Wd6(0);
      Wd7C(0)  = Wd7(0);

      " Initialize the Output Buffers "
      FOR Ix: 1 BY 1 WHILE Ix<33;
         CharToBin(Ix) = 0;

      FOR Ix: 0 BY 1 WHILE Ix<63;
         OutBuff(Ix) = 0;


      " Convert Char To Bin and Pack it "
      FOR Ix: 1 BY 1 WHILE Ix<=32;
         BEGIN
         IF (Ascii0<=CharToBin(Ix)) AND (CharToBin(Ix)<=Ascii9);
            CharToBin(Ix) = CharToBin(Ix) - Ascii0;
         ELSE
            IF (AsciiA<=CharToBin(Ix)) AND (CharToBin(Ix)<=AsciiF);
               CharToBin(Ix) = CharToBin(Ix) - AsciiA + 10;
         HalfByte(Ix) = Nibbles(Ix);
         END


      IF Typ(0) = ' ';
         BEGIN  "This is a binary record"

         IF BufPtr(Buff) + WdsInRcd <=61 AND LdPT = Laddr(0);
            BEGIN  " Old Record and still room for more data fields "
```

```
          " Flip Flop the position of each byte of a 1750A word "
          FOR Ix: 0 BY 1 WHILE Ix<WdsInRcd;
             BEGIN
             BufByte0(BufPtr(Buff)+Ix) = FieldL(Ix+1);
             BufByte1(BufPtr(Buff)+Ix) = FieldH(Ix+1);
             END

          " Point to where info from next ITS 80 column record "
          " is to be placed into this loader record           "
          BufPtr(Buff) = BufPtr(Buff) + WdsInRcd;

          " Update load point so the next ITS record can be checked to "
          " see if it belongs in this loader record                    "
          LdPt = LdPt + WdsInRcd;

          IF BufPtr(Buff) = 61;
             BEGIN  " Loader Record is full and needs to be written "

             WdsInBuffer = 60;
             RcdTypI = AsciiB;
             WriteRcd;
             END

          END

       ELSE
          BEGIN  " Old record and not enough room -- or new record "

          IF LdPt = Laddr(0);
             BEGIN  " Same loader record, but not enough room for all "
                    " data fields in ITS record                       "

             " Swap Bytes of words going into loader record "
             FOR Ix: 0 BY 1 WHILE BufPtr(Buff)+Ix < 61;
                BEGIN
                BufByte0(BufPtr(Buff)+Ix) = FieldL(Ix+1);
                BufByte1(BufPtr(Buff)+Ix) = FieldH(Ix+1);
                LdPt = LdPt + 1;
                END

             " write the full record out "
             WdsInBuffer = 60;
             RcdTypI = AsciiB;
             WriteRcd;

             " Set the load point for this new loader record "
             LdAd(0) = LdPt;

           " Swap bytes of the other ITS data fields and place them into "
           " record. If the next ITS record doesn't have the load point  "
           " computed here, it should be the first entries for another   "
           " loader record                                               "
             FOR Iy: Ix BY 1 WHILE Iy < WdsInRcd;
                BEGIN
                BufByte0(BufPtr(Buff)+Iy) = FieldL(Iy+1);
                BufByte1(BufPtr(Buff)+Iy) = FieldH(Iy+1);
                LdPt = LdPt + 1;
                END
```

```
            END  "Same record not enogh room"

         ELSE

            BEGIN  " this is the start of a new loader record "

            IF NOT FirstPass;
               BEGIN

               IF BufPtr(Buff) <> 1;
                  BEGIN  "the last record didn't get filled up, so it "
                         " hasn't been written yet. The routine      "
                         " WriteRcd sets BufPtr to 1 before exit  "

                  RcdTypl = AsciiB;
                  WdsInBuffer = BufPtr(Buff) -1;
                  WriteRcd;
                  END
               END  " end not first pass "

            FirstPass = False;

            " Set the load point for this loader record "
            LdRd(0) = Laddr(0);

            " Swap bytes of ITS data fields going into loader record "
            FOR Ix: 0 BY 1 WHILE Ix < WdsInRcd;
               BEGIN
               BufByte0(Ix+1) = FieldL(Ix+1);
               BufByte1(Ix+1) = FieldH(Ix+1);
               END

            BufPtr(Buff) = WdsInRcd + 1;
            LdPt = Laddr(0) + WdsInRcd;

            END  " end new record "

         END  "end of old record not enough room -- or new record "

      END  "end of this is a binary record"

   ELSE

      BEGIN  "this is an execution address record "
      IF Typ(0) = 'E';
         BEGIN
         RcdTypl = 8261;   "blank E"
         OutBuff(0) = Laddr(0);
         OutBuff(1) = 30;  " ascii record seperator "
         WriteRcd;
         END

      END  "end execution address record "

   END  "end while loop "

   " Write end of file loader record "
   RcdTypl = 8262;  " blank F "
```

```
        OutBuFF(0) = 30;  " ascii record seperator "

        " Clean up Files used "
        ClnUp;

     END
  TERM
```

```
"*****************************************************************************
*                                                                           *
*  DATE:             10 October 1985                                        *
*  VERSION:          1.0                                                    *
*  NAME:             WriteRcd                                               *
*  MODULE NUMBER:    7                                                      *
*  DESCRIPTION:                                                             *
*                   This routine is called by RefMat to do IO stuff that    *
*                   needs to be done throughout the main procedure. Three   *
*                   types of SPERRY 1631 loader records are written:        *
*                   Binary ,Execution, and End of file. If the record type  *
*                   is a binary record, this routine computes a checksum    *
*                   for it and tcks it on to the end of the record. Then    *
*                   the record type is written out followed by the binary   *
*                   record. If the record type is an execution record or    *
*                   an end of file record, the record type is written out   *
*                   followed by the record. The variable 'Buff' is a        *
*                   global variable that points to the record to be         *
*                   written.                                                *
*  PASSED VARIABLES:  None                                                  *
*  RETURNS:           Nothing                                               *
*  MODULES CALLED:    Printf -- a FORTRAN IO routine                        *
*  AUTHOR:          Capt. Steven A. Hotchkiss and                           *
*                   Capt. Jennifer Fried                                    *
*  HISTORY:         This project was undertaken as a thesis project for     *
*                   partial fulfillment of requirements for an MS degree    *
*                   in Information Science from the Air Force Institute     *
*                   of Technology. Sponsoring organization is the ASD       *
*                   Language Control Branch, Wright Patterson AFB,Oh.       *
*                                                                           *
*****************************************************************************"


START

   !COMPOOL('RfMtCpl');
   !COMPOOL ('IoData');

   REF PROC Printf(RcdTyp,Buffer);
      !LINKAGE FORTRAN;
      BEGIN
      ITEM RcdTyp  S  15;
      ITEM Buffer  C  128;
      END

   DEF PROC WriteRcd;
      BEGIN

      LoopCnt = WdsInBuffer;

      If RcdTypl = AsciiB;
         BEGIN

         ChkSum = 4B'0000';
         OutBuff(0) = LdAd(0);
         ChkSum = ChkSum XOR OutBuffB(0);

         OutBuff(1) = WdsInBuffer;
         ChkSum = ChkSum XOR OutBuffB(1);
```

```
            FOR Ix: 1 BY 1 WHILE Ix <= LoopCnt;
               BEGIN
               OUTBUFF(Ix+1) = BufWd(Ix);
               ChkSum = ChkSum XOR OutBuffB(Ix+1);
               END

            OutBuffB(Ix+1) = ChkSum;

            Printf(RcdTypI,OutFld);

            END
         ELSE

            Printf(RcdTypI,OutFld);


         BufPtr(Buff) = 1;
         Buff = ABS(1-Buff);


         RETURN;
      END
   TERM
```

```
"***********************************************************************
*                                                                      *
*  DATE:              10 October 1985                                  *
*  VERSION:           1.0                                              *
*  NAME:              IoCalls                                          *
*  MODULE NUMBER:     9                                                *
*  DESCRIPTION:                                                        *
*                     This compool is required for RefMat to reference its *
*                     associated FORTRAN IO routines                   *
*  PASSED VARIABLES:  N/A                                              *
*  RETURNS:           N/A                                              *
*  MODULES CALLED:    N/A                                              *
*  AUTHOR:            Capt. Steven A. Hotchkiss and                    *
*                     Capt. Jennifer Fried                             *
*  HISTORY:           This project was undertaken as a thesis project for *
*                     partial fulfillment of requirements for an MS degree *
*                     in Information Science from the Air Force Institute *
*                     of Technology. Sponsoring organization is the ASD *
*                     Language Control Branch, Wright Patterson AFB,Oh. *
*                                                                      *
***********************************************************************"

START

   COMPOOL IoCalls;

      REF PROC  WriteRcd;
         BEGIN
         END

      REF PROC GetHdr;
         !LINKAGE FORTRAN;
         BEGIN
         END

      REF PROC Readf(:ItsRcd,Eof);
         !LINKAGE FORTRAN;
         BEGIN
         ITEM  ItsRcd C 80;
         ITEM  Eof    B  1;
         END

      REF PROC Printf(RcdTyp, Buffer);
         !LINKAGE FORTRAN;
         BEGIN
         ITEM RcdTyp  S  15;
         ITEM Buffer  C  126;
         END

      REF PROC ClnUp;
         !LINKAGE FORTRAN;
         BEGIN
         END

      REF PROC IntFil;
         !LINKAGE FORTRAN;
         BEGIN
         END
 TERM
```

```
"*********************************************************************
*                                                                    *
*  DATE:              10 October 1985                                *
*  VERSION:           1.0                                            *
*  NAME:              IoData                                         *
*  MODULE NUMBER:     8                                              *
*  DESCRIPTION:                                                      *
*                    This compool defines all data required for the JOVIAL  *
*                    routine RefMat and its associated FORTRAN 10 routines  *
*  PASSED VARIABLES:  N/A                                            *
*  RETURNS:           N/A                                            *
*  MODULES CALLED:    N/A                                            *
*  AUTHOR:           Capt. Steven A. Hotchkiss and                   *
*                    Capt. Jennifer Fried                            *
*  HISTORY:          This project was undertaken as a thesis project for    *
*                    partial fulfillment of requirements for an MS degree   *
*                    in Information Science from the Air Force Institute    *
*                    of Technology. Sponsoring organization is the ASD      *
*                    Language Control Branch, Wright Patterson AFB,Oh.      *
*                                                                    *
*********************************************************************"

