

AD-A163 905

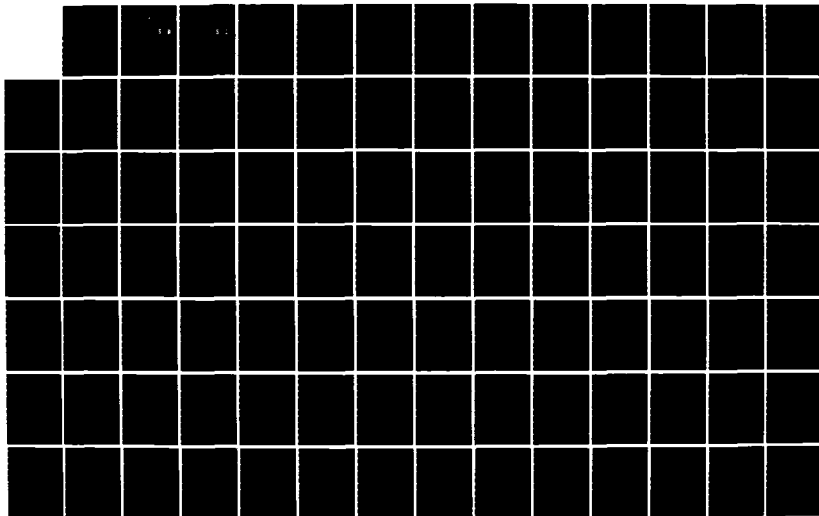
DEVELOPMENT OF A RUN TIME MATH LIBRARY FOR THE 1750A
AIRBORNE MICROCOMPUTER(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
S A HOTCHKISS DEC 85 AFIT/GCS/MA/85D-5

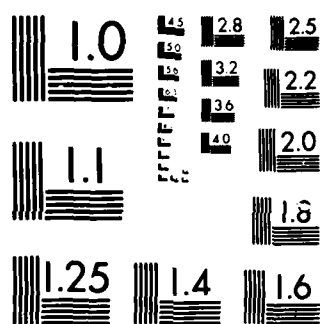
1/3

UNCLASSIFIED

F/G 9/2

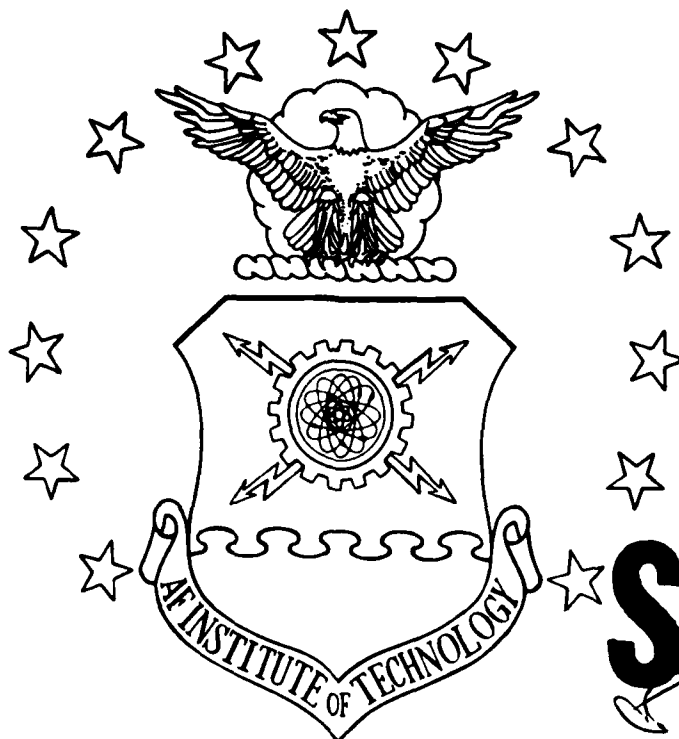
NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

AD-A163 905



DTIC
ELECTE
FEB 11 1986
S D

DEVELOPMENT OF A RUN TIME MATH LIBRARY
FOR THE 1750A AIRBORNE MICROCOMPUTER

THESIS

Steven A. Hotchkiss
Captain, USAF

AFIT/GCS/MA/85D-5

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DTIC FILE COPY

16 2 10 021

AFIT/GCS/MA/85D-5

DTIC
ELECTE
FEB 11 1986
S D D

DEVELOPMENT OF A RUN TIME MATH LIBRARY
FOR THE 1750A AIRBORNE MICROCOMPUTER

THESIS

Steven A. Hotchkiss
Captain, USAF

AFIT/GCS/MA/85D-5

Approved for public release; distribution unlimited

AFIT/GCS/MA/85D-5

DEVELOPMENT OF A RUN TIME MATH LIBRARY
FOR THE 1750A AIRBORNE MICROCOMPUTER

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Information Systems

Steven A. Hotchkiss, M.S.
Captain, USAF

December 1985

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail. and/or Special
A-1	

Approved for public release; distribution unlimited

Preface

The purpose of this thesis was to develop and evaluate the performance of a run time math library for those architectures conforming to MIL-STD-1750A, and is intended to serve as a benchmark for future contractor development. The routines implemented include all circular trigonometric functions and their inverses. Appendix A contains the descriptions of pseudo-operations I've used in explaining the design of these functions, and will prove useful to you as you try to follow my logic.

In developing and doing the performance 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 during those times of confusion and near panic. I am also indebted to Mr. Bobby Evans and Mr. Dale Lange, from the sponsoring organization, for all the help that they gave me in getting all the equipment and outside information that I needed. A special thanks also goes out to my friend Heidi for all her support and patience.

Steven A. Hotchkiss

Table of Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	viii
Abstract	ix
I. Introduction	1
Background	1
Problem	3
Assumptions	5
General Approach	6
Sequence of Pesentation	11
II. Theoretical Development	12
General Discussion	12
Approximation Techniques	14
Polynomials	14
Definition of Chebyshev Polynomials	16
Chebyshev Economization	17
Rational Approximations	19
The Second Algorithm of Remes	21
Design Considerations	26
Sin and Cos	26
Tan and Cot	28

	Page
Asin and Acos	31
Atan	32
III. Development and Design of the Functions	35
General Discussion	35
Sin and Cos Implementation	36
Fixed-Point	36
Floating-Point	43
Tan and Cot Implementation	51
Fixed-Point	51
Floating-Point	58
Asin and Acos Implementation	63
IV. Validation Verification and Performance Evaluation	70
General Discussion	70
Manual Static Analysis Methods	70
Critical Value Testing	72
Performance Evaluation	73
V. Conclusions and Recommendations	76
Conclusions	76
Recommendations	79
Appendix A: Pseudo Operations	81
Appendix B: Function Source Code	86

	Page
Appendix C: Support Software	127
Appendix D: VMS Command Files	146
Appendix E: Approximation Algorithms	151
Bibliography	189
Vita	191

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. Periodicity, Symmetry, Antisymmetry, Identity Properties for (a) sin; (b) cos	27
4. Periodicity, Antisymmetry, Identity Properties for (a) Tan; (b) Cot	29
5. Graph of the inverses of Sine and Cosine	31
6. Graph of the Inverse of Tangent	33
7. Cos and Sin Structured Flowchart	37
8. Sincos Fixed-Point Structured Flowchart	38
9. Sinf and Cosf Structured Flowchart	44
10. Sincosf Structured Flowchart	45
11. Bit Layout of 1750A Floating-Point Numbers	49
12. Tan and Cot Structured Flowchart	51
13. Tancot Structured Flowchart	52
14. Tanf and Cotf Structured Flowchart	59

Figure	Page
15. Tancotf Structured Flowchart	60
16. Asin and Acos Structured Flowchart	64
17. Asincos Stuctured Flowchart	65

List of Tables

Table	Page
1. Information Flow of the Logicalized Software Development Cycle	8
2. Chebyshev Polynomials and Powers of Chebyshev Polynomials	17
3. Coefficients for Polynomial Approximation of Sin	43
4. Coefficients for Polynomial Approximation to Sinf	50
5. Coefficients for Polynomial Approximation to Tan	57
6. Execution Times for 1750A Instructions From Sperry 1631 Programmer Reference Manual	61
7. Coefficients for Polynomial Approximations to Tanf	62
8. Asin/Acos Fixed and Floating-Point Coefficients	67

Abstract

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

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

I. Introduction

Background

The Air Force has a vested interest in reducing the life-cycle costs of its avionics weapon systems. Standardization of high order languages and an Instruction Set Architecture (ISA) is one way the Air Force feels it 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 in 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 will come 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; linker/loader programs; and a 1750A acceptance test program (1: 8.4). Other cost benefits will be realized through the independent development of software and hardware, (2:1) and common maintenance and test equipment. (3:168)

Standardization of languages will also have 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. 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 languages approved by this instruction.

As previously mentioned, the development of a standard ISA such as MIL-STD-1750A will help reduce total life-cycle costs of Air Force avionics weapon systems. This reduction will come partially through the use of common support software tools, many which have already been developed. It was also mentioned that one of the support software tools already developed includes a JOVIAL compiler that generates 1750A ISA code; however, a math library containing all the algebraic and trigonometric functions required by these languages 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 will help the Air Force reduce avionics weapon systems cost through the development of a math library for software support of all 1750A systems.

Problem

At the time of this writing there are no math libraries written that 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 provided is written in the D.O.D approved language JOVIAL. The coding of the library was only a small effort of this thesis. Most of the detail has gone into verification, validation, and performance evaluation of the product. Inasmuch, 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 reinvent the wheel each time a function is needed. Libraries also provide a means for using functions that take full advantage of a particular computer architecture computer architecture.

The design of a procedure for computing the value of functions is not mathematically complete in 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 number of integer and fraction bits in fixed-point numbers, 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 functions define the problem addressed by this thesis effort.

Scope

This effort was limited to the design, code, and performance evaluation of the circular and inverse circular trigonometric 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: sine (sin), cosine (cos), tangent (tan), cotangent (cot), arcsine (asin), arccosine (acos), and arctangent (atan and atan2).

All functions have been written to either accept and return double precision fixed-point values, or to accept and return extended precision floating-point values. Floating-point functions are distinguished from the fixed-point functions by the name used to invoke them. All floating-point functions have an "f" concatenated to the end of them; otherwise, they have the same names as those used by the fixed-point functions. For example: the fixed-point sine function is invoked by using the name "sin", and the floating-point sine function is invoked by using the name "sinf".

Also provided are performance summaries for each of the functions, and algorithms that may be used to determine the polynomial coefficients for computing any of functions addressed by this paper. These algorithms can be found in Appendix A, and 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 errors 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 bears further investigation before implementing on a real-time system.

Another factor discussed during the design review was, that there is a need for both fixed-point and floating-point functions. Floating-point functions are more precise than fixed-point algorithms, but take longer to execute. Conversely, fixed-point functions don't have the precision, but are much faster. In addition, it was mentioned that many avionics applications also use what is termed as pi-radians. Pi-radians are angular units of measure expressed in terms of multiples of pi, and are equal to radian measures divided by pi. For example: 180° is equivalent to 3.141596 radians, or 1.0 pi-radian. If the algorithms use pi-radian measure rather than radian measure, there is no need for overflow condition checks, and therefore, a significant amount of work is eliminated from the domain reduction computations.

As stated earlier, algorithms for both fixed-point and floating-point type functions have been written. Because of time limitations, both methods of unit measure could not be implemented, and only the fixed-point algorithms use the pi-radian metric. The assumption is that if the need arises, this thesis can serve as a guide for implementing both function types in either unit of measure.

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 to the more commonly used "waterfall" method of software system development. The waterfall consists of the apparently neat, concise and logical ordering of a series of steps that must be accomplished 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 is more concerned with the problem definition side of the cycle (see figure 1). This approach makes it more useful for eliminating errors that are typically occur during the waterfall's requirements definition and design phases. Errors generated during these phases of the waterfall model 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.

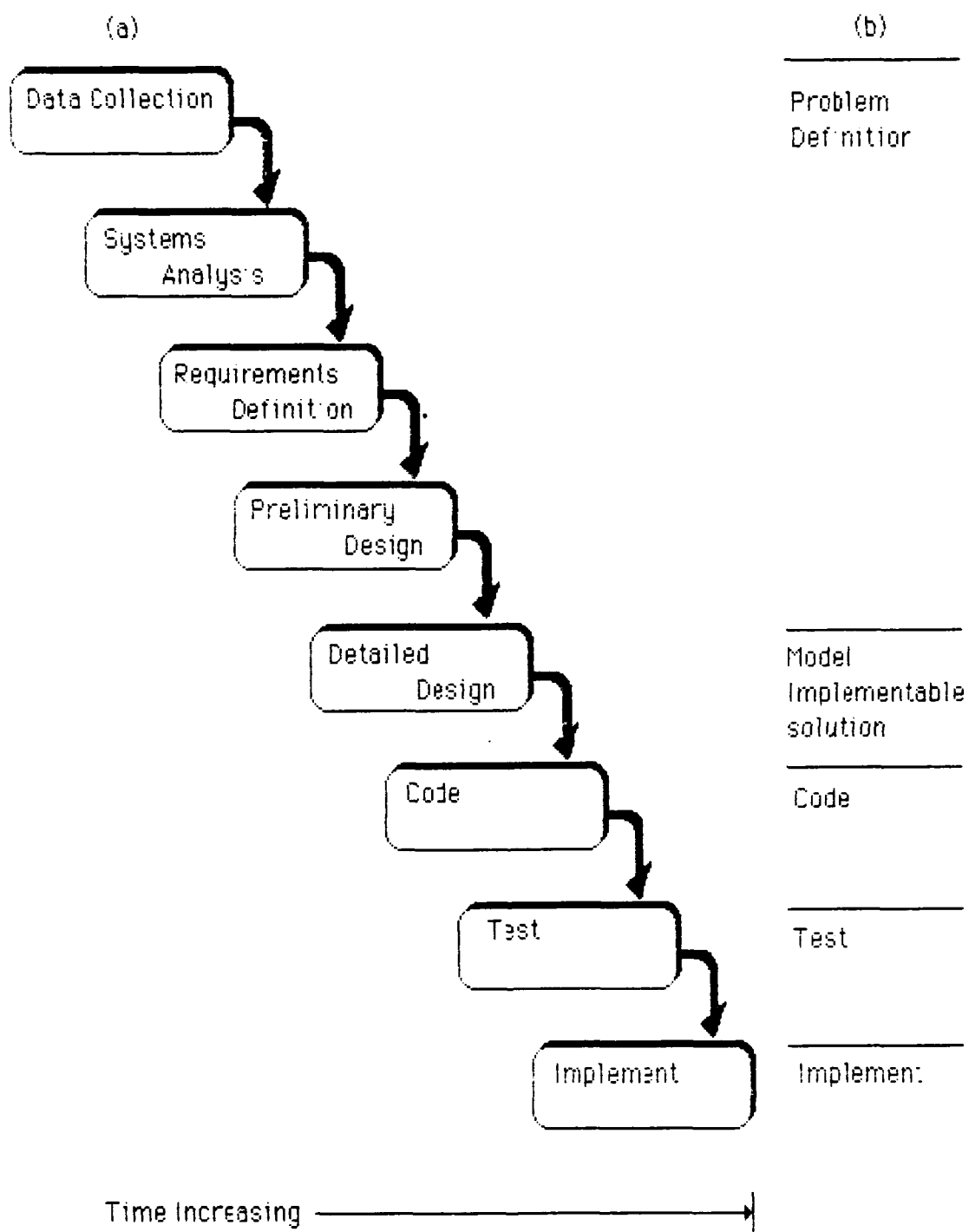


Figure 1 Chart of; (a) Waterfall; and (b) Logicalized Model of a Software Development Cycle

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

(6: 125-33) made it clear that these problems existed , and made the point that design errors not only outnumbered other errors, but that they were also more persistent. For that 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.

The information flow of a logicalized model is summarized in Table 1, and is "analogous to an artist's conception of a building: 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 that 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. From there, 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 trigonometric functions.