START

   COMPOOL IoData;

      DEF ITEM Infil     C  10;
      DEF ITEM Outfil    C  10;
      DEF ITEM Filnam    C   6;
      DEF ITEM Header    C  80;

      DEF TABLE  ItsTable(0)  W 20;
         BEGIN
         ITEM Addr   C 4   POS(16,00);
         ITEM Typ    C 1   POS(16,01);
         ITEM Cnt    C 1   POS(24,01);
         ITEM Cntl   S 7   POS(24,01);
         ITEM Wd1    C 4   POS(08,03);
         ITEM Wd2    C 4   POS(16,05);
         ITEM Wd3    C 4   POS(24,07);
         ITEM Wd4    C 4   POS(00,10);
         ITEM Wd5    C 4   POS(08,12);
         ITEM Wd6    C 4   POS(16,14);
         ITEM Wd7    C 4   POS(24,16);
         END
      DEF ITEM ItsRcd  C 80;
      OVERLAY ItsRcd: ItsTable;

      DEF TABLE  OutRcd  (0:62)  T 16 W;
         BEGIN
         ITEM  OutBuff     S  15  POS(0,0);
         ITEM  OutBuffB    B  16  POS(0,0);
         END
      DEF ITEM    OutFld C 126;
      OVERLAY  OutRcd: OutFld;


      DEF ITEM Eof        B   1;
      DEF ITEM RcdTypl    S  15;
```

```
          DEF ITEM RcdTyp     C   2;
          OVERLAY RcdTyp1: RcdTyp;

          OVERLAY  Infil, OutFil, Filnam, Header, ItsRcd, OutFld, Eof, RcdTyp;

      TERM
```

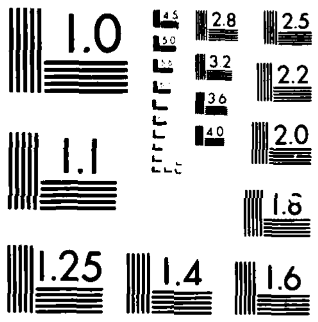
AD-A164 050 DEVELOPMENT AND EVALUATION OF MATH LIBRARY ROUTINES FOR A 1750A AIRBORNE MICROCOMPUTER(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 2/2

UNCLASSIFIED J J FRIED 04 DEC 85 AFIT/GCS/MA/85D-3 F/G 9/2 NL

END

FILMED

DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

```
"*****************************************************************************
*                                                                           *
*  DATE:              10 October 1985                                       *
*  VERSION:           1.0                                                   *
*  NAME:              RfMtCpl                                               *
*  MODULE NUMBER:     10                                                    *
*  DESCRIPTION:                                                             *
*                     This compool contains all the variables and tables    *
*                     that are used to unpack ITS linker records, packs them *
*                     and converts the HEX characters to binary data fields, *
*                     and then places them into a SPERRY 1631 loader record *
*                     format                                                *
*  PASSED VARIABLES:   N/A                                                  *
*  RETURNS:            N/A                                                  *
*  MODULES CALLED:     N/A                                                  *
*  AUTHOR:            Capt. Steven A. Hotchkiss and                         *
*                     Capt. Jennifer Fried                                  *
*  HISTORY:           This project was undertaken as a thesis project for   *
*                     partial fulfillment of requirements for an MS degree  *
*                     in Information Science from the Air Force Institute   *
*                     of Technology. Sponsoring organization is the ASD     *
*                     Language Control Branch, Wright Patterson AFB,Oh.     *
*                                                                           *
*****************************************************************************"

START

   COMPOOL RfMtCpl;

      DEF  ITEM  ChkSum      B  16;
      DEF  ITEM  FirstPass   B   1;

      DEF  ITEM  LdPt        S  15;
      DEF  ITEM  Buff        U   8;
      DEF  ITEM  Ix          U   8;
      DEF  ITEM  Iy          U   8;
      DEF  ITEM  WdsInRcd    S  15;
      DEF  ITEM  LoopCnt     S  15;
      DEF  ITEM  WdsInBuffer S  15;

      DEF  ITEM  Zero    STATIC  C  1 = '0';
      DEF  ITEM  Ascii0  STATIC  S  7;
      OVERLAY Zero: Ascii0;

      DEF  ITEM  Nine    STATIC  C  1 = '9';
      DEF  ITEM  Ascii9  STATIC  S  7;
      OVERLAY Nine: Ascii9;

      DEF  ITEM  AA      STATIC  C  1 = 'A';
      DEF  ITEM  AsciiA  STATIC  S  7;
      OVERLAY AA: AsciiA;

      DEF  ITEM  FF      STATIC  C  1 = 'F';
      DEF  ITEM  AsciiF  STATIC  S  7;
      OVERLAY FF: AsciiF;

      DEF  ITEM  BB      STATIC  C  2 = ' B';
      DEF  ITEM  AsciiB  STATIC  S 15;
      OVERLAY BB: AsciiB;
      DEF TABLE LoadPoint (0);
         BEGIN
         ITEM LdAd  S  15;
         END


      DEF TABLE BufStuf (0:1);
         BEGIN
         ITEM BufPtr     S  7;
         END

      DEF TABLE PackedRcd (0)  W 8;
         BEGIN
         ITEM AddrC  C  4  POS(0,0);
         ITEM Wd1C   C  4  POS(0,1);
         ITEM Wd2C   C  4  POS(0,2);
         ITEM Wd3C   C  4  POS(0,3);
         ITEM Wd4C   C  4  POS(0,4);
         ITEM Wd5C   C  4  POS(0,5);
         ITEM Wd6C   C  4  POS(0,6);
         ITEM Wd7C   C  4  POS(0,7);
         END

      DEF TABLE CharConvert (1:32)  T  8  W;
         BEGIN
         ITEM CharToBin  S  7  POS(0,0);
         ITEM Nibbles    S  3  POS(4,0);
         END

      OVERLAY PackedRcd:  CharConvert;


      DEF TABLE HexBuf (1:32)  T   4  W;
         BEGIN
         ITEM  HalfByte   S  3  POS(0,0);
         END

      DEF TABLE  PakIts (0:7)  T  16  W;
         BEGIN
         ITEM  Laddr   S  15  POS(0,0);
         END

      DEF TABLE BinFields (0:7)  T  16  W;
         BEGIN
         ITEM  Field   S  15 POS(0,0);
         ITEM  FieldH  S   7 POS(0,0);
         ITEM  FieldL  S   7 POS(8,0);
         END

         OVERLAY  HexBuf,Ix: PakIts: BinFields;

       DEF TABLE DatFields  (0) W 1;
         BEGIN
         ITEM  BufByte0   S  7  POS(0,0);
         ITEM  BufByte1   S  7  POS(8,0);
         ITEM  BufWd      S 15  POS(0,0);
         END
   TERM
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
C  DATE:               10 October 1985                                C
C  VERSION:            1.0                                            C
C  NAME:               IntFil                                         C
C  MODULE NUMBER:      6                                              C
C  DESCRIPTION:                                                       C
C                    This routine is called by the JOVIAL routine called C
C                    RefMat. Its purpose is to prompt the user for the   C
C                    name of a file that was created by an ITS link,     C
C                    prompt the user for the name of a file that the     C
C                    reformatted ITS file is to be written to, and then  C
C                    opens both files. The input file must be a ".SO" file C
C                    and the output file is a ".DAT" file.               C
C  PASSED VARIABLES:   None                                           C
C  RETURNS:            Nothing                                        C
C  GLOBAL VARIABLES:   All variables used are global, and have been   C
C                      defined in the common (COMPOOL) called IoData  C
C  MODULES CALLED:     None                                           C
C  AUTHOR:           Capt. Steven A. Hotchkiss and                    C
C                    Capt. Jennifer Fried                             C
C  HISTORY:          This project was undertaken as a thesis project for C
C                    partial fulfillment of requirements for an MS degree C
C                    in Information Science from the Air Force Institute C
C                    of Technology. Sponsoring organization is the ASD   C
C                    Language Control Branch, Wright Patterson AFB,Oh.   C
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


      Subroutine IntFil

        IMPLICIT INTEGER (A-Z)

        CHARACTER*6     Filnam
        CHARACTER*10    Infil,   Outfil

        WRITE(*,*)' Enter File Name (Max 6 Characters) '
        READ(*,10)Filnam
 10     FORMAT(A6)

        I = INDEX(Filnam,'.') - 1
        IF (I.LE.0) THEN
           I = INDEX(Filnam,' ') - 1
           IF (I.LE.0) THEN
              I = 6
           ENDIF
        ENDIF

        Infil  = Filnam(1:I)//'.SO'
        Outfil = Filnam(1:I)//'.DAT'
        WRITE(*,*)'Input File = ',Infil,'Output file = ',Outfil

        OPEN(UNIT = 2, NAME = Infil,  TYPE = 'OLD', FORM = 'FORMATTED')
        OPEN(UNIT = 3, NAME = OutFil, TYPE = 'NEW',
     1FORM = 'UNFORMATTED')

      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                        C
C  DATE:               10 October 1985                                   C
C  VERSION:            1.0                                               C
C  NAME:               GetHdr                                            C
C  MODULE NUMBER:      2                                                 C
C  DESCRIPTION:                                                          C
C                   This routine performs IO for a JOVIAL routine called C
C                   RefMat. It requests a user to input a one line       C
C                   header that will be placed in a loader file.         C
C  PASSED VARIABLES:   None                                              C
C  RETURNS:            Nothing                                           C
C  GLOBAL VARIABLES:   All variables used are global, and are defined in C
C                      the common (COMPOOL) called IoData                C
C  MODULES CALLED:     None                                              C
C  AUTHOR:          Capt. Steven A. Hotchkiss and                        C
C                   Capt. Jennifer Fried                                 C
C  HISTORY:         This project was undertaken as a thesis project for  C
C                   partial fulfillment of requirements for an MS degree C
C                   in Information Science from the Air Force Institute  C
C                   of Technology. Sponsoring organization is the ASD    C
C                   Language Control Branch, Wright Patterson AFB,Oh.    C
C                                                                        C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


      Subroutine GetHdr

         IMPLICIT INTEGER (A-Z)

         INTEGER*2       Spacer1
        CHARACTER*80      Header
        CHARACTER*1       RS
        DATA RS/30/

        Spacer1 = 0
        WRITE(*,*)' Enter Optional 1 Line Header Text '
        Read(*,10)Header
 10     FORMAT(A80)
        WRITE(*,*)Header

        WRITE(3)' D'//Header//RS
        DO 20 I=42,64
           WRITE(3)Spacer1
 20     CONTINUE

      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
C  DATE:               10 October 1985                                C
C  VERSION:            1.0                                            C
C  NAME:               Printf                                         C
C  MODULE NUMBER:      4                                              C
C  DESCRIPTION:                                                       C
C                    This routine is called by the JOVIAL routine called  C
C                    RefMat. It  is used to write SPERRY loader records out C
C                    to a ".DAT" file. The name of the fil being written is C
C                    stored in the global variable Outfil which was set in  C
C                    the routine called IntFil                            C
C  PASSED VARIABLES:  None                                            C
C  RETURNS:            Nothing                                        C
C  GLOBAL VARIABLES:  All variables used are global, and are defined in  C
C                      the common (COMPOOL) called IoData             C
C  MODULES CALLED:   None                                             C
C  AUTHOR:           Capt. Steven A. Hotchkiss and                    C
C                    Capt. Jennifer Fried                             C
C  HISTORY:          This project was undertaken as a thesis project for  C
C                    partial fulfillment of requirements for an MS degree C
C                    in Information Science from the Air Force Institute  C
C                    of Technology. Sponsoring organization is the ASD    C
C                    Language Control Branch, Wright Patterson AFB,Oh.    C
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


      Subroutine Printf(RcdTyp, OutFld)

         IMPLICIT INTEGER (A-Z)
         CHARACTER*2     RcdTyp
         INTEGER*2       OutFld(1:63)

         WRITE(3)RcdTyp,(OutFld(I), I = 1,63)
         WRITE(*,*)'Write next record'

      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                   C
C  DATE:                10 October 1985                             C
C  VERSION:             1.0                                         C
C  NAME:                Readf                                       C
C  MODULE NUMBER:       3                                           C
C  DESCRIPTION:                                                     C
C                     This routine is called by the JOVIAL routine called  C
C                     RefMat. Its is used to read 80 column records created C
C                     by the ITS linker. The name of the fil being read is  C
C                     stored in the global variable Infil which was set in  C
C                     the routine called IntFil. This file is a ".80" file  C
C  PASSED VARIABLES:    None                                        C
C  RETURNS:             Nothing                                     C
C  GLOBAL VARIABLES:    All variables used are global, and are defined in  C
C                       the common (COMPOOL) called IoData          C
C  MODULES CALLED:      None                                        C
C  AUTHOR:            Capt. Steven A. Hotchkiss and                 C
C                     Capt. Jennifer Fried                          C
C  HISTORY:           This project was undertaken as a thesis project for  C
C                     partial fulfillment of requirements for an MS degree C
C                     in Information Science from the Air Force Institute  C
C                     of Technology. Sponsoring organization is the ASD    C
C                     Language Control Branch, Wright Patterson AFB,Oh.    C
C                                                                   C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


      Subroutine Readf(ItsRcd, Eof)

         IMPLICIT INTEGER (A-Z)

         CHARACTER*80   ItsRcd
         LOGICAL*4      Eof

         Eof = .FALSE.

         READ(2,10, END = 20) ItsRcd
10       FORMAT(A80)
         WRITE(*,*)ItsRcd
         GOTO 30
20       Eof = .TRUE.
30       CONTINUE
      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                     C
C  DATE:                 10 October 1985                              C
C  VERSION:              1.0                                          C
C  NAME:                 ClnUp                                        C
C  MODULE NUMBER:        5                                            C
C  DESCRIPTION:                                                       C
C                      This routine is called by the routine RefMat to close C
C                      the files it used for IO                      C
C  PASSED VARIABLES:     None                                         C
C  RETURNS:              Nothing                                      C
C  MODULES CALLED:       None                                         C
C  GLOBAL VARIABLES:     Allvariables used are global, and are defined in  C
C                        the common (COMPOOL) called IoData           C
C  AUTHOR:             Capt. Steven A. Hotchkiss and                  C
C                      Capt. Jennifer Fried                           C
C  HISTORY:            This project was undertaken as a thesis project for  C
C                      partial fulfillment of requirements for an MS degree C
C                      in Information Science from the Air Force Institute  C
C                      of Technology. Sponsoring organization is the ASD    C
C                      Language Control Branch, Wright Patterson AFB,Oh.    C
C                                                                     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


      Subroutine ClnUp

         IMPLICIT INTEGER (A-Z)

         CLOSE(UNIT = 2)
         CLOSE(UNIT = 3)

      END
```

Appendix D

```
$ ! ASM1750 -- Assemble a 1750 source module
$ !
$ ! @ASM1750    file
$ !
$ !             file = input source name of module   file.SI
$ !
$ ! Create Assembler input file UI that designates Mil-Std-1750A as the target
$ ! rather than the alternate 1750A target
$ !
$ CREATE 'P1'.UI
.ASSEMBLE   TARGET=M1750A
$ TY 'P1'.UI
$ !
$ ASSIGN 'P1'.UI                UI  ! UPDATE INPUT COMMANDS FILE (INPUT)
$ ASSIGN 'P1'.SI                SI  ! 1750A ASSEMBLY SOURCE FILE (INPUT)
$ ASSIGN 'P1'.OBJ               OO  ! OBJECT OUTPUT
$ ASSIGN 'P1'.SO                SO  ! SYMBOLIC OUTPUT
$ ASSIGN 'P1'.LO                LO  ! LISTING OUTPUT
$ ASSIGN LIB_JOVIAL_1750A       OI  ! LIBRARY INPUT
$ !
$ SET VERIFY
$ M1750A
$ !
$ DEASSIGN SI
$ DEASSIGN UI
$ DEASSIGN OO
$ DEASSIGN SO
$ DEASSIGN LO
$ DEASSIGN OI
$ !
$ DELETE 'P1'.UI;*
$ SET NOVERIFY
$ ! JOV1750 -- JOVIAL COMPILE FOR MIL-STD-1750A TARGET
$ !
$ ! JOV1750        file [.filetype]  [options]
$ !
$ ! e.g.,         @JOV1750 TEST1     .SRC   /SYNTAX_ONLY/STATISTICS
$ !               @JOV1750 TEST2  /MACHINE_CODE/CROSS
$ !
$ ! Note: If the filetype is JOV, options may be typed as 2nd parameter.
$ !       If a filetype is supplied, it must be preceded by a "." as shown.
$ !
$ ! Resulting object module has type .OBJ
$ !
$ SET VERIFY
$ JOVIAL 'P1''P2'/TARGET=1750A/NOINFO/CROSS/ASSEM'P3'
$ ! LINK1750 -- Link one or more 1750A target object modules.
$ !
$ ! @LINK1750    file
$ !
$ !              file = object file (containing one or more object modules)
$ !                     create object file by first deleting all .obj files for
$ !                     COMPOOLs that don't contain any DEFs. Then use the
$ !                     following commands to create the object file
$ !
$ ! COPY *.OBJ file.O
$ ! RENAME file.O file.OBJ
$ !
$ ! OBJ files created by the compiler and the assembler can be copied to the
$ ! same OBJ file, but the VAX will give an incompatible files warning. Ignore
$ ! the warning, the copy is made anyway
$ !
$ ! Create Linker input file "UI"
$ !
$ SET VERIFY
$ !
$ CREATE 'P1'.UI
.LINK DATA,LIST,DEBUG,INPUTS
ALLOCATE LOCATION=1000 MODULES .
LINKEND
$ !
$ ASSIGN 'P1'.UI             UI              ! LINKER CONTROL (INPUT)
$ ASSIGN 'P1'.OBJ            OO              ! OBJECT MODULE(S) (INPUT)
$ ASSIGN 'P1'.SO             SO              ! LOAD MODULE (OUTPUT)
$ ASSIGN 'P1'.LO             LO              ! LINKER LIST FILE (OUTPUT)
$ ASSIGN LIB_JOVIAL_1750A    OI              ! LIBRARY OBJECT FILE (INPUT)
$ !
$ ITSLINK                    ! RUN 1750A Linker...reads logic device UI
$                            ! Output on SO and LO
$ DEASSIGN UI
$ DEASSIGN OO
$ DEASSIGN SO
$ DEASSIGN LO
$ DEASSIGN OI
$ !
$ DELETE 'P1'.UI;*
$ SET NOVERIFY
$ !
$ !     LOGIN.COM       This command procedure is invoked with each login.
$ !                     and may be changed to tailor your environment.
$ !
$ !     Set standard aliases.  Note that several UNIX-like aliases are set up.
$ !
$       SET NOVERIFY
$       SET PROTECTION=(SYSTEM:R,OWNER:RWED,GROUP:RW,WORLD:RWE)/DEFAULT
$SYMBOLS:
$       BQ      :== SHOW QUEUE/BATCH
$       CD      :== SET DEFAULT
$       DS      :== DIRECTORY /SIZE
$       E       :== EDIT
$       HOME    :== SET DEFAULT DSK$ADOL:[ADOL.HOTCHSA]
$       LO      :== @LOGOUT.COM
$       LS      :== DIRECTORY
$       PQ      :== SHOW QUEUE SYS$PRINT
$       PS      :== SHOW PROCESS        ! Like UNIX ps command
$       PWD     :== SHOW DEFAULT        ! Like UNIX pwd command
$       R       :== RUN
$       SD      :== SHOW DEVICES
$       SG      :== SHOW SYMBOLS /GLOBAL /ALL
$
$       ST      :== SHOW TERMINAL
$       WHO     :== SHOW USERS          ! Like UNIX who command
$       SHQ     :== SHOW QUEUE SLAM$QUEUE/ALL
$       S80     :== SET TERMINAL/WIDTH=80
$       S132    :== SET TERMINAL/WIDTH=132
$       JOV1750 :== @JOV1750
$       LINK1750:== @LINK1750
$       SIM1750 :== @SIM1750
$       ASM1750 :== @ASM1750
$       UNLOCK  :== @UNPROTECT
$ !
$ !     End user defined keyins.
$ !
$ ! DEFINE JOVIAL LIBRARY FOR AUTOMATIC SEARCHING FOR VAX TARGET
$ !
$ ! ASSIGN JOVLIBV:JOVLIBV.OLB LNK$LIBRARY
$ !
$ ! The following defines the 1750A support tools pseudo-commands:
$ !
$ LINK50A :== LINKITS
$ RAIDX :== $TOOLS:RAID
$ !
$ ! END LOGIN.COM
$ !
$FINISH:
$       EXIT
```

Appendix E