The baseline model served as a reference from which all decisions regarding actual implementation could be made. Before preceding 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; to minimize the effort required to correct errors in sections of code found to be deficient; upgrading sections when more reliable, functional, or efficient techniques are discovered; and to minimize the life-cycle costs of the software. (8:32) In addition it must 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 aforementioned reasons, but also for its simplicity and understandability from a reviewer standpoint.

Sequence of Presentation

This thesis addresses the 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). In particular, abstract design considerations for each of the implemented functions is discussed. The next area for discussion is 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 used.

Appendices include algorithms useful for determining the coefficients of each of the functions (appendix E), 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), and the VAX VMS command files required to compile link and run the developed product (Appendix D).

II. Theoretical Development

General Discussion

The purpose of this thesis was to create and analyze trigonometric 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.

Fortunately, the trigonometric functions lend themselves to domain reduction through the properties of periodicity, symmetry, and antisymmetry. This allows function arguments to be reduced so that a more accurate approximation may be calculated than what is possible without argument reduction. How these properties are actually used in domain reduction depends on the function, and are described in the following subsections.

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 one, 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 $\tau_n(x)/2^{n-1}$. 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 $[-1,1]$.

Alternation Theorem: 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.

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

$$\tau_n(x) = \cos(n\theta) \quad \begin{array}{l} \theta = \arccos(x) \\ n = 1, 2, \dots \end{array} \quad (1)$$

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.

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

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

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.

A	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 (**A**) Chebyshev Polynomials; (**B**) Powers of Chebyshev Polynomials

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 E contains an algorithm that generates both the Chebyshev polynomials, and their powers.

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).

$$P_n(x) = d_0\tau_0(x) + \dots + d_n\tau_n \quad (4)$$

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 $f(x)$.

$$P_n(x) = a_0 + a_1 x + \dots + a_n x^n \quad (5)$$

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

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.

$$P_k^*(x) = d_0 \tau_0(x) + \dots + d_k \tau_k \quad (7)$$

$$\varepsilon_n + d_{k+1} + \dots + d_n \leq \varepsilon \quad (8)$$

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

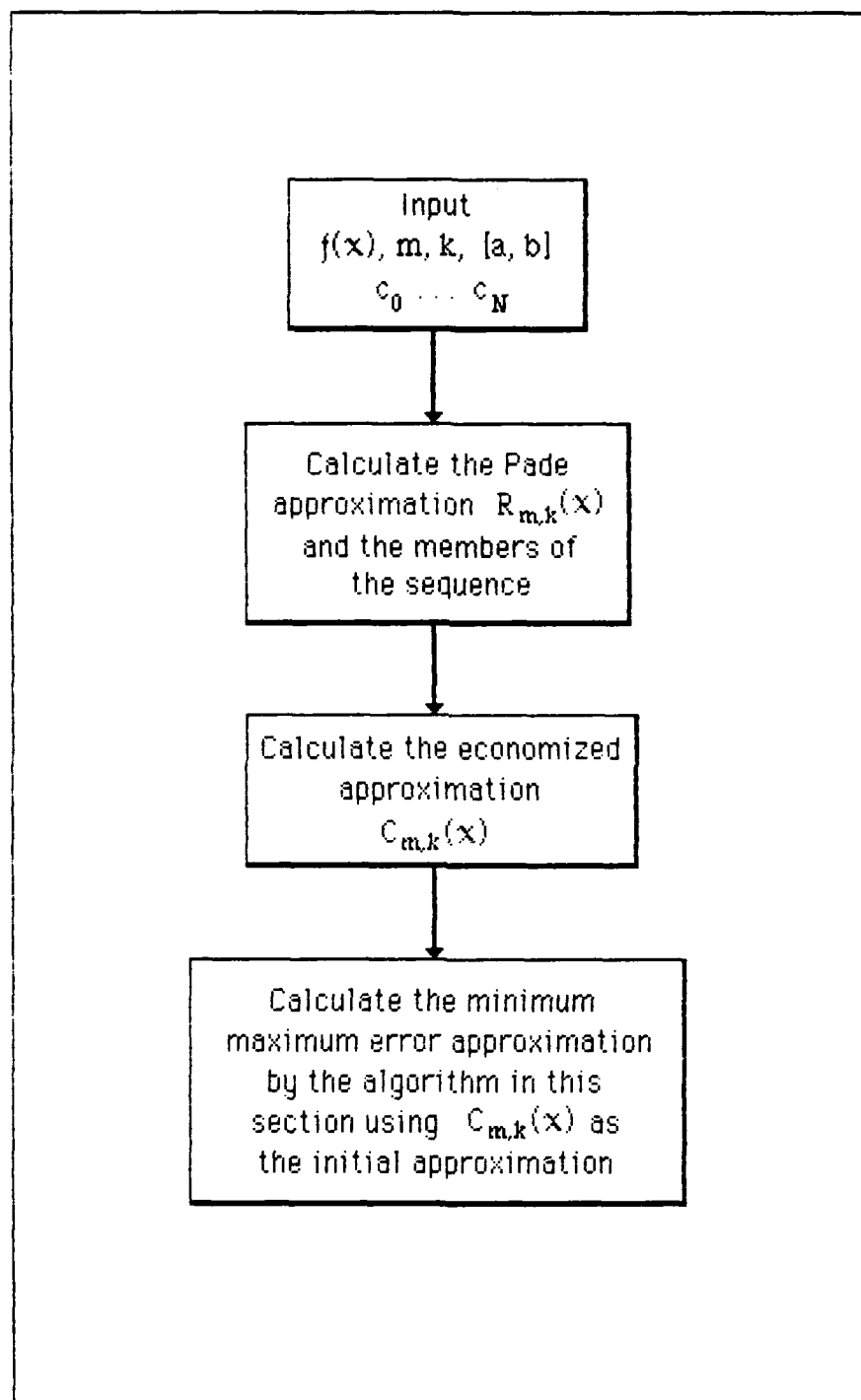


Figure 2 Calculation of Remes Rational Approximations

are several algorithms that generate rational approximations that can be considered Chebyshev approximations; however, the ones that generate the 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 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).

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

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

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 $C_0 \dots C_N$. 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 $C_0 \dots C_N$ 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 $R_{0,0}^{(0)}(x)$, $R_{1,0}^{(1)}(x)$, $R_{1,1}^{(2)}(x)$, $R_{2,1}^{(3)}(x)$, and $R_{2,2}^{(4)}(x)$. 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 $d^{(2,2)}_5$.

$$R_{i,j-1}^{(j)}(x) = \frac{P_i^{(j)}(x)}{Q_{j-1}^{(j)}(x)} \quad j = 0, \dots, N-1 \quad (11)$$

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

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.

$$\sum_{j=0}^k C_{N-s-j} b_j = 0 \quad \begin{matrix} s = 0, 1, \dots, N-m-1 \\ (C_j = 0 \text{ if } j < 0, \quad b_0 = 1) \end{matrix} \quad (13)$$

$$a_r = \sum_{j=0}^r C_{r-j} b_j \quad \begin{matrix} r = 0, 1, \dots, m \\ (b_j = 0 \text{ if } j > k) \end{matrix} \quad (14)$$

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.

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

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

$$C_{m,k}(x) = \frac{P_m(x) + \sum_{j=0}^{N-1} \gamma_{j+1} P_1^{(j)}(x) + \gamma_0}{Q_k(x) + \sum_{j=0}^{N-1} \gamma_{j+1} Q_{j-1}^{(j)}(x)} \quad (16)$$

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 $a_0^{(0)}, \dots, a_m^{(0)}, b_1^{(0)}, \dots, b_k^{(0)}$, 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.

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

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 , $i=0, \dots, N+1$. 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 $f(x) - R_{m,k}^{(0)}(x)$ 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 $x_i^{(1)}$.

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

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

Design Considerations

Sin and Cos. Close examination of figures 3a and 3b reveals a relationship that can be used to reduce the amount of storage space required by the routines that compute the values of these two functions. The relationship is expressed in (19). Therefore, by adjusting the argument that is passed to the Cos function, both it and the Sin function can be written as short linkages to another procedure that performs a majority of the calculations.

$$\cos(\alpha) = \sin(|\alpha| + \pi/2) \quad (19)$$

The three properties discussed in the preceding section, periodicity, symmetry, and antisymmetry, are illustrated in figures 3a and 3b. These

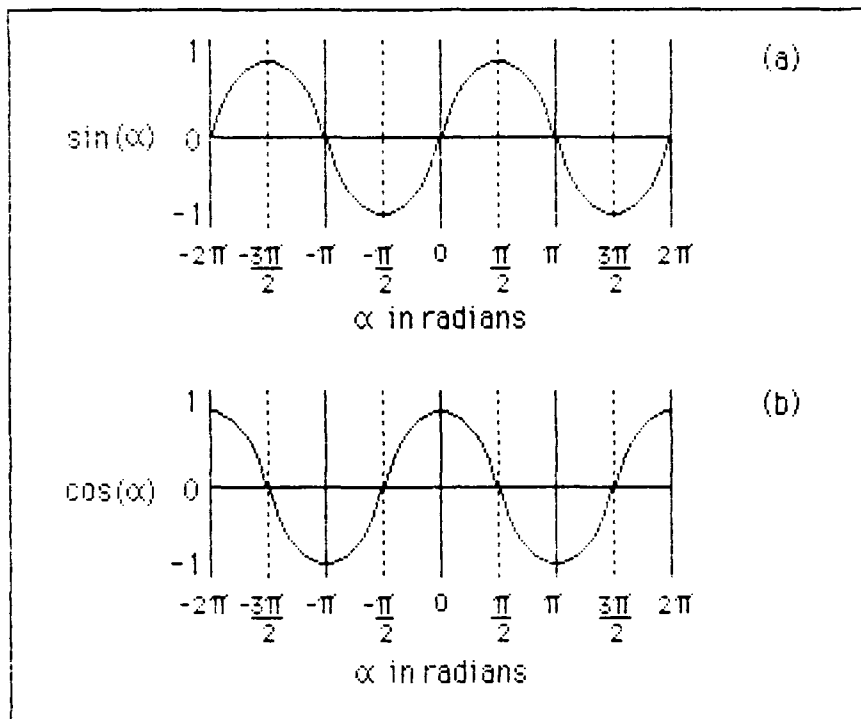


Figure 3 Periodicity, Symmetry, AntiSymmetry, Identity Properties for; (a) sin; (b) cos

properties are exploited to reduce an argument to within the smallest possible domain of values accurately representing $\sin(\alpha)$ or $\cos(\alpha)$ for all α . By the periodicity property shown in (20) and displayed in Figure 3a, it is easy to see that the argument α can be reduced to within the interval $[-2\pi, 2\pi]$. To compute $\sin(\alpha)$, determine β such that $\alpha = \beta + 2\pi k$, where k is an integer. Then $\sin(\alpha)$ equals $\sin(\beta)$.

$$\sin(\alpha) = \sin(\alpha + 2\pi) \quad (20)$$

Examination of Figure 3a also illustrates that $\sin(\alpha)$ is antisymmetric about the point zero- π . That is, $\sin(\alpha)$ equals $-\sin(-\alpha)$. By noting the sign change of the result, this property can be used to reduce an argument to within the interval $[0, 2\pi]$. The interval $[\pi, 2\pi]$ in $\sin(\alpha)$ is also antisymmetric with respect to the interval $[0, \pi]$. Therefore, by noting the sign change of this result and the relationship shown in (21), α can be set to $\alpha - \pi$. The argument is then reduced to within the interval $[0, \pi]$.

$$\sin(\alpha + \pi) = -\sin(\alpha) \quad (21)$$

Lastly, examination of Figure 3a also illustrates that $\sin(\alpha)$ is symmetric about the point $\pi/2$. As shown in (22), the argument α can be reduced to within the interval $[0, \pi/2]$, by setting α to $\pi - \alpha$.

$$\sin(\alpha + \pi/2) = \sin(\pi/2 - \alpha) \quad (22)$$

Tan and Cot. As with the sin and cos functions, close examination of the graphs for Tan and Cot (figures 4a and 4b) reveal a relationship that can be used to reduce the amount of storage space required to implement both functions. The relationship is expressed in equation (23). Therefore, by flagging which routine was originally called, both routines can serve as

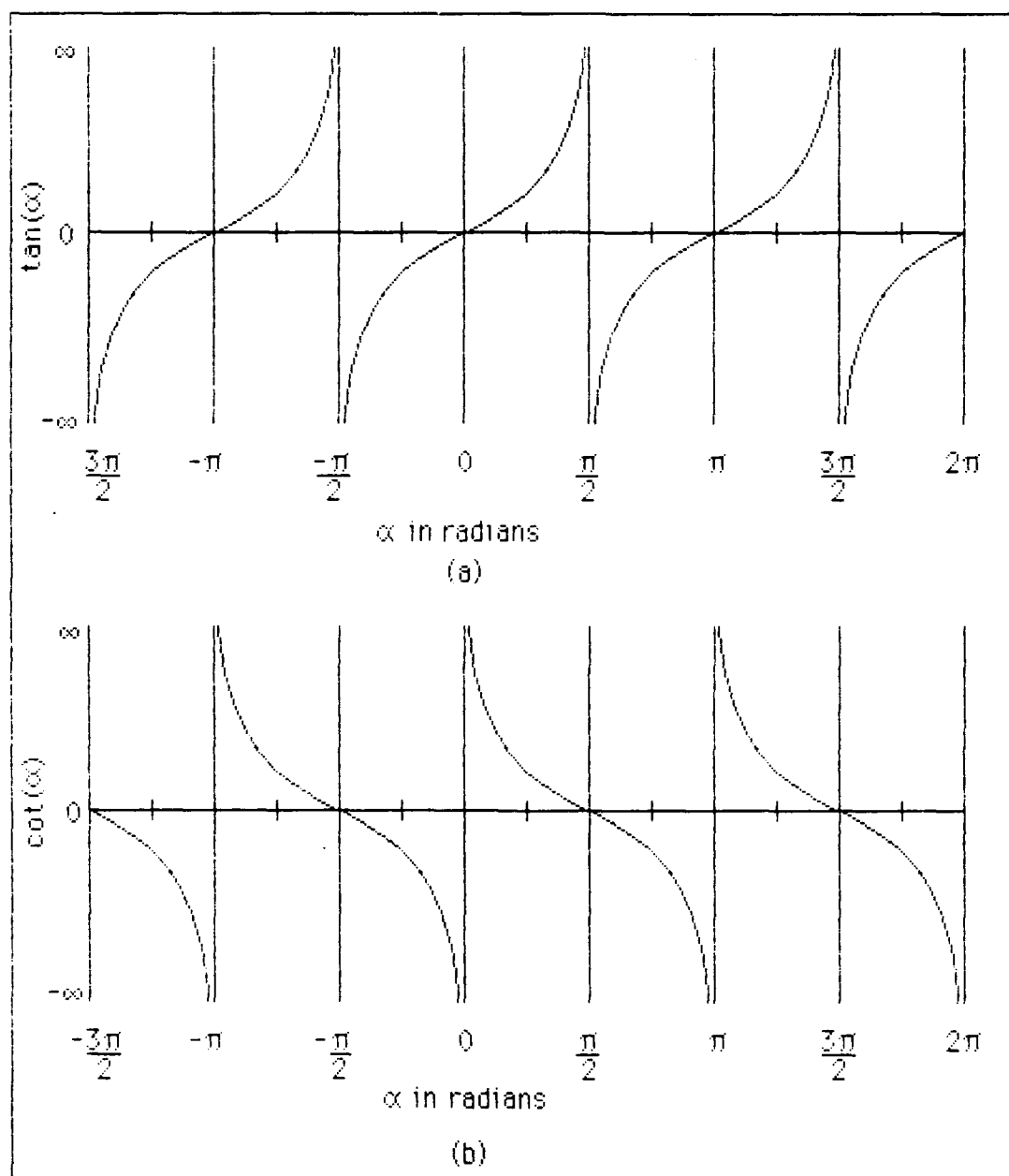


Figure 4 Periodicity, Antisymmetry, Identity Properties for
(a) Tan; (b) Cot

short linkages to another procedure that performs a majority of the calculations.

$$\cot(\alpha) = 1 / \tan(\alpha) \quad (23)$$

The properties of periodicity and antisymmetry are also illustrated in figures 4a and 4b. These properties are exploited to reduce an argument to within the smallest possible domain of values accurately representing $\tan(\alpha)$. By the property of periodicity shown in (24) and displayed in Figure 4a, it is easy to see that the argument α can be reduced to within the interval $[-\pi/2, \pi/2]$. To compute $\tan(\alpha)$, simply determine β such that $\alpha = \beta + \pi k$, where k is an integer. Then $\tan(\alpha)$ equals $\tan(\beta)$.

$$\tan(\alpha) = \tan(\alpha + \pi) \quad (24)$$

Examination of Figure 4a also illustrates that $\tan(\alpha)$ is antisymmetric about the point zero-pi. That is, $\tan(\alpha)$ equals $\tan(-\alpha)$. By noting the sign change of the result, this property can be used to reduce the argument to within the interval $[0, \pi/2]$.

Knowledge of plane trigonometry will allow reduction of the argument to within an even smaller interval. Let α be any angle whose vertex is at the origin of a rectangular Cartesian coordinate system. Also,

let $P(x, y)$ be any point on that angle. The variable x is measured along the horizontal axis, and y is measured along the vertical axis. Then the function $\tan(\alpha)$ is defined as y/x . From this, it can be shown that for any angle within the interval $[\pi/4, \pi/2]$, equation 25 holds true.

Therefore, by setting α to $\pi/2 - \alpha$ and returning the reciprocal of $\tan(\alpha)$, the angle α can effectively be reduced to lie within the interval $[0, \pi/4]$.

$$\tan(\alpha) = 1/\tan(\pi/2 - \alpha) \quad (25)$$

Asin and Acos. The equation $\beta = \sin(\alpha)$ determines an infinite number of real values for β within the interval $[-1, 1]$. Therefore, the inverse, $\alpha = \sin^{-1}(\beta)$, has many possible solutions. However, for $\sin^{-1}(\beta)$ to be a true function, there is the restriction of one α for every value β . For this reason, it is necessary to pick a range of principal values that will satisfy this restriction. Figure 5 depicts the range of principle values defined for this function, and is the interval $[-\pi/2, \pi/2]$.

Careful examination of Figure 5 reveals a relationship that can be used to reduce the amount of storage space required by the routines that compute the values of these functions. The relationship is expressed in (26). From the range of values defined for $\sin^{-1}(\alpha)$ (see Figure 5), and the relationship defined in (26); the range of values over which $\cos^{-1}(\alpha)$ is defined includes $[0, \pi]$. By adjusting the results that are computed in

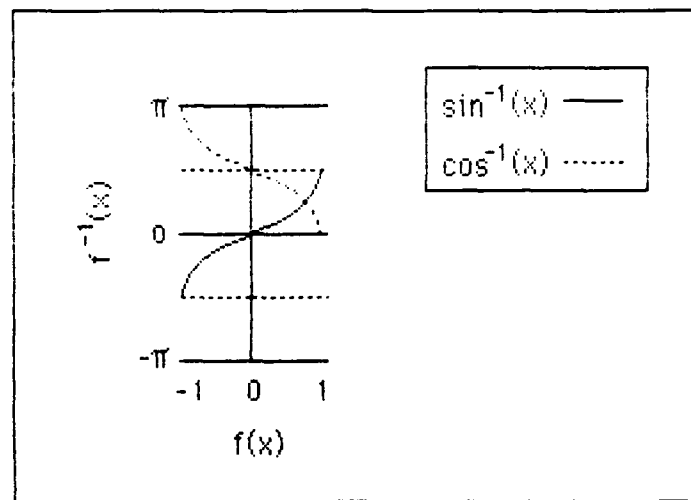


Figure 5 Graph of the Inverses of Sine and Cosine

a separate procedure, both Asin and Acos can serve as short linkages to that procedure.

$$\cos^{-1}(\alpha) = \pi/2 - \sin^{-1}(\alpha) \quad (26)$$

By restricting the argument of the functions to the interval $[0, .5]$, the identity function shown in (27) can also be used. Note that by solving for $\sin^{-1}(\alpha)$ in equation (26) and substituting the result into the left-hand-side of (27), we come up with the definition for $\cos^{-1}(\alpha)$ shown in (28).

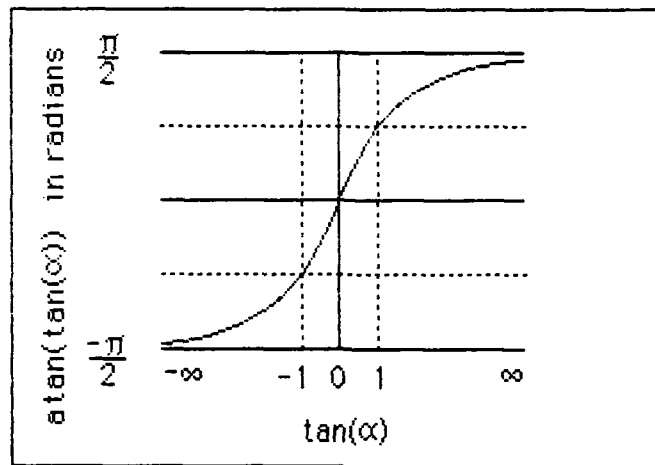


Figure 6 Graph of the Inverse of Tangent

$$\sin^{-1}(\alpha) = \pi/2 - 2 \sin^{-1} \sqrt{(1-\alpha)/2} \quad (27)$$

$$\cos^{-1}(\alpha) = 2 \sin^{-1} \sqrt{(1-\alpha)/2} \quad (28)$$

Atan. The equation $\beta = \tan(\alpha)$ determines an infinite number of real values for β within the interval $[-1, 1]$. Therefore, the inverse, $\alpha = \tan^{-1}(\beta)$, has many possible solutions. However for $\tan^{-1}(\beta)$ to be a true function, there can only be one α for every value β . For this reason, it is necessary to pick a range of principal values that will satisfy this restriction. Figure 6 depicts the range of principal values defined for this function, and is the interval $[-\pi/4, \pi/4]$.

Examination of Figure 6 illustrates that $\tan^{-1}(\alpha)$ is antisymmetric about the point zero. That is, $\tan^{-1}(\alpha) = -\tan^{-1}(-\alpha)$. By noting the sign

change of the result, this property can be used to reduce an argument to within the interval $[0, \infty]$.

In a manner similar to the reduction method described in equation (24) of the tangent subsection, the argument can be reduced to within the interval $[0,1]$. The identity function depicted in (29) is valid for those arguments greater than or equal to one. This same identity function can be used to reduce the argument to within the interval $[0, \pi/4]$. This is performed by subtracting from $\pi/2$, the arctangent of the inverse for the reduced argument.

$$\text{Atan}(\alpha) = \pi/2 - \text{Atan}(1/\alpha) \quad (29)$$

The final range reduction reduces the argument to within the interval $[0, 2 - \sqrt{3}]$. This is done by making use of the identity functions shown in equations (30) and (31).

$$\beta = (\alpha \sqrt{3} - 1) / (\sqrt{3} + \alpha) \quad (30)$$

$$\text{Atan}(\alpha) = \pi/6 + \text{Atan}(\beta) \quad (31)$$

III. Development and Design of the Functions

General Discussion

This chapter deals with the detailed design of each of the specific functions. As was stated earlier, two type representations have been designed and implemented. Therefore, each of the following sections contain a discussion on the design of each function's fixed-point and floating-point implementation.

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. In addition, fixed-point algorithms were designed to be invoked with arguments expressed in pi-radian measure. This metric was discussed in the "Assumptions" section of chapter one.

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.

Sin and Cos Implementation

Fixed-Point. In Chapter two, it was noted that the identity function for "Cos", equation (19), could be used to design a procedure that will compute the values for both the sine and cosine functions. Indeed, if one looks at Figure 3, this concept can be verified. For example: examine the graph of cosine and note that $\text{Cos}(-\pi)$ is equivalent to $\text{Sin}(|-\pi| + \pi/2)$. Therefore, by adjusting the cosine's argument α , to its absolute value, plus $\pi/2$, one routine can be developed that will compute the values of both functions.

JOVIAL, the high order language used to implement these functions, doesn't have the facility for supporting multiple entry points. As a result, both "Sin" and "Cos" have been implemented as shown in figures 7a, 7b, and 8. Both functions have their own unique entry points, but both call the function "SinCos" to compute the desired results.

Note that steps 2, 3, and 4 of Figure 7a, represent the identity function for "Cos". The fixed-point algorithms use pi-radian measures (pi-radians = radians/ π); therefore, one half is used in lieu of $\pi/2$.

Since an argument's legal range of values for this fixed-point implementation is $[-1, 1]$; these functions have been defined as having one sign bit, one integer bit, and 30 fractional bits ($-2 < x < 2$). The JOVIAL compiler will not let programmers invoke this routine with an incompatible type; however, it is still possible for "Cos" to generate a value outside the function's defined range. This could potentially cause a problem on another machine, but not on a 1750A. Adding .5

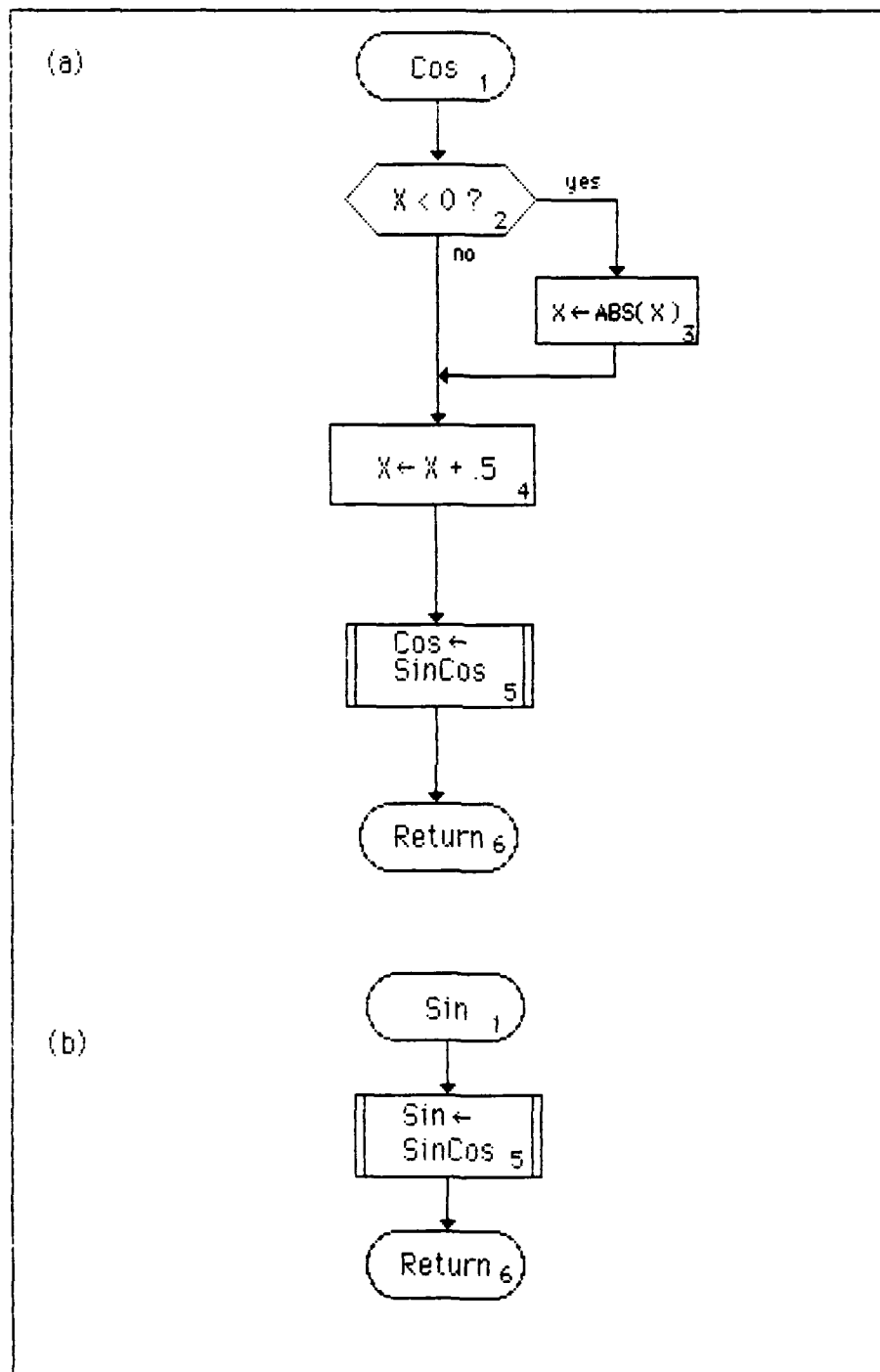


Figure 7 a) Cos; b) Sin Structured Flowchart

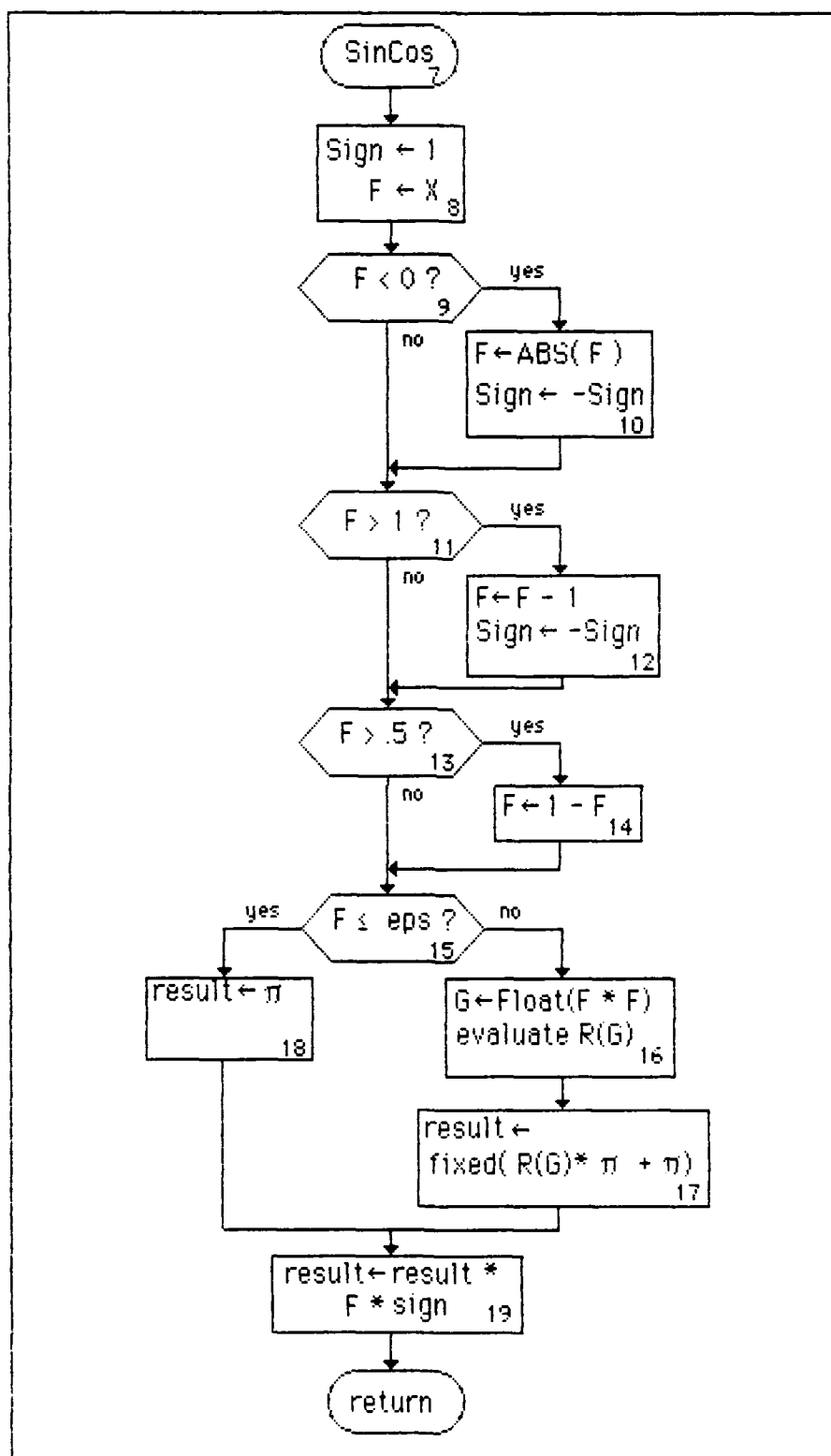


Figure 8 SinCos Fixed-Point Structured Flow Chart

pi-radians to the argument whose absolute value (through programmer error) is already within the interval $[1.5, 2)$, will cause an overflow. However, the MIL-STD-1750A specifies that fixed-point overflows will result in the most-significant-bits being lost, and the least significant retained. This feature of the 1750A architecture is actually a benefit to this design implementation. The cosine function has a period of 2π radians (2 in pi-radian measure), and since an overflow causes the result to be a modulo of the period length, the final result is in its reduced form.