```
--**************************************************************************

with TEXT_IO;
use TEXT_IO;
with TCHEBYSHEF_PACKAGE;
use TCHEBYSHEF_PACKAGE;

procedure TCHEBYSHEF_ECONOMIZATION is

--**************************************************************************
--  This procedure is the main driver for the Tchebyshef economization
--    of a polynomial.
--**************************************************************************

  ECONOMIZED_POLYNOMIAL: FLOAT_VECTOR (0..MAX_DEGREE) :=
                         (0..MAX_DEGREE => 0.0);
    --The is the resulting economized coefficients to the polynomial
  SUM: FLOAT_VECTOR (0..MAX_DEGREE) := (0..MAX_DEGREE => 0.0);
    --This value is a temporary work area for the sum of the columns
    --  of the work matrix
  WORK_MATRIX: FLOAT_MATRIX (0..MAX_DEGREE, 0..MAX_DEGREE) :=
          (0..MAX_DEGREE => (0..MAX_DEGREE => 0.0));
    --Temporary work area for forming the economized coefficients
```

```
--********************************************************************

procedure DISPLAY_VECTOR (PRINT_VECTOR: in VECTOR) is
--The sole purpose of this routine is to display an integer vector

  package INT_IO is new INTEGER_IO (integer);
  use INT_IO;

begin  --Display Vector.
  for I in 0..DEGREE_OF_POLYNOMIAL loop
    put (I);
    put (" ");
    put (PRINT_VECTOR (I));
    new_line;
  end loop;
end DISPLAY_VECTOR;
```

```
-------------------------------------------------------------------------

procedure DISPLAY_FLOAT_VECTOR (PRINT_VECTOR: in FLOAT_VECTOR) is
--The sole purpose of this routine is to display a floating point vector

  package INT_IO is new INTEGER_IO (integer);
  use INT_IO;
  package FLT_IO is new FLOAT_IO (float);
  use FLT_IO;

begin  --Display Float Vector.
  for I in 0..DEGREE_OF_POLYNOMIAL loop
    put (I);
    put (" ");
    put (PRINT_VECTOR (I));
    new_line;
  end loop;
end DISPLAY_FLOAT_VECTOR;
```

```
--**************************************************************************

procedure DISPLAY_MATRIX (PRINT_MATRIX: in MATRIX) is
--The sole purpose of this routine is to display an integer matrix

  package INT_IO is new INTEGER_IO (integer);
  use INT_IO;
  package FLT_IO is new FLOAT_IO (float);
  use FLT_IO;

begin  --Display Matrix.
  for I in 0..DEGREE_OF_POLYNOMIAL loop
    put (I);
    put (" ");
    for J in 0..DEGREE_OF_POLYNOMIAL loop
      put (PRINT_MATRIX (I,J));
      put (" ");
    end loop;
    new_line;
  end loop;
end DISPLAY_MATRIX;
```

```
--*************************************************************

procedure DISPLAY_FLOAT_MATRIX (PRINT_MATRIX: in FLOAT_MATRIX) is
--The sole purpose of this routine is to display a floating point matrix

  package INT_IO is new INTEGER_IO (integer);
  use INT_IO;
  package FLT_IO is new FLOAT_IO (float);
  use FLT_IO;

begin  --Display Float Matrix.
  for I in 0..DEGREE_OF_POLYNOMIAL loop
    put (I);
    put (" ");
    for J in 0..DEGREE_OF_POLYNOMIAL loop
      put (PRINT_MATRIX (I,J));
      put (" ");
    end loop;
    new_line;
  end loop;
end DISPLAY_FLOAT_MATRIX;
```

```
--*****************************************************************

begin  --Tchebyshef Economization.

  INPUT_COEFFICIENTS;
  put ("Input Coefficients");
  new_line;
  DISPLAY_FLOAT_VECTOR (COEFFICIENTS);

  COMPUTE_TCHEBYSHEF_POLYNOMIAL;
  new_line;
  put ("Tchebyshef Polynomial");
  new_line;
  DISPLAY_MATRIX (TCHEBYSHEF_POLYNOMIALS);

  COMPUTE_POWERS_OF_TCHEBYSHEF;
  new_line;
  put ("Powers of Tchebyshef");
  new_line;
  DISPLAY_FLOAT_MATRIX (POWERS_OF_TCHEBYSHEF);

  --Generate the work matrix used in the final calculations of the economized
  --  polynomial.  Again the matrix is lower triangular.
  for I in 0..DEGREE_OF_POLYNOMIAL loop
    for J in 0..I loop
      WORK_MATRIX (I,J) := float(MULTIPLIER (I)) * POWERS_OF_TCHEBYSHEF (I,J)
           * COEFFICIENTS (I);
    end loop;
  end loop;

  --Accumu  'e the sum of the work matrix columns
  for I in 0..DEGREE_OF_POLYNOMIAL loop
    for J in 0..DEGREE_OF_POLYNOMIAL loop
      SUM(J) := SUM(J) + (WORK_MATRIX(I,J));
    end loop;
  end loop;

  --Perform the final additions and multiplications to form the result.
  for I in 0..(DEGREE_OF_POLYNOMIAL - 1) loop
    for J in 0..I loop
      ECONOMIZED_POLYNOMIAL (J) := ECONOMIZED_POLYNOMIAL (J) +
           float(TCHEBYSHEF_POLYNOMIALS (I,J)) * SUM (I);
    end loop;
  end loop;

  new_line;
  put ("Economized Polynomial");
  new_line;
  DISPLAY_FLOAT_VECTOR (ECONOMIZED_POLYNOMIAL);

end TCHEBYSHEF_ECONOMIZATION;

--*****************************************************************
```

```
--****************************************************************************
--   DATE:  December 1, 1985
--   VERSION:  1.0
--   NAMES:  TCHEBYSHEF_PACKAGE
--           STRING_TO_INT
--           INPUT_COEFFICIENTS
--           COMPUTE_TCHEBYSHEF_POLYNOMIAL
--           COMPUTE_POWERS_OF_TCHEBYSHEF
--           DISPLAY_VECTOR
--           DISPLAY_MATRIX
--   DESCRIPTIONS:  Provided with each routine.
--   PASSED VARIABLES:  The input to this system is the description of
--                      the polynomial to be economized.
--   RETURNS:  The result of processing is the coefficients of the
--             economized polynomial.
--   CALLING MODULES:  TCHEBYSHEF_ECONOMIZATION
--   AUTHOR:  Capt Jennifer Fried and
--            Capt Steven Hotchkiss
--   HISTORY:  Original version, Dec 1, 1985
--****************************************************************************

with TEXT_IO; use TEXT_IO;

package TCHEBYSHEF_PACKAGE is

--****************************************************************************
--  This package receives the coefficients of a polynomial that is to be
--  economized, computes its Tchebysheg polynomial, and the powers of
--  Tchebyshef matrix.
--****************************************************************************

--Unconstrained type declarations
  type MATRIX is array (integer range <>, integer range <>) of integer;
    --Matrix of integer values, used to contain the Tchebyshef polynomials
  type FLOAT_MATRIX is array (integer range <>, integer range <>) of float;
    --Matrix of floating point values, used to contain the powers of
    --  Tchebyshef
  type VECTOR is array (integer range <>) of integer;
    --Vector of integer values, used to contain the multiplier of the matrix
  type FLOAT_VECTOR is array (integer range <>) of float;
    --Vector of floating point values, used to contain the coefficients of
    --  the polynomial

--Variable declarations
  MAX_DIGIT: integer := 19;
    --The maximum number of digits permitted in a number is nine.
    --  This value represents the maximum input string length for two numbers
    --  and a slash, "/".
  MAX_DEGREE: integer := 9;
    --The maximum value of the largest exponent of the polynomial
  DEGREE_OF_POLYNOMIAL: integer := 0;
    --The actual value of the largest exponent as input by the user
  COEFFICIENTS: FLOAT_VECTOR (0..MAX_DEGREE) := (0..MAX_DEGREE => 0.0);
    --Contains a coefficient for each degree of the polynomial that was
    --  specified by the user
  MULTIPLIER: VECTOR (0..MAX_DEGREE) := (0..MAX_DEGREE => 0);
    --This vector contains the reciprocal of the values contained on
    --  the diagonal of the Tchebyshef polynomial matrix.
    --  Used in generating the economized polynomial.
  TCHEBYSHEF_POLYNOMIALS: MATRIX (0..MAX_DEGREE, 0..MAX_DEGREE) :=
          (0..MAX_DEGREE => (0..MAX_DEGREE => 0));
    --The matrix obtained when using the Tchebyshef formula.
  POWERS_OF_TCHEBYSHEF: FLOAT_MATRIX (0..MAX_DEGREE, 0..MAX_DEGREE) :=
          (0..MAX_DEGREE => (0..MAX_DEGREE => 0.0));
    --The matrix formed when applying the second step of the economization
    --  algorithm

  function STRING_TO_INT (S: string) return integer;
    --This function is used to convert the input coefficient string into an
    --  integer value that equates to the numerator and the denominator.