As stated earlier, the legal range of values for these functions is $[-1, 1]$. In addition to the aforementioned situation, it is possible that an adjustment to the cosine argument would result in its being outside the legal interval yet not overflow. No further reduction is performed; however, as the same potential problem exists in programs that invoke the "Sin" function. Therefore, any further reductions are handled in "SinCos".

The variable "Sign" shown in step 8 of the "SinCos" function (Figure 8), is what is used to note the sign changes discussed on pages 26 and 27. Since the initial step of this algorithm is to reduce the original argument to within the interval $[0, .5]$, the approximating polynomial will always compute a positive result. Therefore, during each phase of the domain reduction process, it is necessary to keep track of those arguments whose functional values are, in reality, antisymmetric with the computed result. The value returned to the calling procedure will be the result of the polynomial approximation multiplied by the value of "Sign" (± 1).

The variable "X" is the argument passed to the "SinCos" function. Since domain reduction on the argument is required, and since JOVIAL

treats formal parameters as read only, the value of "X" is assigned to "F". The variable "F" is used throughout the domain reduction phase. (step 8)

The first step of the domain reduction process is to restrict "F" to within the interval $[0, 2)$. If "F" is negative, it is antisymmetric with respect to, its corresponding positive value. Examine Figure 3a and note that $\sin(\alpha)$ equals $-\sin(-\alpha)$; therefore, the sign change is noted ($\text{Sign} = -\text{Sign}$), and "F" is set to its absolute value. The built-in function for absolute value was found to give inconsistent results, so the absolute value is found by: $F = -F$. (steps 9 and 10 of Figure 8)

The second step of the domain reduction phase is to reduce the argument to within the interval $[0, 1]$. Examine Figure 3a, and note that the interval $[1, 2]$ (in pi-radians) is antisymmetric, with respect to, the interval $[0, 1]$. Again, the sign difference must be noted ($\text{Sign} = -\text{Sign}$), and the argument reduced to: $F = 1 - F$. (steps 11 and 12 of Figure 8)

The final phase of the domain reduction step, places the argument, "F", within the interval $[0, .5]$. This reduction is accomplished by exploiting the sine function's property of symmetry. Examine Figure 3a again, and note that the interval $[0, 1]$ is symmetric about the point one-half. That is, $\sin(0) = \sin(1)$, $\sin(.2) = \sin(.8)$, . . . , etc.. Stated another way, if "F" is greater than .5; then $\sin(F)$ can be computed using $\sin(1-F)$. Therefore, "F" is reduced by setting F to $1 - F$. (steps 13 and 14 of Figure 8)

The next step of the algorithm is to compute the result of a polynomial approximation for $\sin(F)$. This polynomial is a "Chebyshev Economization" of the Maclaurin series for sine. However, if the reduced argument is so small that its use within the approximation would cause an

underflow (i.e. "F" is less than some epsilon "eps"), the approximation is set to $\pi * F$. In systems using a radian measure -- $\sin(F)$ approaches "F" as "F" approaches zero. Therefore, in systems using pi-radian measure -- $\sin(F)$ approaches $F * \pi$ as "F" approaches zero. The lower limit is set to the value that prevents the computation $r_1 * F^3$ from underflowing (r_1 is a coefficient from the approximating polynomial). As with an overflow condition, the 1750A handles underflows without interrupting software processing, but sets the result to zero. Underflow may not create a problem, but the check for underflow may eliminate the overhead associated with computing the polynomial approximation. (steps 15 and 18 of Figure 8)

The coefficients used in this algorithm are modifications of those presented by Cody and Waite (4: 132). The approximation described by Cody and Waite is similar to the form shown in (32). The "r" values represent the coefficients of the approximation, and $\pi\theta$ represents the angle being approximated. These modifications were necessary because the algorithms of Cody and Waite use arguments expressed in radian measure (i.e. $\pi\theta$), rather than pi-radians as in this implementation.

$$P(\pi\theta) = r_5(\pi\theta)^{11} + r_4(\pi\theta)^9 + r_3(\pi\theta)^7 + r_2(\pi\theta)^5 + r_1(\pi\theta)^3 + (\pi\theta) \quad (32)$$

By restructuring (32) into the form of that shown in (33), the new coefficients can be easily computed. From (32), (33), and the definition for pi-radians; it can be shown that the new coefficients, r' , can be computed by: $r'_n = r_n \pi^y$. Where "y", is the power of the angle expressed by the radian metric -- $r_n(\pi\theta)^y$ in (34).

$$P(\pi\theta) = \pi^{11} r_5 \theta^{11} + \pi^9 r_4 \theta^9 + \pi^7 r_3 \theta^7 + \pi^5 r_2 \theta^5 + \pi^3 r_1 \theta^3 + \pi\theta \quad (33)$$

A cursory examination of this method of approximation shows, that it would be very inefficient if the algorithm were coded exactly as shown (32 or 33). This problem is eliminated by making use of "Horner's Rule". If "G" is set to the value " π^2 ", the polynomial can be computed as shown in (34) and (35). (steps 16 and 17 of Figure 8)

$$\text{result} = ((((((r_5 G + r_4) G + r_3) G + r_2) G + r_1) G \quad (34)$$

$$\text{result} = \text{result} \pi + \pi \quad (35)$$

This last method of approximation requires eight multiplications and five additions; as compared to the five multiplications, five additions, and five powers shown in (32). The significant difference in the number of arithmetic operations, will also prevent the loss of precision that accompanies the computation of the individual powers of " π ".

	r - Cody & Waite	Power of π	new r'
r_1	-0.16666_66560_883 E+0	π^2	-1.64493_40611_400 E+0
r_2	0.83333_30720_556 E-2	π^4	0.81174_21707_751 E+0
r_3	-0.19840_83282_313 E-3	π^6	-0.19074_76226_769 E+0
r_4	0.27523_97106_775 E-5	π^8	0.02611_62053_162 E+0
r_5	-0.23868_34640_601 E-7	π^{10}	-2.23522_40374_060 E-3

Table 3 Coefficients for Polynomial approximation to Sin

The coefficients for this method of approximation were determined from the relationship, $r'_n = r_n \cdot \pi^n$, and the powers of π implicit to (34). The new values were computed using the coefficients given by Cody and Waite, the value of π in (36), and a machine of higher precision than that of the 1750A. The new coefficients are shown in Table 3.

$$\pi = 3.14159_26535_90 \quad (36)$$

Floating-Point. The method used in approximating the floating-point sine and cosine functions, is very similar to that just described. So, there will be references to the discussions of the preceding subsection. The major difference between the previous algorithm and this one is, that this algorithm accepts arguments expressed in the conventional radian measure.

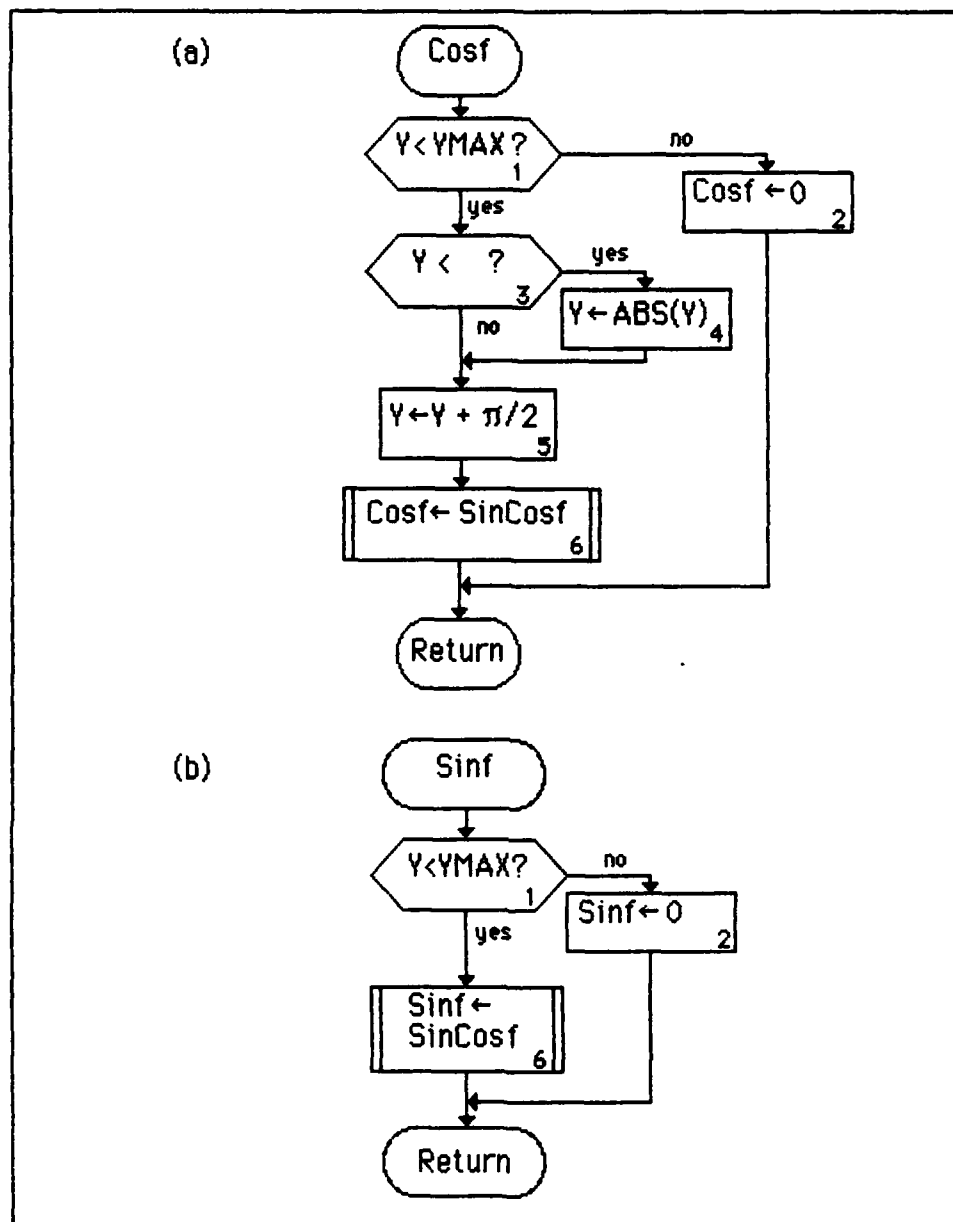


Figure 9 a) Sinf; b) Cosf Structured Flowchart

Even though these algorithms accept arguments of radian measure, they are still quite different from the algorithms described by Cody and Waite. Their algorithm converts the argument to pi-radians,

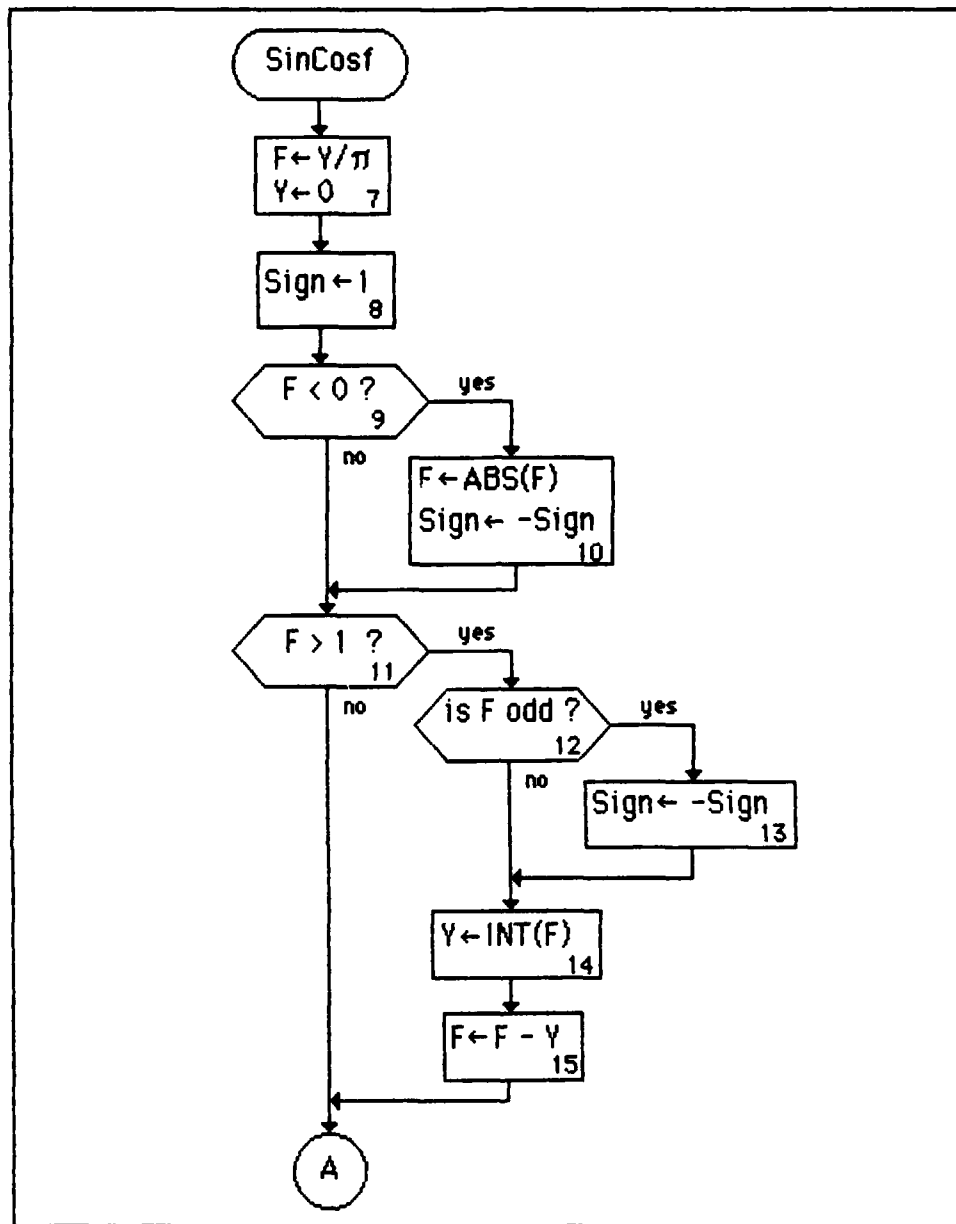


Figure 10a SinCosf Entry Point Structured Flowchart

makes some adjustments, and then converts the argument back into radians. This takes time, unnecessarily introduces the possibility of overflow conditions, and forces the algorithm to eliminate errors

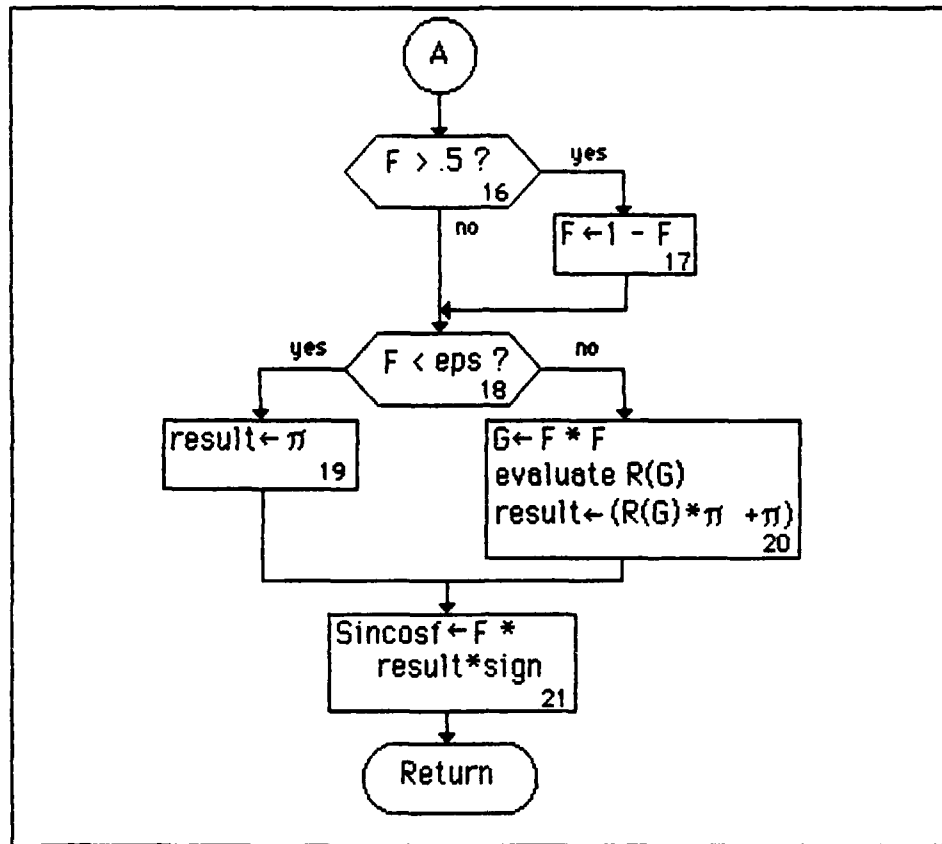


Figure 10b SinCosf Exit Point Structured Flowchart

generated through the use of guard digits within the architecture. Therefore, since the argument has to be converted to pi-radians anyway, the new design leaves it in that unit of measure throughout the procedure.

As was discussed in the previous subsection, JOVIAL cannot support multiple entry points, and as a consequence, both "sinf" and "Cosf" are little more than linkages to the routine "SinCosf" (see Figure 9).

Note the naming convention used for these functions. They are distinguished from their fixed-point counterparts by the letter "f" at the end of their name. This convention is used for all functions implemented as a result of this thesis, and will not be elaborated elsewhere.

Prior to invoking the function "SinCosf", both "Sinf" and "Cosf" must perform a limit check on their passed argument. This step is necessary to insure a good approximation for large arguments. Due to the way floating-point values are stored, least significant bits are lost as numbers become larger. It is possible for a number to be so large, that it is no longer representable as a reasonable multiple of pi. Therefore, the maximum limit used in these algorithms is similar to that recommended by Cody and Waite (4: 134). The maximum size an argument can take is, the integer value $\pi * 2^{t/2}$ (where "t" is the number of non-sign bits of the floating-point coefficient). If the argument is greater than the maximum limit, each of the respective functions will return a value of zero. This is an area in need of further analysis, and should lead to an acceptable method for handling exceptions within this avionics weapon systems. (steps 1 and 2 of figures 9a and 9b)

Before "Cosf" can invoke the function "Sincosf", it must make use of the identity function described in equation (19). Since the floating-point argument is still in radian measure at this point, $\pi/2$ is added to the absolute value of the variable "Y". (steps 3 - 5)

The first step of the "SinCosf" function is to initialize the variables that will be used during the domain reduction phases. The working variable "F", will contain the eventual reduced argument. Dividing "F" by π initializes it to a multiple of pi-radians. In the actual implementation, "F"

is multiplied by the constant in (37) rather than divided by π . This method is preferred, because multiplications are more efficient than division in most architectures. (step 7 Figure 10a)

$$1/\pi = 0.31830_98861_838 \quad (37)$$

The variable "Y", initialized in step 7, will be used to find the integer portion of "F". Since JOVIAL doesn't include the built-in function "INT" for extracting the integer portion of a floating-point number, other features of the language are used to perform the equivalent action. This method is discussed in Appendix A; however, at this point the variable must be initialized to zero.

As in the fixed-point algorithm described in the preceding subsection, it is necessary to note any reduction process that has an affect on the sign of the final result. Therefore, the "Sign" flag must be initialized to one. (step 8 Figure 10a)

The first phase of range reduction, is to reduce the argument to within the range of all positive multiples π . If "F" is negative, it is antisymmetric with its positive counterpart; therefore, after setting "F" to its absolute value, the sign change must be noted ($\text{Sign} = -\text{Sign}$). (steps 9-10 Figure 10a)

The next domain reduction phase reduces the argument to within the interval $[0, 1]$, and is represented by steps 11 through 14 of the "SinCos" flowchart. If the argument "F", is greater than or equal to one, it has an integer portion defined within it. If the integer portion of "F" is

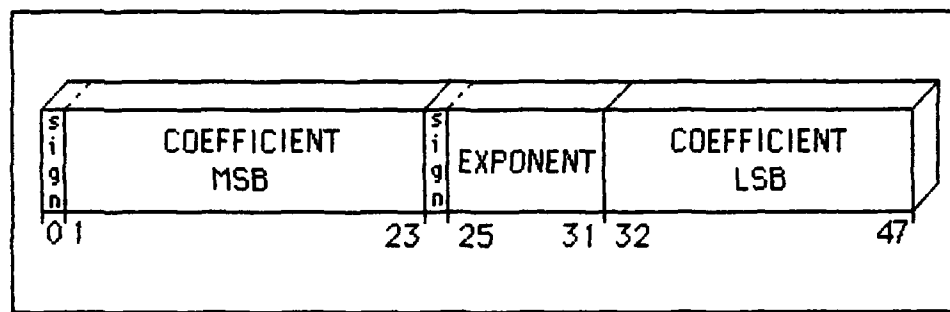


Figure 11 Bit Layout of 1750A Floating-Point Number

odd, then the argument lies in an interval that is antisymmetric, with respect to, those arguments that lie within the interval $[0, 1]$. If it is antisymmetric, the sign difference must be noted before proceeding.

The description of the ODD function (step 12 Figure 10a) in Appendix A makes it sound as though the process is rather involved. To the contrary, it is very simple and fast as compared to the method described by Cody and Waite. This function can be implemented by using JOVIAL specified tables that take advantage of the floating-point bit pattern shown in Figure 11.

In a manner similar to that described for the ODD function, the "INT" function can be simulated. If the exponent points to the least significant bit of a floating-point number's integer field, then bits zero through the least significant bit, contain the actual integer. The integer bits of the argument "F" are copied into the equivalent bit positions of the variable "Y", and then the exponent field of "Y" is set to the value of the exponent field contained in "F". Subtracting "Y" from "F" will reduce the argument "F" to within the interval $[0, 1]$. (steps 14 and 15, Figure 10a)

	r - Cody & Waite	Power of π	new r'
r_1	-0.16666_66666_666 E+0	π^2	-1.64493_40668_480 E+0
r_2	+0.13333_33333_276 E-2	π^4	0.81174_24252_778 E+0
r_3	-0.19841_26982_322 E-3	π^6	-0.19075_18239_486 E+0
r_4	+0.27557_31642_129 E-5	π^8	2.61478_45158_310 E-2
r_5	-0.25051_87088_347 E-7	π^{10}	-2.34605_87938_600 E-3
r_6	+0.16047_84463_238 E-9	π^{12}	1.48325_28223_590 E-4
r_7	-0.73706_62775_071 E-12	π^{14}	-6.72364_47557_180 E-6

Table 4 Coefficients for Polynomial approximation to Sinf

The last step of the range reduction phase is to place the argument to within the interval $[0, .5]$. This step is identical to that described in the fixed-point algorithm, and involves comparing "F" to one-half. If "F" is greater than one-half the "F" is set to: $1 - F$. (step 17, Figure 10b)

The next two steps of the algorithm are: to approximate the function $\sin(F)$; and then adjust it by multiplying it by "F" and "Sign". These steps are identical to those described in the previous subsection, and need not be expanded again. Even though the coefficients used in the approximating polynomial are different from those used by the fixed-point algorithm, they were derived in the same manner as described before. These values are displayed in Table 4.

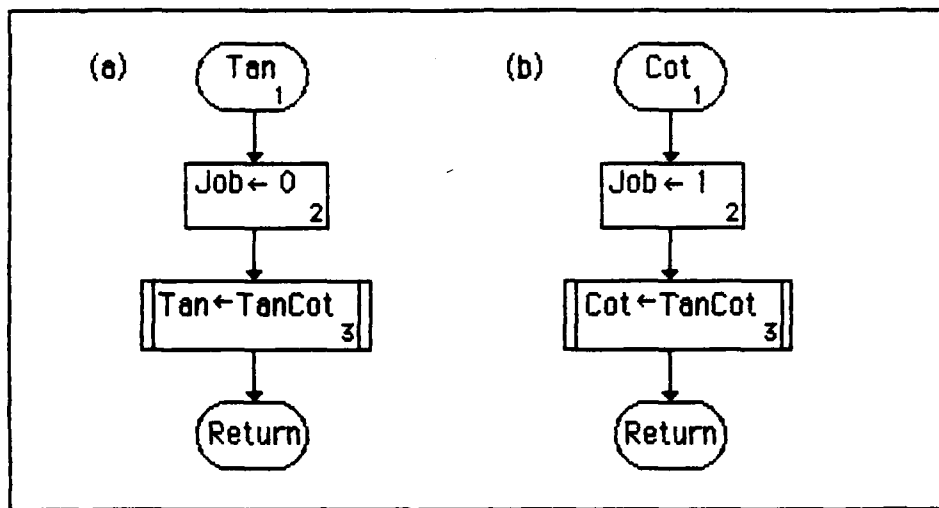


Figure 12 a) Tan; b) Cot Structured Flowchart

Tan and Cot Implementation

Fixed-Point. In chapter two, the identity function for "Tan" and "Cot", equation (23), established that a procedure could compute approximations for both the tangent and cotangent functions. A cursory examination of Figure 28 will verify this. Therefore, by noting which function is invoked, a single procedure can be called to do both computations.

Since JOVIAL doesn't have the facility for supporting multiple entry-points, these functions have been implemented as shown in figures 12a, 12b, and 13. Both "Tan" and "Cot" have their own unique entry-points, but both call "TanCot" to compute the desired results.

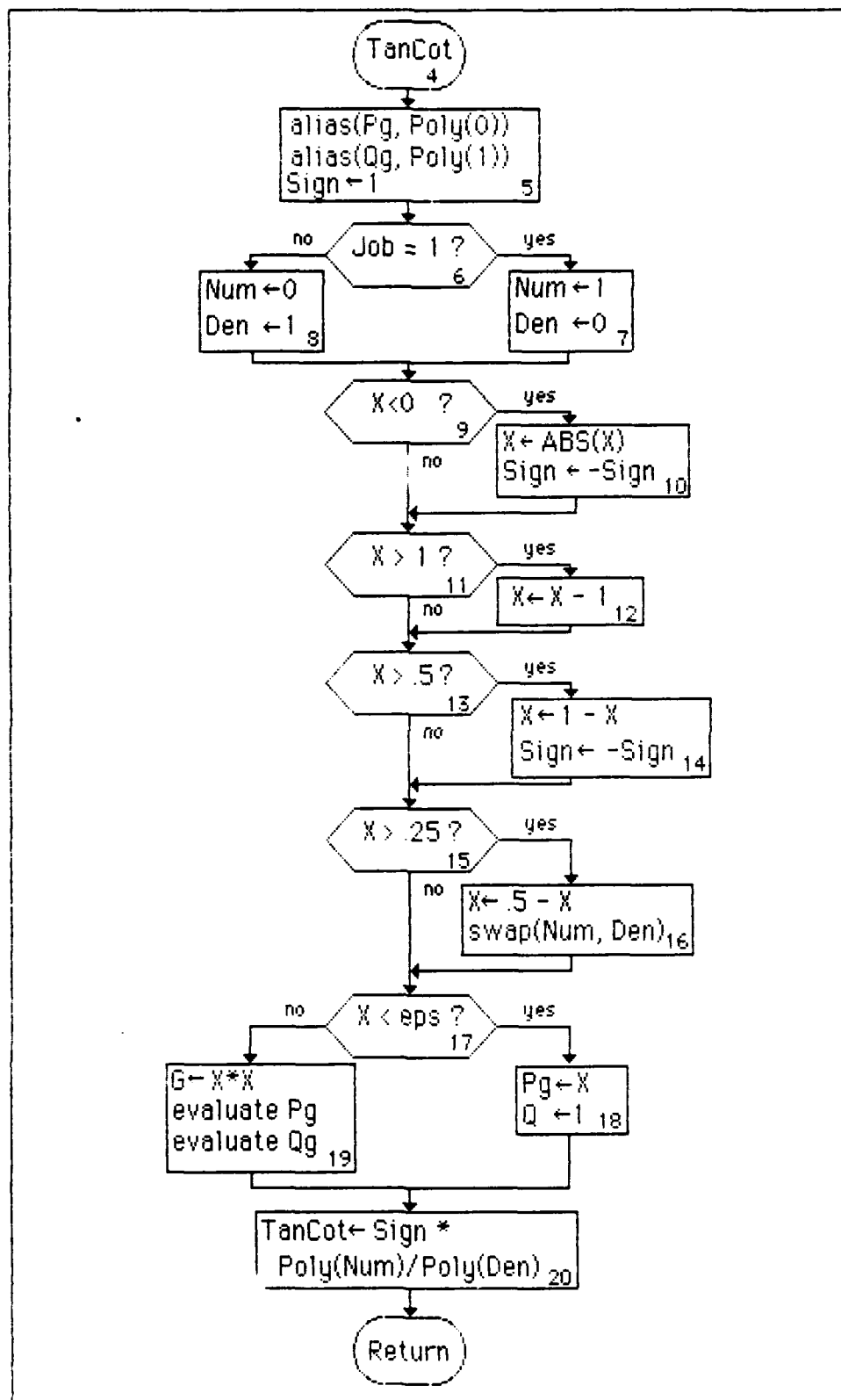


Figure 13 TanCot Structured Flowchart

Note that the "Cot" function does not implement the identity function, as do the "Cos" or "Cosf" functions. The approximation method used, is the "Remes" algorithm discussed in the preceding chapter. Since this method is a rational approximation that uses a polynomial evaluation for the numerator, and another for the denominator; the identity function can be implemented within "TanCot". "TanCot" must choose which polynomial evaluations are to be used as the numerator, and as the denominator. This decision is based on the value of "Job", a flag passed to it by "Tan" and "Cot". (step 2, figures 12 and 12b) which function has been invoked, one routine can be developed that will compute the values of both functions.