--These procedures perform the functions specified by this package
  procedure INPUT_COEFFICIENTS;
    --Get the input coefficients for the polynomial
  procedure COMPUTE_TCHEBYSHEF_POLYNOMIAL;
    --Generate the Tchebyshef polynomial matrix
  procedure COMPUTE_POWERS_OF_TCHEBYSHEF;
    --Generate the powers of Tchebyshef matrix

end TCHEBYSHEF_PACKAGE;
```

```
--*********************************************************************

package body TCHEBYSHEF_PACKAGE is

--*********************************************************************

function STRING_TO_INT (S: string) return integer is
  --String to integer equivalent conversion.

    CHAR            : character;  --Individual number in each
                                  --  placeholder of the input string.
    DIGIT           : integer;  --Individual number in each placeholder
                                --  of the output integer.
    MULTIPLIER      : integer := 1;  --Tens value of the integer
                                     --  pointer.
    FINAL_RESULT    : integer := 0;  --Output integer being generated.
    POSITION        : integer := S'last;  --Pointer into input string
                                          --  (moves right to left).

begin  --String to integer conversion.

    --Starting from the end of the input string, process each
    --  successive character until all characters have been converted.
    while POSITION >= S'first loop

      --Get one character digit from the input string.
      CHAR := S(POSITION);

      --If this is a valid character digit representation, convert the
      --  character into its numeric representation, and multiply it by
      --  its tens value.
      if CHAR in '0'..'9' then
        DIGIT := character'pos(CHAR) - character'pos('0');
        DIGIT := DIGIT * MULTIPLIER;

        --If the final value will be the most negative number,
        --  designate it as the most negative number and stop
        --  processing. The reason this is done is to adjust for the
        --  problem that the absolute value of the most negative number
        --  is 1 digit larger than the most positive number and will
        --  result in an out-of-bound condition.
        if integer'last = (FINAL_RESULT - 1) + DIGIT then
          FINAL_RESULT := integer'first;
          POSITION := S'first;
        else

          --Otherwise, this is not the most negative number. Thus,
          --  add the current digit to the rest of those found, and
          --  increment the tens value to the next larger number.
          FINAL_RESULT := FINAL_RESULT + DIGIT;
          MULTIPLIER := MULTIPLIER * 10;
        end if;

      --If the original input was negative, then negate the results.
      elsif CHAR = '-' then
        FINAL_RESULT := -FINAL_RESULT;
      end if;

      --Adjust the pointer into the input string to point to the next
      --  character to the left.
      POSITION := POSITION - 1;
    end loop;

   --Conversion finished, return the generated integer.
   return FINAL_RESULT;
end STRING_TO_INT;
```

```
--***************************************************************************

procedure INPUT_COEFFICIENTS is
  --This procedure obtains the information about the input polynomial and
  --  converts the coefficients into floating point format

  package INT_IO is new INTEGER_IO(integer);
  use INT_IO;

  POWERS: integer := 3;
    --Indicates whether all powers, only the even, or only the odd powers
    --  are present in the input polynomial.  Originally set to out of
    --  bounds condition to verify proper input.
  STEPS: integer := 2;
    --Increment value for entering the coefficients of the polynomial
  INITIAL: integer := 0;
    --Starting value for the value of the exponent
  COUNTER: integer;
    --Loop counter through the input string
  NUMERATOR: integer;
    --Numerator of the coefficient
  DENOMINATOR: integer;
    --Denominator of the coefficient
  CONVERT_STRING: string (1..MAX_DIGIT);
    --String representation of the coefficient
  LAST_DIGIT: integer;
    --Actual length of the input string

begin  --Input Coefficients.

  --Obtain the value of the largest exponent of the polynomial.
  --  It must be between 2 and 9.
  while DEGREE_OF_POLYNOMIAL < 2 or DEGREE_OF_POLYNOMIAL > MAX_DEGREE loop
    put ("Enter the degree of polynomial desired. (Minimum is 2):  ");
    get (DEGREE_OF_POLYNOMIAL);
    new_line;
  end loop;

  --Obtain an indicator for the type of the polynomial's exponents
  while POWERS < 0 or POWERS > 2 loop
    put ("Enter 0 for coefficients for ALL powers of X");
    new_line;
    put ("Enter 1 for coefficients for ODD powers of X");
    new_line;
    put ("Enter 2 for coefficients for EVEN powers of X");
    new_line;
    get (POWERS);
  end loop;

  --Set the initial and incremental values for obtaining the polynomial
  --  coefficients.  Saves time.
  if POWERS = 0 then
    STEPS := 1;
  elsif POWERS = 1 then
    INITIAL := 1;
  end if;
```

```
--Obtain the coefficients for each element of the polynomial
put ("Enter the coefficients of the series being expanded by");
new_line;
put ("  entering a fraction, i.e. -2/3 or +2/3 or 2/3");
new_line;
put ("Coefficient for X** ");
new_line;
--Loop through all elements
while INITIAL <= DEGREE_OF_POLYNOMIAL loop
  put (INITIAL);
  put (" = ");
  get_line (CONVERT_STRING,LAST_DIGIT);
  new_line;
  COUNTER := 1;

  --Step through the input string looking for the "/" which separates
  --  the numerator from the denominator.  If one does not exist, or it
  --  appears in either the first or the last position in the string,
  --  then the coefficient must be reentered.
  while COUNTER <= LAST_DIGIT loop
    if (CONVERT_STRING (COUNTER) = '/') and
      (COUNTER /= CONVERT_STRING'first and
      COUNTER /= LAST_DIGIT) then

      declare
        --Once the "/" has been located and is in a proper location
        --  obtain the numerator string and the denominator string.
        NUMERATOR_STRING: string renames
          CONVERT_STRING (CONVERT_STRING'first..(COUNTER - 1));
        DENOMINATOR_STRING: string renames
          CONVERT_STRING ((COUNTER + 1)..LAST_DIGIT);

      begin  --Block
        --Convert the two strings into integers
        NUMERATOR := STRING_TO_INT (NUMERATOR_STRING);
        DENOMINATOR := STRING_TO_INT (DENOMINATOR_STRING);

        --If the denominator is a valid value, then generate the floating
        --  point value for the coefficient
        if DENOMINATOR /= 0 then
          COEFFICIENTS (INITIAL) := float(NUMERATOR) / float(DENOMINATOR);
          --Increment to the next element in the polynomial.
          INITIAL := INITIAL + STEPS;
        end if;

        --Indicate that this coefficient has been found and converted
        COUNTER := LAST_DIGIT;
      end;  --Block

    end if;
    --Point to the next character in the input string
    COUNTER := COUNTER + 1;
  end loop;
end loop;
end INPUT_COEFFICIENTS;
```

```
--*****************************************************************************

procedure COMPUTE_TCHEBYSHEF_POLYNOMIAL is
  --Generate the matrix of the Tchebyshef polynomial.  The procedure uses
  --  values of the matrix elements that have already been found.
  --  The algorithm is recursive in that respect.

begin  --Compute Tchebyshef Polynomial.

  --The first two elements must be initialized to allow the following
  --  passes to use them.
  TCHEBYSHEF_POLYNOMIALS (0,0) := 1;
  TCHEBYSHEF_POLYNOMIALS (1,1) := 1;

  --Loop through the lower triangular portion of the matrix
  --  and calculate the Tchebyshef polynomial values.

  for I in 2..MAX_DEGREE loop
    for J in 0..I - 2 loop
      TCHEBYSHEF_POLYNOMIALS (I,J) :=
          TCHEBYSHEF_POLYNOMIALS (I,J) - TCHEBYSHEF_POLYNOMIALS (I - 2,J);
    end loop;

    for J in 0..I - 1 loop
      TCHEBYSHEF_POLYNOMIALS (I,J + 1) :=
          TCHEBYSHEF_POLYNOMIALS (I,J + 1) +
          (2 * TCHEBYSHEF_POLYNOMIALS (I - 1,J));
    end loop;
  end loop;
end COMPUTE_TCHEBYSHEF_POLYNOMIAL;
```

```
--*****************************************************************

procedure COMPUTE_POWERS_OF_TCHEBYSHEF is
--Compute the matrix for the powers of Tchebyshef

  COEFFICIENT_LIST: FLOAT_VECTOR (0..MAX_DEGREE) :=
                    (0..MAX_DEGREE => 0.0);
  INDEX: integer := DEGREE_OF_POLYNOMIAL;
  STEP: integer;
  POINTER: integer;

begin  --Compute Powers of Tchebyshef.
  while INDEX >= 0 loop
    MULTIPLIER (INDEX) := 1 / TCHEBYSHEF_POLYNOMIALS (INDEX,INDEX);
    STEP := INDEX;
    while STEP >= 0 loop
      COEFFICIENT_LIST (STEP) := float(TCHEBYSHEF_POLYNOMIALS (INDEX,STEP));
      STEP := STEP - 1;
    end loop;
    POWERS_OF_TCHEBYSHEF (INDEX,INDEX) := 1.0;
    STEP := INDEX - 2;
    while STEP >= 0 loop
      POWERS_OF_TCHEBYSHEF (INDEX,STEP) :=
          - (COEFFICIENT_LIST (STEP))
          / float(TCHEBYSHEF_POLYNOMIALS (STEP,STEP));
      POINTER := STEP;
      while POINTER >= 0 loop
        COEFFICIENT_LIST (POINTER) :=
            COEFFICIENT_LIST (POINTER) + POWERS_OF_TCHEBYSHEF (INDEX,STEP)
            * float(TCHEBYSHEF_POLYNOMIALS (STEP,POINTER));
        POINTER := POINTER - 2;
      end loop;
      STEP := STEP - 2;
    end loop;
    INDEX := INDEX - 1;
  end loop;
end COMPUTE_POWERS_OF_TCHEBYSHEF;