Step 5, of Figure 13, is not a computer operation. However, it is an efficient means of implementing the "Tan" and "Cot" identity function, and one of the range reduction phases. The rational approximation is of the form shown in equation (10); where the polynomial used in the numerator is represented by P_g , and the polynomial used in the denominator is represented by Q_g . Each polynomial's evaluation is contained in a table called "Poly": P_g is in entry zero, and Q_g is contained in entry one. The "alias" pseudo-operation, is equivalent to a JOVIAL "Define". It is a note to the compiler, that there are two methods of referring to each table entry. The reason for this may not be intuitively obvious at this point, but will become clear later.

As in every function discussed previously, it is necessary to note any domain reduction processes that have an impact on the sign of the final result. Since the final domain reduction phase of this algorithm reduces an argument to within the pi-radian interval $[0, .25]$, computed results are always positive. Therefore, the "Sign" flag is initialized to indicate a

positive result. If, at any point during the domain reduction step, it is determined that the true functional value is antisymmetric with what is currently indicated, the flag is negated (i.e. $\text{Sign} = -\text{Sign}$). (step 5, Figure 13)

Steps 6, 7, and 8 of the TanCot flowchart, are where the initial numerator and denominator are determined. If "TanCot" was called by "Cot" the "Num" flag (indicates which table entry of "Poly" is the numerator) is set to one, and its counterpart, "Den" is set to zero. This is where the use of an alias may start making sense. Even though this function could have been coded without using aliases; it is more convenient, and more readable, if the design doesn't have to be concerned with "which" entry contains "what" polynomial evaluation. All that is necessary, is to keep track of how each will be used: either as a numerator or as a denominator. This is how the identity function has been implemented. In the event that "TanCot" is called by "Tan", "Num" and "Den" are set to zero and one, respectively.

The next step of this algorithm is domain reduction, and the first phase of this step is to limit the argument to all positive values of pi-radians. If the argument "X" is negative, it is antisymmetric with its positive counterpart. The variable "X" is set to its absolute value, and the sign dichotomy is noted ($\text{Sign} = -\text{Sign}$). (steps 9 and 10, 13)

As was the case for "Sin" and "Cos", "Tan" and "Cot" accept arguments expressed in pi-radian measure. The legal range of values accepted by these routines is $[-1, 1]$, and their argument's fixed-point attributes are the same as those defined in the "Sin" and "Cos" functions. Consequently,

MIL-STD-1750A architectures will allow the value of arguments to lie within the the interval $(-2, 2)$. After the aforementioned domain reduction phase, arguments will be limited to the interval $[0, 2)$. Since the architecture will allow arguments outside the function's defined interval, it is necessary to consider those exceptions during domain reduction. Examine Figure 4a, and note that the period of tangent is one pi-radian. The expression, $\text{Tan}(X) = \text{Tan}(X-1)$, is a true equality, and illustrates that function calls with an argument in the interval $(1, 2)$, can still be approximated if their argument is reduced by one. (steps 11 and 12, Figure 13)

The next domain reduction phase will reduce the argument to within the interval $[0, .5]$. Examine Figure 4a, and note that those arguments falling in the interval $[.5, 1]$ are antisymmetric, with respect to, those lying in the interval $[0, .5]$. That is, if $X > .5$, then $\text{Tan}(X) = -\text{Tan}(1-X)$. As an example, assume that $X = .75$. Examine Figure 4a, and note that $\text{Tan}(.75) = -1$. Also note that, $\text{Tan}(1-.75) = \text{Tan}(.25) = 1 = -\text{Tan}(.75)$. This example illustrates that, the argument can be reduced if the sign change is noted, and the argument set to: $X = 1-X$. (steps 13 and 14)

The final phase of domain reduction, reduces the argument to within the interval $[0, .25]$. Again, examine Figure 4a. Note that all arguments lying in the interval $[.25, .5]$, are inversely related to those arguments in the interval $[0, .25]$. That is, if $X > .25$, then $\text{Tan}(X) = 1/\text{Tan}(.5-X)$. Further argument reduction is accomplished by swapping the values of the "Poly" subscripts, "Num" and "Den", and setting the variable "X" to $X = .5-X$. (steps 15 and 16, Figure 13)

As stated earlier, fixed-point underflow and overflow will not create a problem for software packages running on a 1750A architecture. However, if the argument is not examined for such conditions, there will be an unnecessary amount of overhead incurred while obtaining an approximation to these functions. Step 17 of the "TanCot" flowchart is intended to eliminate the unwarranted overhead. If the argument, "X", is less than some epsilon, the approximation is found by setting the polynomial "Pg" to $X*\pi$, and the polynomial "Qg" to one. Otherwise, each polynomial must be evaluated separately. The epsilon used during implementation, is the inverse of MAX/π ; where "MAX" is the maximum representable value allowed by the attributes of the returned function. The attributes of the returned function are: one signed bit, 12 integer bits, and 18 fractional bits (steps 17 and 18, Figure 13)

The process of evaluating the two polynomials, "Pg" and "Qg", is similar to that of $P(x)$, in the "Sin" and "Cos" implementation. Each polynomial was derived through the use of the "Second Algorithm of Remes", and each are of the form shown in (38) and (39), respectively. Coding the evaluations in the manner implied by (38) and (39), is very inefficient. This inefficiency, is eliminated through the use of "Horner's" rule.

$$Pg = P_2(\pi\theta)^5 + P_1(\pi\theta)^3 + P_0(\pi\theta) \quad (38)$$

$$Qg = Q_2(\pi\theta)^4 + Q_1(\pi\theta)^2 + Q_0 \quad (39)$$

	Cody & Waite	Power of π	Pi-Radian Coeff
P_0	0.10000_00000_000 E+0	π	3.14159_265358980 E+0
P_1	-0.11135_14403_566 E+0	π^2	-0.10990_93361_855 E+1
P_2	0.10751_54738_488 E-2	π^4	0.10472_98457_970 E+0
Q_0	0.10000_00000_000 E+1	π^0	0.10000_00000_000 E+1
Q_1	-0.44469_47720_281 E+0	π^2	-0.43889_61479_150 E+1
Q_2	0.15973_39213_300 E-1	π^4	0.15559_53608_405 E+1

Table 5 Coefficients for Polynomial approximation to Tan

Application of "Horner's" rule, will leave the polynomials in the form shown in equations (40) and (41) .

$$Pg = \{ [P_2 (\pi\theta)^2 + P_1] (\pi\theta)^2 + P_0 \} (\pi\theta) \quad (40)$$

$$Qg = [Q_2 (\pi\theta)^2 + Q_1] (\pi\theta)^2 + Q_0 \quad (41)$$

These equations use the radian metric, $\pi\theta$. The subscripted values of "P" and "Q", represent the coefficients determined by Cody and Waite (4: 162). If (40) and (41) are restructured as in (42) and (43); the angle, θ , is a pi-radian metric. The radian coefficients, multiplied by their

associated power of pi, determine new pi-radian coefficients. The new coefficients are given in Table 5. (step19, Figure 13)

$$Pg = \{ [(P_2 \pi^4) \theta^2 + P_1 \pi^2] \pi \theta^2 + P_1 \pi \} \theta \quad (40)$$

$$Qg = [(Q_2 \pi^4) \theta^2 + Q_1 \pi^2] \theta^2 + Q_1 \quad (41)$$

The evaluation of $\text{Tan}(X)$ is the final step of this algorithm. The result of "Pg" is stored in "Poly(0)", and the result of "Qg" is stored in "Poly(1)". The evaluation to be used as the numerator, as well as the one to be used as the denominator, is determined by the values in "Num" and "Den". The value returned is the result of the polynomial division, multiplied by "Sign". (step 20, Figure 13)

Floating-Point. In chapter two, it was noted that the identity function for "Tan" and "Cot", equation (23), could be used to design a procedure that will compute the values for both the tangent and cotangent functions. A cursory examination of Figure 28 will verify this. Therefore, by noting which function has been invoked, one routine can be developed that will compute the values of both functions.

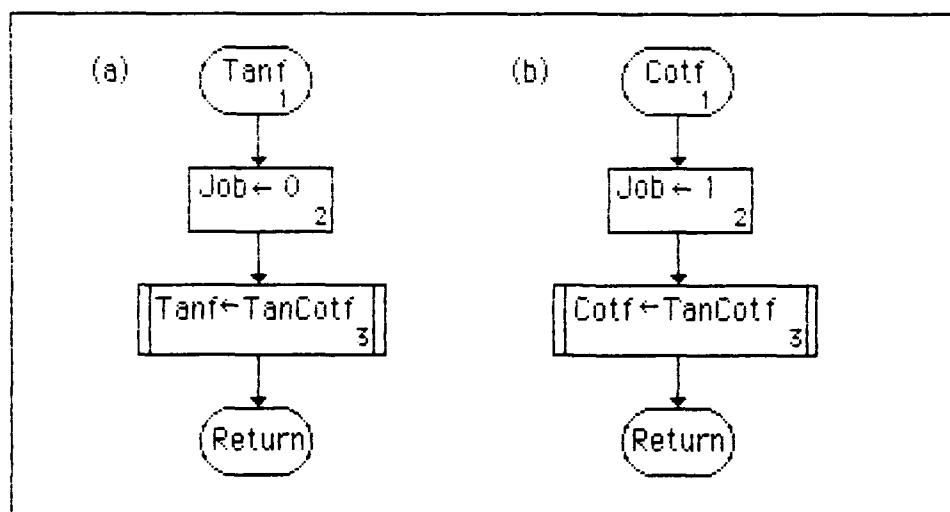


Figure 14 a) Tanf; b) Cotf Structured Flowchart

Both "Tanf" and "Cotf" are logically identical to "Tan" and "Cot". (see figures 12 and 14) The only difference between the two type implementations, is that the floating-point algorithms are invoked with arguments expressed in radian measure.

Step 5 of "TanCotf" is similar to step 5 of Figure 13. The only difference is that the argument, "X", is divided by $\pi/2$. As mentioned earlier, multiplications are typically more efficient than divisions; so the division is implemented as the product of the argument and the constant in (42).

$$\pi/2 = 1.57079_63267_95 \quad (42)$$

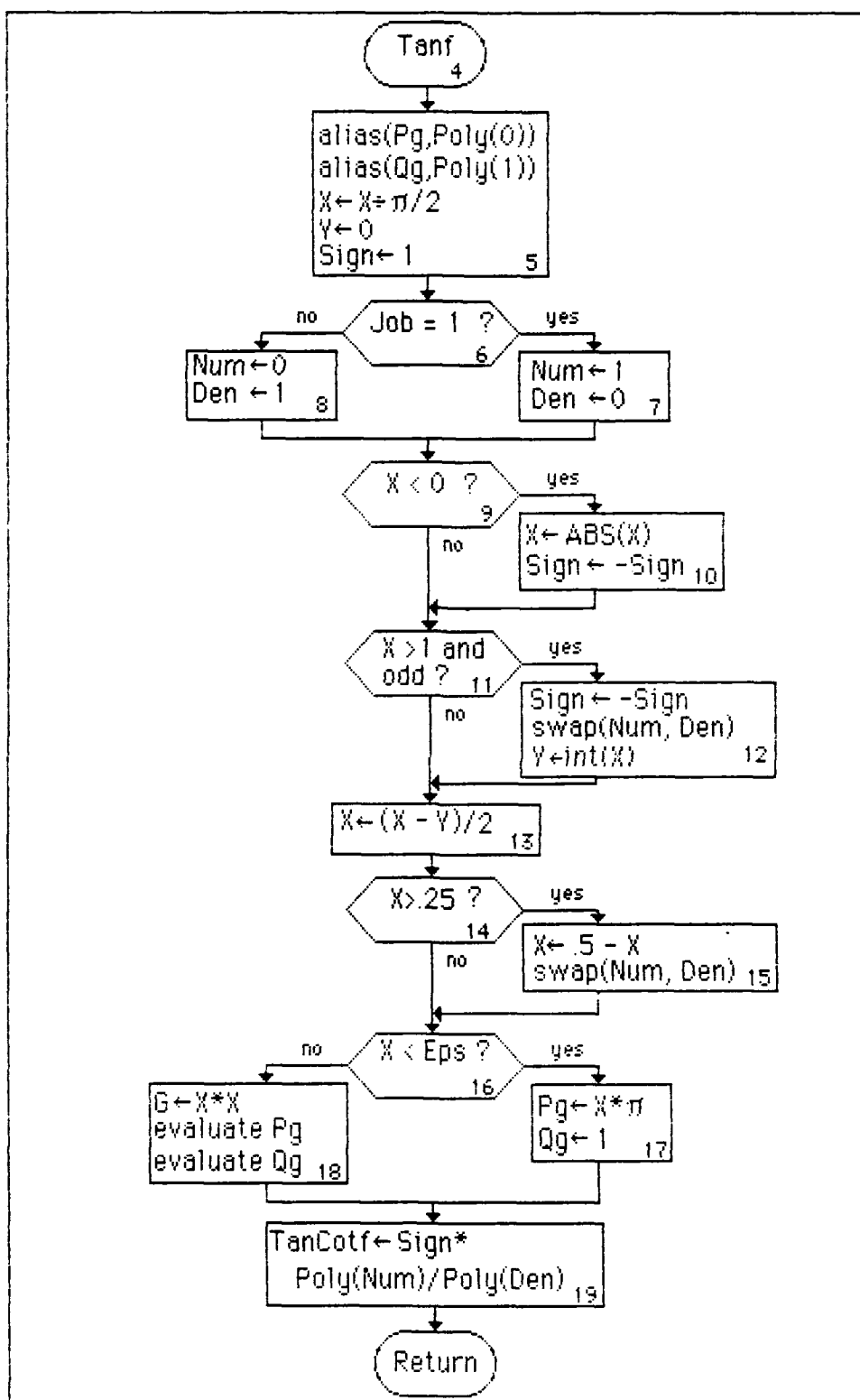


Figure 15 TanCotf Structured Flowchart

Floating-Point Multiply		Exponent Adjustment	
OPERATION	EXECUTION TIME in μ sec	OPERATION	EXECUTION TIME in μ sec
EFL	4.00	LR	0.56
EM	8.96	LB	1.57
EFST	4.10	SLL	0.81*
		SRA	0.81*
		AR	0.55
		STLB	2.30
Totals	17.06 μ sec		6.60 μ sec
* Times not determined, worst case times from similar operations used.			

Table 6 Execution Times for 1750A Instructions. From
Sperry 1631 Programmer Reference Manual

Steps 6 through 9, of Figure 15, are identical to the same numbered steps in Figure 13. The logic, for both cases, is identical, and won't be expanded any further.

The division performed in step 5 of this algorithm, expresses the argument "X" as a multiple of $\pi/2$. Examine Figure 4a and note that every other multiple of $\pi/2$, is the negative inverse of the interval $[0, \pi/2]$. Therefore, if $X \geq 1$ and its integer portion is odd; the sign dichotomy is noted, and the table subscripts for "Poly" are swapped. The subscript swap, changes which polynomials will be used as the numerator and denominator of this rational approximation. (step 11 and 12, Figure 15)

Step 13 of this algorithm, is used to express the argument in pi-radian measure (i.e. $X * 2/\pi * 1/2 = X/\pi = \text{pi-radians}$). Rather than dividing by two or multiplying by one-half, a faster and more accurate method is used. The layout of a 1750A floating-point number was presented in the "SinCosf"

	Cody & Waite	Power of π	Pi-Radian Coeff
P_0	0.10000_00000_000 E+1	π	3.14159_265358980 E+0
P_1	-0.12828_34704_096 E+0	π^2	-0.12661_07104_141 E+1
P_2	+0.28059_18241_170 E-2	π^4	0.27332_19453_881 E+0
P_3	-0.74836_34966_612 E-5	π^6	-0.71946_85785_563 E-2
Q_0	0.10000_00000_000 E+1	π^0	0.10000_00000_000 E+1
Q_1	-0.46161_68037_429 E+0	π^2	-0.45559_75237_838 E+1
Q_2	+0.23344_85282_207 E-1	π^4	0.22740_00893_720 E+1
Q_3	-0.20844_80442_204 E-3	π^6	-0.20039_96971_354 E+0

Table 7 Coefficients for Polynomial approximation to Tanf

section. This knowledge can be used to make multiplications, involving powers-of-two, more efficient. Remember, the exponent field of a floating-point number is expressed as a power-of-two; therefore, a division by two can be accomplished by subtracting 1 from the exponent field. The subtraction can be made through the use of JOVIAL specified tables, containing a signed integer field that overlays the exponent of "X".

Table 6 compares the execution times of the "Exponent Adjustment" method just described, against floating-point multiplies. It's interesting to note, that the implemented method is almost three times faster than a straight multiply.

The logic for the rest of the design is identical to the last six steps of of the fixed-point algorithm, and needs no further explanation. Two implementation exceptions are: the values of epsilon, and the coefficients used in the approximations. Epsilon is set to a value; such that P_g/Q_g , and its inverse, will not cause an overflow, or an underflow. The coefficients,

for this approximation, were found in the same manner as described in the "TanCot" subsection, and are summarized in Table 7.

ASin and ACos Implementation

Up to this point, all discussions of the detailed design section have been divided into two subsections: one concerned with fixed-point implementation, and a second concerned with floating-point implementation. The last two sections of this chapter discuss the inverse trigonometric functions, and their implementation. The fixed-point and floating-point implementations have arguments that are in the same metric, and ;therefore, do not differ significantly in their design. With but two exceptions, the only difference between the two type "ASin" and "ACos" functions are the coefficients used in their approximations. For that reason, there is only one design discussion. The lone exceptions will be noted later.

The identity function that relates "ACos" to "ASin" is shown in equation (26), and is expanded upon in (27) and (28) . These equations illustrate the point that one procedure can be written to compute the approximations of both functions. The routine "ASinCos" , will return the appropriate value, depending on which function invoked it. As mentioned several times prior to this, JOVIAL does not support multiple

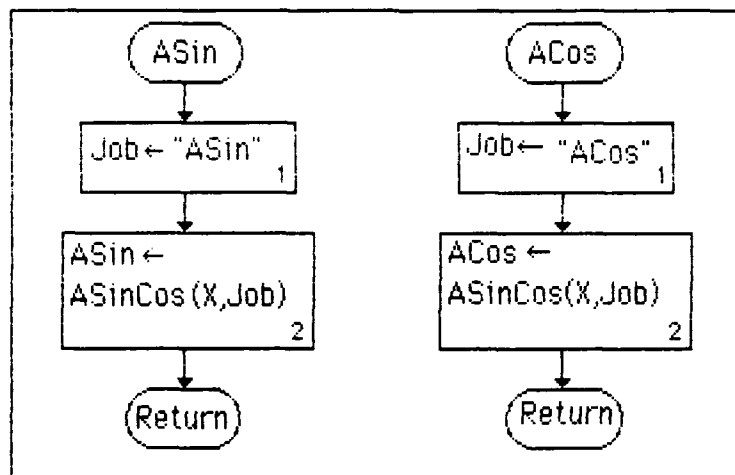


Figure 16 ASin and ACos Structured Flowchart

entry-points, and forces the two functions to be implemented as shown in figures 16 and 17.

Each function invokes the procedure "ASinCos" by passing the argument, and a flag indicating which function is making the call.

The function approximation of "ASinCos" is performed in three steps: reduction of the argument "Y" to within the interval $[0, .5]$; the evaluation of $\sin^{-1}(Y)$; and reconstruction of $\sin^{-1}(Y)$ to the representation of the calling function $\sin^{-1}(X)$.

The first step of argument reduction is performed by step 3 in the flowchart of Figure 17. The sign change is not noted, because the final reduction is not a multiple of ± 1 .

The computation of $\sin^{-1}(Y)$ is sensitive to error for large arguments, especially for those that are close to 1. Therefore, careful argument reduction is required to limit this problem. Steps 5 through 8 are designed to do just that, and the reader is encouraged to reference

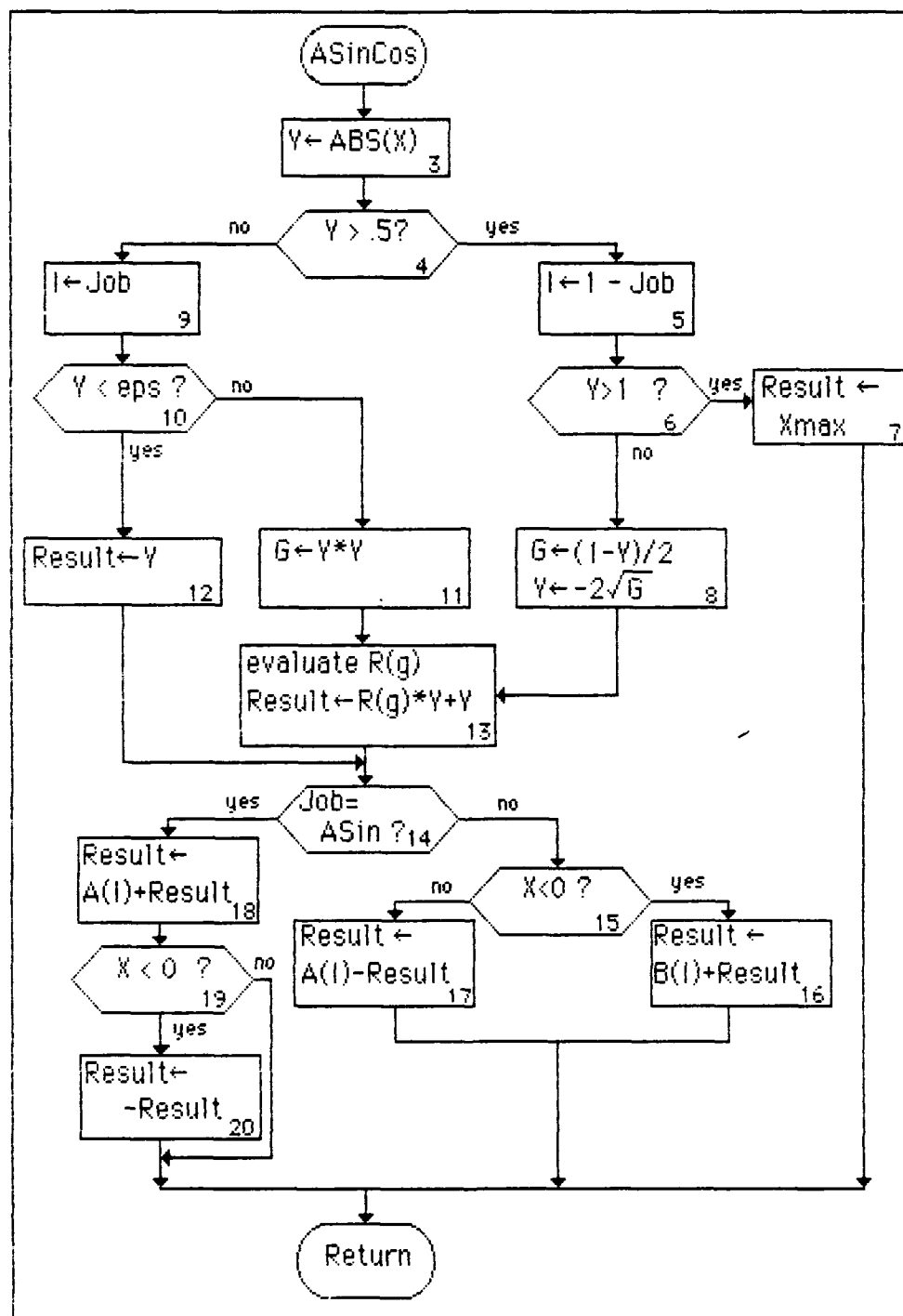


Figure 17 ASinCos Structured Flowchart

Cody and Waite (4: 174) for a detailed discussion on the benefits of this design consideration.

If the argument, reduced by step 3, is greater than .5, then a table index is set to: one minus the value of "Job". That is, if "ASinCos" was invoked by "ACos", the table index is given the value "0", and if called by "ASin" it is given the value "1". The index value is used to indicate which path was taken from step 4.

If the right path of the flowchart was taken, then the next step is insure that a legal argument was passed by a user. If the argument is greater-than one it is an undefined argument. The current design sets the return value to the maximum representable value of the implemented function type. However, this is an area requiring further analysis, and should provide a better means of handling such exceptions.

Step 8 reduces the argument, such that step 13 will compute the arcsine of the compliment angle. Examine Figure 5 in chapter 2. If the results of step 13 are plotted for those arguments lying in the interval $[-.5, 1]$ (as a result of taking the right path of the flowchart); they would form a curve that is the the "mirror" of the arccosine curve (i.e. for the portion of the arccosine curve in the interval $[-.5, 1]$, rotate it 90 degrees about the horizontal axis). This information will prove useful in understanding how $\sin^{-1}(X)$ is reconstructed from $\sin^{-1}(Y)$.

If, in step 4, the argument is less-than-or-equal-to .5; the left path of the flowchart is taken. If the the argument is less than a predefined epsilon, then "Result" is set to "Y". This is one of two steps that differ for the two implementations. The fixed-point implementation

P_0	0.00000_00000_000 E+1	0.00000_00000_000 E+1
P_1	-0.27516_55529_060 E+1	+0.85372_16436_677 E+1
P_2	0.29058_76237_486 E+1	-0.13428_70791_343 E+2
P_3	-0.59450_14419_325 E+0	+0.59683_15761_775 E+1
P_4		-0.65404_06899_934 E+0
Q_0	-0.16509_93320_242 E+2	0.51223_29862_011 E+2
Q_1	0.24864_72896_916 E+2	-0.10362_27318_640 E+3
Q_2	-0.10333_86707_211 E+2	0.68719_59765_381 E+2
Q_3	0.10000_00000_000 E+1	-0.16429_55755_750 E+2
Q_4		0.10000_00000_000 E+1

Table 8 ASin/ACos Fixed and Floating-Point Coefficients

converts "Result" to pi-radian measure by dividing by pi. If the argument is greater-than the epsilon, then "G" is set to Y^2 . (steps 10-12, Figure 17)

Step 13 computes the rational approximation of arcsine, and is implemented in a manner identical to that described in the "Tan and Cot" section of this chapter. This is the only other step that differs in the two algorithms. The fixed-point implementation is expressed in pi-radian measure by dividing "R(g)" by pi. The coefficients of each polynomial used in the two type implementations are shown in Table 8.

The last step of the algorithm is necessary to reconstruct $\sin^{-1}(X)$ from the result, $\sin^{-1}(Y)$, generated by step 13. The two tables "A" and "B" are used in this process, and require the knowledge of which function

invoked the "ASinCos" function, as well as, which path was taken during the argument reduction step.

Examine Figure 5, and also note the relationships expressed in (26), (27), and (28). Earlier, it was mentioned that, if the right path of the flowchart is taken for argument reduction (i.e. the argument "Y" is in the interval $[-.5, 1]$); the arcsine forming the compliment angle is computed. A curve of the possible results would "mirror" the arccosine curve shown in Figure 5. If the function "ACos" is the invoking routine, and the original argument is positive, then the result lying on the mirrored curve is negative, with respect to, the arccosine curve shown in Figure 5. Therefore, the results from step 13, are subtracted from zero. (steps 14, 15, and 17; Figure 17) If the original argument is negative, the results on the "mirrored" curve are π less than the arccosine curve, and requires that π (1 in π -radian measure) be added to reconstruct the true function. (steps 14-16, Figure 17)

If arccosine is being approximated, and the left path of argument reduction is taken (i.e. the argument "Y" is in the interval $[0, .5]$), the results of step 13 represent the arcsine curve shown in Figure 5. If the original argument is positive, then arcsine curve is reconstructed into an arccosine curve by subtracting the results of step 13 from $\pi/2$. (steps 14, 15, and 17; Figure 17) If the original argument was negative, the sine curve in the interval $[0, .5]$, is $\pi/2$ (.5 in π -radian measure) less than that represented by the arccosine curve shown in the interval $[-.5, 0]$; therefore, the arccosine is constructed from the results of step 13, by adding $\pi/2$. (steps 14-16, Figure 17)

If the arcsine function is being approximated, and the right path of argument reduction was taken (i.e. "Y" lies in the interval $[\pi/2, \pi]$); the results of step 13 are represented by the "mirrored" curve. This curve is $\pi/2$ (.5 in pi-radian measure) less than the function being approximated, and requires that $\pi/2$ be added to reconstruct it to the true function. If the left path was taken, the results of step 13 are already represented by the arcsine curve. If the original argument was negative, then the results of step 18 are antisymmetric, with respect to, the true value of the function, and are complimented.

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.

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

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

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 Chebyshev 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 of 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,

$$\text{ADX}(1.0,2) = 4.0$$

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.

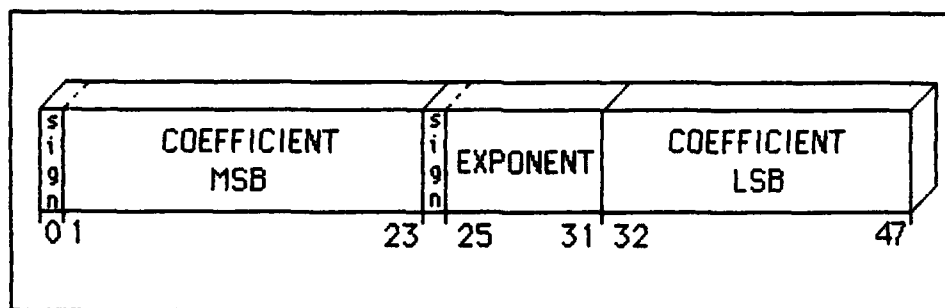


Figure 18 Bit Layout of 1750A Floating-Point Number

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 18 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 18).

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 $\times 2^\epsilon$. 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.

[illegible]

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:

$$\begin{array}{r}
 1 * 2^{-1} = .50 \\
 - 1 * 2^{-1} = .25 \\
 \hline
 = .75
 \end{array}$$

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).