--*****************************************************************

end TCHEBYSHEF_PACKAGE;

--*****************************************************************
```

```
--
-- Date:              28 November 1985
-- Version:           1.0
-- Name:              Approx_Driver
-- Module Number:     1.0
-- Description:       This routine loops until a user is done approximating
--                    whichever function he desires
-- Passed Variables:  None
-- Returns:           None
-- Globals Used:      Choice
-- Modules Called:    MENU
-- Author:            Capt. Steven A. Hotchkiss and
--                    Capt. Jennifer Fried
-- History:           Developed as a thesis and ADA project


with GLOBAL_DATABASE;    use GLOBAL_DATABASE;
with APPROXIMATORS;      use APPROXIMATORS;
with TEXT_IO;            use TEXT_IO;
procedure APPROX_DRIVER is

   NUM: integer := 0;
   DEN: integer := 1;
   CHOICE, KEY  : character;
   QUIT         : character := '7';

   package INT_IO is new INTEGER_IO(INTEGER);
   use INT_IO;

   package FLT_IO is new FLOAT_IO(LONG_FLOAT);
   use FLT_IO;

   begin

   set_page_length(24);

   -- initialize data points
   COMPUTE_TCHEBYSHEV;

   -- let the user approximate as many functions as needed
```

```
  while (CHOICE /= QUIT) loop

     -- select function to approximate
     -- by giving users a menu of options
     MENU(CHOICE);

     -- use the built functions to make a more accurate approximation
     COMPUTE_PADE_APPROXIMATIONS;
     COMPUTE_CMK;

     if CHOICE /= QUIT then
        for I in 0..M loop
           if  C(NUM,I) /= 0.0  or  C(Den,I) /= 0.0 then
              put("a**");
              put(I);
              put(" ==> ");
              put(C(NUM,I));
              put("   b**");
              put(I);
              put(" ==> ");
              put(C(DEN,I));
              new_line;
           end if;
        end loop;
     end if;

     put("Hit any key to continue");
     get(KEY);
     new_line;


  end loop;

end APPROX_DRIVER;
```

```
--
-- Date:               28 November 1985
-- Version:            1.0
-- Name:               GLOBAL_DATABASE
-- Module Number:      2.0
-- Description:        Contains all global variables
-- Passed Variables:   N/A
-- Returns:            N/A
-- Globals Used:       ALL
-- Modules Called:     N/A
-- Author:             Capt. Steven A. Hotchkiss and
--                     Capt. Jennifer Fried
-- History:            Completed for Thesis and ADA project


package GLOBAL_DATABASE is


   type LONG_FLOAT is digits 9;
   type VECTOR is array(integer range 0..25) of LONG_FLOAT;
   type MATRIX is array(integer range 0..25, integer range 0..25) of
        LONG_FLOAT;
   type PADE_MATRIX is array (integer range 0..25, integer range 0..1,
                        integer range 0..25) of LONG_FLOAT;


   T: MATR!X;                     -- Matrix containing the coefficients of
                                  -- different powers of Tchebyshev polynomials
   A: PADE_MATRIX;                -- Used to contain the series of PADE approx
                                  -- A(S,N or D,C)
                                  -- S  is the series number
                                  -- N or D      N - 0 for the numerator
                                  --             D - 1 for the denominator
                                  -- C - coefficient for a power of X for the
                                  --    particular series' numerator or
                                  --    denominator
   D: VECTOR;                     -- Error values of PADE approximations
   M: integer;                    -- Power of the numerator polynomial
   K: integer;                    -- Power of the denominator polynomial
   N: integer;                    -- Power of the initial power series
   MACLAURIN: VECTOR;             -- Contains the coefficients for the
                                  --   different powers of "X" for the power

                                  --   series expansion of a function
   COEFFICIENT: string(1..33);    -- Used to contain user entered coefficients
                                  --   a power series expansion
   EPS: LONG_FLOAT;               -- Convergent epsilon
   C:   MATRIX;                   -- Final rational approximation

end GLOBAL_DATABASE;
```

```
--
-- DATE:                28 November 1985
-- Version:             1.0
-- Name:                COMMON_PROCS
-- Module Number:       3.0
-- Description:         This package contains procedures that are invoked
--                      throughout the system
-- Passed Variables:    N/A
-- Returns:             N/A
-- Globals Used:        N/A
-- Modules Called:      N/A
-- Author:              Capt. Steven A. Hotchkiss and
--                      Capt. Jennifer Fried
-- History:             Developed as a thesis and ADA project


with GLOBAL_DATABASE;   use GLOBAL_DATABASE;
package COMMON_PROCS is

   procedure POWER_PROMPT(NUM, DEN: out integer; Epsilon: out LONG_FLOAT);

   procedure GET_COEFFICIENTS(STRUCTURE:in character; POWER: in integer);

   function  PRODUCT(FROM, TO, BY: integer) return LONG_FLOAT;

   function  FACTORIAL (NUMBER: integer) return LONG_FLOAT;

end COMMON_PROCS;
```

```
with TEXT_IO;            use TEXT_IO;
package body COMMON_PROCS is

   package INT_IO is new INTEGER_IO(integer);
   use INT_IO;

   package FLT_IO is new FLOAT_IO(LONG_FLOAT);
   use FLT_IO;

   procedure POWER_PROMPT(NUM, DEN: out integer; Epsilon: out LONG_FLOAT) is

      begin

      set_page_length(24);

      loop

         new_page;

         put("Enter the power of the numerator(must be integer)  ");
         get(NUM);
         new_line;

         put("Enter the power of the denominator(must be integer)   ");
         get(DEN);
         new_line;

         put("Enter the epsilon of convergence.");
         put("This must be a real fraction and entered as 0.x");
         put("Where x is any string of digits up to 9 in length  ");
         get(EPSILON);
         new_line;

         exit;
      end loop;
      exception
         when data_error =>
            put_line("Invalid Entry. Reenter data");

   end POWER_PROMPT;
```

```
procedure GET_COEFFICIENTS(STRUCTURE: in character; POWER: in integer) is


     COEFF           : LONG_FLOAT    := 0.0;
     FROM, TO, BY    : integer;

     procedure GET_POWER(NUMBER: in integer; COEFF: out LONG_FLOAT) is

        LAST_SWAP    : integer := COEFFICIENT'last-1;
        NUMERATOR    : boolean := TRUE;
        OUT_COEFF    : LONG_FLOAT;
        CHAR_PTR     : integer;
        NUM, DEN     : string(1..15);
        INPUT_ERROR  : exception;


        procedure COMPUTE_REAL_COEFF(NUM, DEN: in  string;
                                     COEFF: out LONG_FLOAT) is

           CHAR_PTR:            integer      := NUM'first;
           NUMERATOR:           LONG_FLOAT   := 0.0;
           DENOMINATOR, SIGN:   LONG_FLOAT   := 1.0;


           begin -- COMPUTE_REAL_COEFF

           if (NUM(CHAR_PTR) = '+') then
              CHAR_PTR := CHAR_PTR + 1;
           elsif (NUM(CHAR_PTR) = '-') then
              SIGN := -SIGN;
              CHAR_PTR := CHAR_PTR + 1;
           end if;

           while ((NUM(CHAR_PTR) /= ' ') and (CHAR_PTR <= NUM'last)) loop
              NUMERATOR := NUMERATOR * 10.0 +
                 LONG_FLOAT(character'pos(NUM(CHAR_PTR)) -
                 character'pos('0'));
           end loop;

           CHAR_PTR := DEN'first;
           if (DEN(CHAR_PTR) = '+') then
              CHAR_PTR := CHAR_PTR + 1;
           elsif (DEN(CHAR_PTR) = '-') then
              SIGN := -SIGN;
              CHAR_PTR := CHAR_PTR + 1;
           end if;

           while ((DEN(CHAR_PTR) /= ' ') and (CHAR_PTR <= DEN'last)) loop
              NUMERATOR := NUMERATOR * 10.0 +
                 LONG_FLOAT(character'pos(DEN(CHAR_PTR)) -
                 character'pos('0'));
           end loop;

           COEFF := NUMERATOR/DENOMINATOR*SIGN;

        end COMPUTE_REAL_COEFF;
```

```
begin  -- GET_POWER

new_page;

loop
   -- prompt the user
   put("Enter the coefficients for x**");
   put(NUMBER);
   put(" ==> ");
   get(COEFFICIENT);

   -- pack and seperate
   for I in COEFFICIENT'range loop
      if ('0' <= COEFFICIENT(I) and COEFFICIENT(I) <= '9') or
          COEFFICIENT(I) = '-' or COEFFICIENT(I) = '+' or
          COEFFICIENT(I) = '/' or COEFFICIENT(I) = ' ' then

          if COEFFICIENT(I) = '/' then
             CHAR_PTR := DEN'first;
             NUMERATOR := FALSE;
          elsif COEFFICIENT(I) = '+' or COEFFICIENT(I) = '-' or
                ('0' <= COEFFICIENT(I) and COEFFICIENT(I) <= '9')
             then
             if NUMERATOR then
                NUM(CHAR_PTR) := COEFFICIENT(I);
             else
                DEN(CHAR_PTR) := COEFFICIENT(I);
             end if;
             CHAR_PTR := CHAR_PTR + 1;
          end if;
       else
          raise INPUT_ERROR;
       end if;
    end loop;

    exit;
 end loop;

 COMPUTE_REAL_COEFF(NUM, DEN, OUT_COEFF);
 COEFF := OUT_COEFF;
 put(OUT_COEFF);
 new_line;

 exception
    when INPUT_ERROR => put_line("Input Error. Reenter value.");
                        new_line;

end GET_POWER;
```

```
begin  -- GET_COEFFICIENTS

     set_page_length(24);

     new_page;

     put_line("Enter the coefficients for each");
     put_line("power of the 'X' in fractional form.");
     put_line("If a sign is entered, it must be the ");
     put_line("first character. No blanks are allowed.");
     put_line("The max allowable size is 9 digits per");
     put_line("number.");
     new_line;
     put_line("Sample entries:   1/2 , +1/2 ,  or -1/2");
     new_line;

     TO := POWER;
     case STRUCTURE is
        when  '1'  => FROM := 0;
                      BY   := 1;
        when  '2'  => FROM := 0;
                      BY   := 2;
        when  '3'  => FROM := 1;
                      BY   := 2;
        when others=> FROM := 0;
                      BY   := 1;
     end case;

     while (FROM <= TO) loop
        GET_POWER(FROM, COEFF);
        MACLAURIN(FROM) := COEFF;
        FROM := FROM + BY;
     end loop;

  end GET_COEFFICIENTS;
```

```
function PRODUCT(FROM, TO, BY: integer) return LONG_FLOAT is

      RESULT: LONG_FLOAT := 1.0;
      LOOP_TEST: integer := FROM;

      begin

      while (LOOP_TEST <= TO) loop
         RESULT := RESULT * LONG_FLOAT(LOOP_TEST);
         LOOP_TEST := LOOP_TEST + BY;
      end loop;

      return(RESULT);

   end PRODUCT;
```

```
function FACTORIAL (NUMBER: integer) return LONG_FLOAT is

      RESULT: LONG_FLOAT := 1.0;

      begin

      for I in 2..NUMBER loop
         RESULT := RESULT * LONG_FLOAT(I);
      end loop;

      return(RESULT);

   end FACTORIAL;



end COMMON_PROCS;
```

```
--
-- Date:              28 November 1985
-- Version:           1.0
-- Name:              FUNCTION_PACKAGE
-- Module Number:     4.0
-- Description:       This package contains modules that are called to either
--                    compute a predefined power series expansion of a function
--                    or allow a users to enter their own
-- Passed Variables:  N/A
-- Returns:           N/A
-- Globals Used:      GLOBAL_DATABASE
-- Modules Called:    None
-- Author:            Capt. Steven A. Hotchkiss and
--                    Capt. Jennifer Fried
-- History:           Developed as a thesis and ADA project


package FUNCTION_PACKAGE is

   procedure SIN_SERIES;

   procedure TAN_SERIES;

   procedure ASIN_SERIES;

   procedure ATAN_SERIES;

   procedure EXP_SERIES;

   procedure BUILD_SERIES;

end FUNCTION_PACKAGE;
```

```
with GLOBAL_DATABASE;     use GLOBAL_DATABASE;
with COMMON_PROCS;        use COMMON_PROCS;
with TEXT_IO;             use TEXT_IO;
package body FUNCTION_PACKAGE is


   procedure SIN_SERIES is

      N: integer;

      begin
      -- get the powers of the numerator and denominator polynomials.
      -- Also prompt the user for a convergent epsilon.
      POWER_PROMPT(M,K,EPS);

      -- Compute the power of the Maclaurin series. It is the sum of the power
      -- of the numerator, denominator, and the value two
      N := M + K + 2;

      -- Compute the initial approximating polynomial
      for I in 0..25 loop
         MACLAURIN(I) := 0.0;
      end loop;

      for I in 1..((N+1)/2) loop
         MACLAURIN(I*2-1) := -1.0**I/FACTORIAL(2*I-1);
      end loop;

   end SIN_SERIES;
```

```
procedure TAN_SERIES is

   N: integer;

   begin
   -- get the powers of the numerator and denominator polynomials.
   -- Also prompt the user for a convergent epsilon.
   POWER_PROMPT(M,K,EPS);

   -- Compute the power of the Maclaurin series. It is the sum of the power
   -- of the numerator, denominator, and the value two
   N := M + K + 2;

   -- Compute the initial approximating polynomial
   for I in 0..25 loop
      MACLAURIN(I) := 0.0;
   end loop;

   MACLAURIN(1) := 1.0;

   for I in 1..((N+1)/2) loop
      MACLAURIN(2*I+1) := PRODUCT(2, 2*I, 2) /
                          FACTORIAL(2*I+1);
   end loop;

end TAN_SERIES;
```

```
procedure ASIN_SERIES is

   N: integer;

   begin
   -- get the powers of the numerator and denominator polynomials.
   -- Also prompt the user for a convergent epsilon.
   POWER_PROMPT(M,K,EPS);

   -- Compute the power of the Maclaurin series. It is the sum of the power
   -- of the numerator, denominator, and the value two
   N := M + K + 2;

   -- Compute the initial approximating polynomial
   for I in 0..25 loop
      MACLAURIN(I) := 0.0;
   end loop;

   for I in 1..((N+1)/2) loop
      MACLAURIN(I*2-1) := PRODUCT(1, ((I-2)*2+1),2) /
                          PRODUCT(2,(I*2-2),2) *
                          LONG_FLOAT(I*2-1);
   end loop;

end ASIN_SERIES;
```

```
procedure ATAN_SERIES is

   N: integer;

   begin
   -- get the powers of the numerator and denominator polynomials.
   -- Also prompt the user for a convergent epsilon.
   POWER_PROMPT(M,K,EPS);

   -- Compute the power of the Maclaurin series. It is the sum of the power
   -- of the numerator, denominator, and the value two
   N := M + K + 2;

   -- Compute the initial approximating polynomial
   for I in 0..25 loop
      MACLAURIN(I) := 0.0;
   end loop;

   for I in 1..((N+1)/2) loop
      MACLAURIN(2*I-1) := -1.0**(I-1)/FACTORIAL(2*I-1);
   end loop;

end ATAN_SERIES;
```

```
procedure BUILD_SERIES is

   N          : integer;
   STRUCTURE  : character;

   begin

   set_page_length(24);

   -- get the powers of the numerator and denominator polynomials.
   -- Also prompt the user for a convergent epsilon.
   POWER_PROMPT(M,K,EPS);

   -- Compute the power of the Maclaurin series. It is the sum of the power
   -- of the numerator, denominator, and the value two
   N := M + K + 2;

   -- Compute the initial approximating polynomial
   for I in 0..25 loop
      MACLAURIN(I) := 0.0;
   end loop;

   -- Prompt the user for the structure of the polynomial
   new_page;

   L1:
   loop
      put_line("Enter  1  if all powers of X");
      put_line("Enter  2  if only even powers of X");
      put("Enter  3  if only odd  powers of X ==> ");
      get(STRUCTURE);
      new_line;

      if '1' > STRUCTURE or STRUCTURE >'3' then
         put_line("Bad Entry. Try again.");
      else
         GET_COEFFICIENTS(STRUCTURE,N);
      end if;

      exit L1;

   end loop L1;

end BUILD_SERIES;
```

```
      procedure EXP_SERIES is

         N: integer;

         begin
         -- get the powers of the numerator and denominator polynomials.
         -- Also prompt the user for a convergent epsilon.
         POWER_PROMPT(M,K,EPS);

         -- Compute the power of the Maclaurin series. It is the sum of the power
         -- of the numerator, denominator, and the value two
         N := M + K + 2;

         -- Compute the initial approximating polynomial
         for I in 0..25 loop
            MACLAURIN(I) := 0.0;
         end loop;

         for I in 0..N loop
            MACLAURIN(I) := 1.0 / FACTORIAL(I);
         end loop;

      end EXP_SERIES;
   end FUNCTION_PACKAGE;
```

```
--
--Date:              28 November 1985
-- Version:          1.0
-- Name:             APPROXIMATORS
-- Module Number:    5.0
-- Description:      This package includes procedures that compute
--                   approximations to user selected functions
-- Passed Variables: N/A
-- Returns:          N/A
-- Globals Used:     GLOBAL_DATABASE
-- Modules Called:   N/A
-- Author:           Capt. Steven A. Hotchkiss and
--                   Capt Jennifer Fried
-- History:          Developed as a thesis and ADA project


package APPROXIMATORS is

   procedure COMPUTE_TCHEBYSHEV;

   procedure MENU(CHOICE: out character);

   procedure COMPUTE_PADE_APPROXIMATIONS;

   procedure COMPUTE_CMK;

end APPROXIMATORS;
```

```
with GLOBAL_DATABASE;     use GLOBAL_DATABASE;
with FUNCTION_PACKAGE;    use FUNCTION_PACKAGE;
with TEXT_IO;             use TEXT_IO;
package body APPROXIMATORS is

   package FLT_IO is new FLOAT_IO(LONG_FLOAT);   use FLT_IO;

   procedure COMPUTE_TCHEBYSHEV is
      begin
      -- build the global table "T" containing the coefficients for
      -- each of a series of Tchebyshev polynomials

      T(0,0) := 1.0;
      T(1,1) := 1.0;
      T(2,0) := -1.0;
      T(2,2) := 2.0;

      for I in 3..25 loop

         for J in 0..25 loop
            T(I,J) := T(I,J) - T(I-2,J);
         end loop;

         for J in 0..24 loop
            T(I,J+1) := T(I,J+1) + 2.0 * T(I-1,J);
         end loop;

      end loop;

   end COMPUTE_TCHEBYSHEV;
```

```
procedure MENU(CHOICE: out character) is

     OUT_CHOICE: character;
     BAD_CHOICE: exception;

     begin

     set_page_length(24);

     -- clear screen and print menu
     new_page;
     put_line("Choose function to be approximated");
     new_line;

     put_line("Enter  1  for  sin");
     put_line("Enter  2  for  tan");
     put_line("Enter  3  for  arcsin");
     put_line("Enter  4  for  arctan");
     put_line("Enter  5  for  exp");
     put_line("Enter  6  for user defined function");
     put_line("Enter  7  to quit");

     loop

        for I in MACLAURIN'range loop
           MACLAURIN(I) :=  0.0;
        end loop;

        new_line;
        put("==> ");
        get(OUT_CHOICE);
        CHOICE := OUT_CHOICE;
        case OUT_CHOICE is

           when '1' => SIN_SERIES;
           when '2' => TAN_SERIES;
           when '3' => ASIN_SERIES;
           when '4' => ATAN_SERIES;
           when '5' => EXP_SERIES;
           when '6' => BUILD_SERIES;
           when '7' => null;
           when others => raise BAD_CHOICE;

        end case;
        exit;

     end loop;
     exception
        when BAD_CHOICE =>
              put_line("Invalid entry. Try again");

  end MENU;
```

```
procedure COMPUTE_PADE_APPROXIMATIONS is

   N_MAX: integer;
   NUM  : integer := 0;
   DEN  : integer := 1;
   TEMP : LONG_FLOAT;
   WORK : MATRIX;
   B    : VECTOR;

   begin

   -- this procedure converts the initial approximating polynomial
   -- (the Maclaurin power series) into a rational approximation
   -- clear out this PADE approximation's numerator
   -- and denominator polynomials
   for SERIES in 0..25 loop
      for NUM_DEN in 0..1 loop
         for COEFFICIENT in 0..25 loop
            A(SERIES,NUM_DEN,COEFFICIENT) := 0.0;
         end loop;
      end loop;
   end loop;


   for I in 0..M loop     -- loop for all powers of the numerator

      for J in 0..K loop  -- loop for all powers of the denominator

         if (I >= J) then
            -- build a work matrix to solve simultaneous equations
            B(0) := 1.0;
            N_MAX := I + J;

            for S in 0..(N_MAX - I - 1) loop
               for N1 in 0..J loop
                  WORK(S+1,N1) := MACLAURIN(abs(N_MAX - S - N1));
                  if (N1 = 0) then
                     B(S+1) := -MACLAURIN(abs(N_MAX - S - N1));
                  end if;
               end loop;
            end loop;

            -- Solve simultaneous equations for denominator coefficients
            for N1 in 1..J loop
               if (WORK(N1,N1) = 0.0) then
                  SETUP:
                  for N2 in 1..J loop
                     if (WORK(N2,N1)/= 0.0) then
                        TEMP := B(N2);
                        B(N2) := B(N1);
                        B(N1) := TEMP;
                        for N3 in 1..J loop
                           TEMP := WORK(N2,N3);
                           WORK(N2,N3) := WORK(N1,N3);
                           WORK(N1,N3) := TEMP;
                         end loop;
                         exit SETUP;
                      end if;
                   end loop SETUP;
```

```
                end if;

                TEMP := WORK(N1, N1);
                if TEMP /= 0.0 then
                   B(N1) := B(N1)/TEMP;
                else
                   B(N1) := 0.0;
                end if;

                for N2 in 1..J loop
                   if TEMP /= 0.0 then
                      WORK(N1,N2) := WORK(N1,N2)/TEMP;
                   else
                      WORK(N1,N2) := 0.0;
                   end if;
                end loop;

                for N2 in 1..J loop
                   if (N1 /= N2) then
                      TEMP := -WORK(N2,N1);
                      for N3 in 1..J loop
                         WORK(N2,N3) := WORK(N2,N3) + WORK(N1,N3) * TEMP;
                      end loop;
                      B(N2) := B(N2) + B(N1) * TEMP;
                   end if;
                end loop;

             end loop;

             -- use denominator coefficients to compute the numerator
             -- coefficients, and build the series of PADE approximations
             for N1 in 0..I loop
                for N2 in 0..N1 loop
                  R(I+J,NUM,N1) := R(I+J,NUM,N1) + B(N2) * MACLAURIN(N1-N2)/
                                   B(0);
                end loop;
             end loop;
             for N1 in reverse 0..J loop
                R(I+J,DEN,N1) := B(N1)/B(0);
                B(N1) := B(N1) / B(0);
             end loop;

             -- Compute the D's that are used to compute C(m,k)
             -- D(I+J+1) = SUM[L=0 to J (Maclaurin(I+J+1-L)*B(L)]
             D(I+J+1) := 0.0;
             for L in 0..J loop
                D(I+J+1) := D(I+J+1) + MACLAURIN(I+J+1-L) * B(L);
             end loop;

          end if;
       end loop;
    end loop;

end COMPUTE_PADE_APPROXIMATIONS;
```

```
procedure COMPUTE_CMK is

      A: integer := 0;
      B: integer := 1;
      LAMDA: VECTOR;

      begin


      -- Compute the Lamdas (alpha=1)
      LAMDA(0) := -(D(M+K+1) * T(M+K,0)) / (2.0**(M+K));

      for J in 0..(M+K-1) loop
         if D(J+1) /= 0.0 then
            LAMDA(J+1) := (D(M+K+1) * T(M+K+1,J+1)) / ((2.0 **(M+K)) * D(J+1));
         else
            LAMDA(J+1) := 0.0;
         end if;
      end loop;


      -- Load Pm(X) and Qm(X) with their  A and B coefficients respectively
      for I in 0..M loop
         C(A,I) := R(M+K,A,I);
      end loop;

      for I in 0..K loop
         C(B,I) := R(M+K,B,I);
      end loop;

      -- Compute coefficients "A" of numerator and "B" of denominator
      for J in 0..(M+K-1) loop
         for K in 0..25 loop
            R(J,A,K) := R(J,A,K) * LAMDA(J+1);
            C(A,K) := C(A,K) + R(J,A,K);
            R(J,B,K) := R(J,B,K) * LAMDA(J+1);
            C(B,K) := C(B,K) + R(J,B,K);
         end loop;
      end loop;
      C(A,0) := C(A,0) + LAMDA(0);

      for I in reverse 0..25 loop;
         C(A,I) := C(A,I)/C(B,0);
         C(B,I) := C(B,I)/C(B,0);
      end loop;


   end COMPUTE_CMK;

end APPROXIMATORS;
```

Bibliography

1. TRW. "A Study of Embedded Computer Systems Support," ECS Technology Forecast, 8: (September 1980).

2. Department of the Air Force. Military Sixteen-Bit Computer Instruction Set Architecture. MIL-STD-1750A. Washington: Government Printing Office, 1980

3. Lynn, H. C. and R. K. Moore."MIL-STD-1750 Chip Set: Possible Designs," 4th AIAA/IEEE Digital Avionics System Conference. 168-172. A Collection of Technical Papers. New York: American Institute of Aeronautics and Astronautics, (November 17-19, 1981).

4. Cody, William J. and William Waite. Software Manual for the Elementary Functions,:Englewood Cliffs, N.J.: Prentice-Hall Inc., 1980.

5. Thayer, T. A. "Understanding Software Through Analysis of Empirical Data," Proceedings of the 1975 National Computer Conference (44). 335-41. Montvale, N.J.: AFIPS Press,1975.

6. Boehm, B.W., R. L. McClean, and D. B. Urfrig, "Some Experiences with Automated Aids to the Design of Large-Scale Reliable Software," IEEE Transactions on Software Engineering (SE-1).125-33. March 1975

7. Peters, L. J. Software Design: Methods and Techniques. New York: Yourdon Press, 1981

8. Jensen, R. W. "Structured Programming," Computer.31-48, March 1981

9. Conte, S.D. and Carl de Boor. Elementary Numerical Analysis, An Algorithmic Approach (Second Edition). New York: McGraw-Hill Book Company, 1972

10. Ralston, Anthony. A First Course In Numerical Analysis. New York: McGraw-Hill Book Company,1965

11. Hart, John F. Computer Approximations (Second Edition).Huntington, N.Y.: Robert E. Krieger Publishing Company, 1978

VITA

Captain Jennifer J. Fried was born on 19 October 1951 at Ft. Sill, Oklahoma. In May of 1969, she graduated from High School in Newport News, Virginia. She later enlisted in the United States Air Force as a Computer Programmer, and was assigned to Holloman AFB, New Mexico. In 1979, she was accepted into the Airman Education and Commissioning Program and attended New Mexico State University. Upon receiving a Bachelor of Science in Computer Science and Mathematics in January 1981, she was sent to Officer Training School. Upon graduation, she was stationed at Peterson AFB, Colorado where she became Chief of the Missile Warning/Space Computer Test Section. While working toward a degree of Master of Science in Computer Data Management, she was selected to enter the School of Engineering, Air Force Institute of Technology, in June of 1984.

Permanent address: 5113 Windgate Court
Colorado Springs, Colorado 80917

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/MA/85D-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19	

12. PERSONAL AUTHOR(S)
Jennifer J. Fried, B.S., Capt, USAF

13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM ______ TO ______	14. DATE OF REPORT (Yr., Mo., Day) 1985 December 4	15. PAGE COUNT 149

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.
12	01	
09	01	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Functions (Mathematics), Approximations,
Computer Programs, MIL-STD-1750A

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: DEVELOPMENT AND EVALUATION OF MATH LIBRARY
ROUTINES FOR A 1750A AIRBORNE MICROCOMPUTER

Thesis Chairman: Panna B. Nagarsenker
Associate Professor of Mathematics and Computer Science

Approved for public release: IAW AFR 190-1.
[illegible] 16 Jan 86
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AFC)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL Panna B. Nagarsenker	22b. TELEPHONE NUMBER (Include Area Code) 513-255-7210	22c. OFFICE SYMBOL AFIT/ENC

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

This project produced a run-time math library for the MIL-STD-1750A embedded computer architectures. The math library consists of the algebraic functions. In addition, the steps required for a performance analysis of the math library have been outlined.

Several approximation methods were investigated. The Chebyshev Economization of Maclaurin series polynomials, and rational approximations derived from the second algorithm of Remes were determined to be the best methods available. Each function's implementaion was designed to take advantage of features of MIL-STD-1750A architectures. The recommended test procedures provide measures of the average and worst case generated errors within each approximation.

END

FILMED

3-86

DTIC