AD-A163 905

DEVELOPMENT OF A RUN TIME MATH LIBRARY FOR THE 1750A
AIRBORNE MICROCOMPUTER(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING

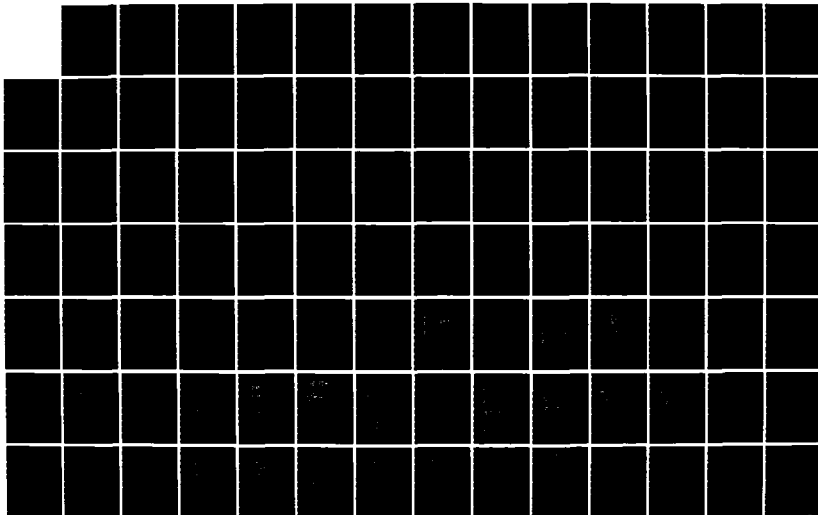
2/3

UNCLASSIFIED

S A HOTCHKISS DEC 85 AFIT/GCS/MA/850-5

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

Appendix B

```

*
* DATE:          18 JULY 1985
* VERSION:       1.0
* NAME:          Cos
* MODULE NUMBER: 1.0
* DESCRIPTION:
*   This function is invoked to compute the fixed-point value for the
*   cosine of an angle that has been expressed in pi-radians. A
*   pi-radian can be expressed as a value between -1.0 and 1.0, and
*   whose value when multiplied by pi is equivalent to an angle
*   expressed in radian measure. Fixed-point has the advantage of
*   speed of computation over floating-point algorithms, and using
*   pi-radians further exploits this difference by simplifying the
*   critical step of range reduction. The range reduction effort is
*   performed in SinCos, and is used to reduce the passed argument
*   into the interval  $(-.5\pi, .5\pi)$ . This range reduction insures
*   maximum accuracy of the approximated function. Both the argument
*   'Xx' and the returned value 'Cos' are expressed in 1750A double-
*   precision fixed-point representation. Both have one sign bit, one
*   integer bit, and 30 fractional bits. The actual value returned is
*   computed by the function 'SinCos'. Because the identity function,
*    $\text{Sin}(x) = \text{cos}(x - \pi/2)$ , 'SinCos' can be invoked to compute both
*   sine and cosine values.
* PASSED VARIABLES: Xx - The pi-radian measure for which cosine computed.
*   The argument is in double precision fixed-point.
* RETURNS: Cos - the computed value in double precision fixed-point
* MODULES CALLED: SinCos
* AUTHOR:        Capt. Steven A. Hotchkiss
* 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 SinCos  RENT (Xx) A  1,30;
  BEGIN
  ITEM  Xx          A  1,30;
  END

```

***** Cosine Procedure *****

```

DEF PROC Cos      RENT (Xx)          A  1,30;

  BEGIN

  ITEM Xx          A  1,30;
  ITEM Arg         A  1,30;

```

```
IF Xx < 0.0;  
  Ang = -Xx;  
  
Ang = Xx + .5;  
  
Cos = SinCos(Ang);  
  
RETURN;  
END  
TERM
```

```

*****
* DATE:          18 JULY 1985
* VERSION:       1.0
* NAME:          Sin
* MODULE NUMBER: 1.0
* DESCRIPTION:
*   This function is invoked to compute the fixed-point value for the
*   sine of an angle that has been expressed in pi-radians. A
*   pi-radian can be expressed as a value between -1.0 and 1.0, and
*   whose value when multiplied by pi is equivalent to an angle
*   expressed in radian measure. Fixed-point has the advantage of
*   speed of computation over floating-point algorithms, and using
*   pi-radians further exploits this difference by simplifying the
*   critical step of range reduction. The range reduction effort is
*   performed in SinCos, and is used to reduce the passed argument
*   into the interval  $(-.5\pi, .5\pi)$ . This range reduction insures
*   maximum accuracy of the approximated function. Both the argument
*   'Xx' and the returned value 'Sin' are expressed in 1750A double-
*   precision fixed-point representation. Both have one sign bit, one
*   integer bit, and 30 fractional bits. The actual value returned is
*   computed by the function 'SinCos'. Because the identity function,
*    $\sin(x) = \cos(x - \pi/2)$ , 'SinCos' can be invoked to compute both
*   sine and cosine values.
* PASSED VARIABLES: Xx - The pi-radian measure for which sine is computed.
*                   The argument is in double precision fixed-point.
* RETURNS: Sin - the computed value in double precision fixed-point
* MODULES CALLED: SinCos
* AUTHOR:        Capt. Steven A. Hotchkiss
* 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 SinCos RENT (Xx) A 1,30;
  BEGIN
  ITEM Xx A 1,30;
  END

***** Sin Procedure *****

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

  Sin = SinCos(Xx);

  RETURN;
END
TERM

```

```

*
*
* DATE:          19 JULY 1985
* VERSION:       1.0
* NAME:          SinCos
* MODULE NUMBER: 1.1
* DESCRIPTION:
*
*   This function is called by either 'Sin' or 'Cos'. The algorithm
*   used is designed around a polynomial approximation of the sine
*   function. The coefficients were derived through a Chebyshev
*   economization of the power series. Since 'Sin' and 'Cos' use
*   pi-radians as an argument, the economized coefficients were
*   multiplied by an appropriate power of pi. The first step of the
*   algorithm is to reduce the argument to the interval for which
*   the polynomial is valid (-.5 pi, .5 pi). The next step is to
*   determine if the argument would cause an underflow, if it does
*   the value of the argument is the value returned (sin(x) -> x for
*   small x). In fixed-point arithmetic, underflow causes doesn't
*   cause problems, but checking for underflow prevents unnecessary
*   computations. The computed result of the polynomial reflects the
*   use of Horner's rule. This function and its argument are double
*   precision fixed-point values.
*
* PASSED VARIABLES: Xx - double precision fixed-point representation of
*                      an angle expressed in pi-radians (radians
*                      divided by pi equals pi-radians). Legal range
*                      of values include -1.0 to 1.0
*
* RETURNS:          A 1750A double precision fixed-point representation
*                      of 'Sin' or 'Cos'
*
* MODULES CALLED:   None
*
* AUTHOR:           Capt. Steven A. Hotchkiss
*
* 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 SinCos RENT (Xx) A 1,30;

  BEGIN

    CONSTANT ITEM Eps      A 1,30 = 0.0005648517;
    CONSTANT ITEM PiFixed  A 2,29 = 3.1415926536;

    CONSTANT ITEM R1       F 39 = -1.644934061140;
    CONSTANT ITEM R2       F 39 = +0.8117421707751;
    CONSTANT ITEM R3       F 39 = -0.1907476226769;
    CONSTANT ITEM R4       F 39 = +0.0261162053162;
    CONSTANT ITEM R5       F 39 = -2.2352240374056E-3;

    CONSTANT ITEM PiFloat  F 39 = 3.141592653589;
    CONSTANT ITEM One      F 39 = +1.0;

    ITEM Ff      A 1,30;
    ITEM Rr      A 1,30;

```

```

ITEM  Xx          A  1,30;
ITEM  Gg          F  39;
ITEM  Sign        A  1,0;
ITEM  Result      F  39;

```

" Range reduction phase "

```

Sign = One;
Ff = Xx;

```

```

IF Ff < 0.0;
BEGIN
  Ff = -Ff;
  Sign = -Sign;
END

```

```

IF Ff > 1.0;
BEGIN
  Sign = -Sign;
  Ff = Ff - 1.0;
END

```

```

IF Ff > .5;
  Ff = 1.0 - Ff;

```

" Test for underflow conditons "

```

IF Ff <= Eps;
BEGIN
  Ar = PIFixed;
END

```

```

ELSE
BEGIN
  " Begin Computation of polynomial "
  Gg = (* F 39 *) ((* A 1,30 *) (Ff * Ff));
  Result = (((((R5*Gg+R4)*Gg+R3)*Gg+R2)*Gg+R1)*Gg;
  Result = Result * PIFloat + PIFloat;
  Ar = (* A 1,30 *) ( Result );
END

```

```

SinCos = (* A 1,30 *) ((* A 1,30 *) ( Ar * Ff ) * Sign);
RETURN;

```

END

TERM

```

*****
*
* DATE:          18 JULY 1985
* VERSION:       1.0
* NAME:          CosF
* MODULE NUMBER: 1.0
* DESCRIPTION:
*   This function is invoked to compute the floating point value for
*   the cosine of an angle that has been expressed in radian measure.
*   Both the argument 'Xx' and the returned value 'Cos' are in 1750A
*   extended precision floating point representation. The actual
*   value returned is computed by the function 'SinCosF'. Because of
*   the identity function,  $\sin(x) = \cos(x - \pi/2)$ , 'SinCosF' can
*   be invoked to compute both sine and cosine values.
* PASSED VARIABLES: Xx - The radian measure for which sine is computed.
*                   The argument is in extended precision float.
* RETURNS: CosF - the computed value in extended floating point.
* MODULES CALLED: SinCosF
* AUTHOR:        Capt. Steven A. Hotchkiss
* 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 SinCosF  RENT (Xx) F  39;
  BEGIN
    ITEM  Xx          F      39;
  END

```

***** Cosf Procedure *****

```

DEF PROC CosF      RENT (Xx)          F  39;

  BEGIN

    ITEM          Xx          F      39;
    ITEM          Yy          F      39;

    CONSTANT ITEM Ymax          F      39    = +0.2329350209881E+7;
    CONSTANT ITEM PiDivByTwo    F      39    = 1.57079632679489;
    CONSTANT ITEM Zero          F      39    = 0.0;

    Yy = Xx;

    IF (Yy > Ymax) OR (Yy < -Ymax);
      CosF = Zero;
    ELSE
      IF Yy < Zero;
        Yy = - Yy;

      Yy = PiDivByTwo + Yy;

```

```
      CosF = SinCosF(Vy);  
    RETURN;  
  END  
TERM
```

```

*****
*
* DATE:          18 JULY 1985
* VERSION:       1.0
* NAME:          SinF
* MODULE NUMBER: 1.0
* DESCRIPTION:
*   This function is invoked to compute the floating point value for
*   the sine of an angle that has been expressed in radian measure.
*   Both the argument 'Xx' and the returned value 'Sin' are in 1750A
*   extended precision floating point representation. The actual
*   value returned is computed by the function 'SinCosF'. Because of
*   the identity function,  $\sin(x) = \cos(x - \pi/2)$ , 'SinCosF' can
*   be invoked to compute both sine and cosine values.
* PASSED VARIABLES: Xx - The radian measure for which sine is computed.
*                   The argument is in extended precision float.
* RETURNS: SinF - the computed value in extended floating-point.
* MODULES CALLED: SinCosF
* AUTHOR:         Capt. Steven A. Hotchkiss
* 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 SinCosF  RENT (Xx) F  39;
  BEGIN
  ITEM  Xx          F  39;
  END

```

***** SinF Procedure *****

```

DEF PROC SinF      RENT (Xx)          F  39;
  BEGIN
  ITEM  Xx          F  39;
  SinF = SinCosF(Xx);
  RETURN;
END
TERM

```

```

*****
* DATE:          19 JULY 1985
* VERSION:       1.0
* NAME:          SinCosF
* MODULE NUMBER: 1.1
* DESCRIPTION:
*   This function is called by either 'SinF' or 'CosF'. The algorithm
*   used is a polynomial approximation of the sine function. The
*   coefficients were determined through a Chebyshev Economization
*   of the power series, and were carried out to 1750R machine
*   precision. For efficient computation, the polynomial was
*   computed using Horner's Rule. This function returns an extended
*   precision floating-point value.
* PASSED VARIABLES: Job - a flag indicating whether to compute either
*   the sine or cosine of an angle expressed in
*   radians
*   AbsX - the absolute value of the angle under
*   consideration
*   Yy - originally set to AbsX, but modified
*   by the algorithm for range reduction of
*   the original angle.
* RETURNS:       An Extended precision floating-point approximation to
*   a user requested call to either 'SinF' or 'CosF'
* MODULES CALLED: None
* AUTHOR:        Capt. Steven A. Hotchkiss
* 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 SinCosF RENT (Xx) F 39;
```

```
BEGIN
```

```

CONSTANT ITEM R1      F 39  = -1.6449340668480;
CONSTANT ITEM R2      F 39  = +0.8117424252778;
CONSTANT ITEM R3      F 39  = -0.1907518239486;
CONSTANT ITEM R4      F 39  = +2.6147845158310E-2;
CONSTANT ITEM R5      F 39  = -2.3460587938600E-3;
CONSTANT ITEM R6      F 39  = +1.4832528223590E-4;
CONSTANT ITEM R7      F 39  = -6.7236447557180E-6;

CONSTANT ITEM Eps      F 39  = +0.134869915234861E-5;
CONSTANT ITEM Pi       F 39  = 3.1415926535898;
CONSTANT ITEM One      F 39  = 1.0;
CONSTANT ITEM Zero     F 39  = 0.0;
CONSTANT ITEM OneHalf  F 39  = 0.5;
CONSTANT ITEM OneDivByPi F 39  = +0.31830988618379E0;

ITEM Sign              F      39;
ITEM Gg                F      39;
ITEM Xx                F      39;
ITEM PiRadians         F      39;

```

```
ITEM Result F 39;
```

```
TABLE Overlays (0) W 6;
```

```
BEGIN
```

```
ITEM Ff F 39 POS(0,0);
```

```
ITEM Fexp S 7 POS(8,1);
```

```
ITEM Fbits B 48 POS(0,0);
```

```
ITEM Vy F 39 POS(0,3);
```

```
ITEM Yexp S 7 POS(8,4);
```

```
ITEM Ybits B 48 POS(0,3);
```

```
END
```

```
Ff(0) = Xx * OneDivByPi;
```

```
Sign = One;
```

```
IF Ff(0) < Zero;
```

```
BEGIN
```

```
Ff(0) = -Ff(0);
```

```
Sign = - Sign;
```

```
END
```

```
Vy(0) = Zero;
```

```
IF Fexp(0) >= 1;
```

```
BEGIN
```

```
IF BIT(Fbits(0),Fexp(0),1) = 18'1';
```

```
Sign = -Sign;
```

```
Yexp(0) = Fexp(0);
```

```
BIT(Ybits(0),0,Fexp(0)+1) = BIT(Fbits(0),0,Fexp(0)+1);
```

```
END
```

```
PiRadians = Ff(0) - Vy(0);
```

```
IF PiRadians > OneHalf;
```

```
PiRadians = One - PiRadians;
```

```
IF PiRadians < Eps;
```

```
Result = PiRadians * Pi;
```

```
ELSE
```

```
BEGIN
```

```
Gg = PiRadians * PiRadians;
```

```
Result = ((((((( R7 * Gg + R6) * Gg + R5) * Gg + R4) * Gg + R3) * Gg  
+ R2) * Gg + R1) * Gg) + One) * PiRadians * Pi;
```

```
END
```

```
SinCosF = Result * Sign;
```

```
RETURN;
```

```
END
```

```
TERM
```

```

*****
*
* DATE:          6 August 1985
* VERSION:       1.0
* NAME:          Cot
* MODULE NUMBER: 1.0
* DESCRIPTION:
*   Cot is invoked to compute the cotangent of a user passed angle. The
*   angle is expressed in pi-radians:  pi-radians --> radians/pi.
*   The range of legal values for the angle are <-1.0,1.0> Both the
*   argument and result are expressed as fixed point results. The
*   value returned is computed by the function TanCot.
* PASSED VARIABLES: Xx - the angle in pi-radians
* RETURNS:         a value for the cotangent of Xx.
*                  plus and minus infinity are considered as -4095 and 4095
* MODULES CALLED:  TanCot
* AUTHOR:          Capt. Steven A. Hotchkiss
* 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 TanCot RENT(Arg, Job) A 12,18;
BEGIN
  ITEM Arg  A 1,30;
  ITEM Job  B;
END

```

***** Cot Procedure *****

```

DEF PROC Cot RENT(Xx) A 12,18;

```

```

BEGIN

```

```

  DEFINE Tangent  "1B'0'";
  DEFINE Cotangent "1B'1'";

```

```

  ITEM Xx          A 1,30;
  ITEM Job         B;

```

```

  Job = Cotangent;
  Cot = TanCot(Xx, Job);

```

```

  RETURN;

```

```

END
TERM

```

```

*
* DATE:          6 August 1985
* VERSION:       1.0
* NAME:          Tan
* MODULE NUMBER: 1.0
* DESCRIPTION:
*   Tan is invoked to compute the tangent of a user passed angle. The
*   angle is expressed in pi-radians:  pi-radians --> radians/pi.
*   The range of legal values for the angle are <-1.0,1.0> Both the
*   argument and result are expressed as fixed point results. The
*   value returned is computed by the function TanCot.
* PASSED VARIABLES: Xx - the angle in pi-radians
* RETURNS:         a value for the tangent of Xx.
*                  plus and minus infinity are considered as -4095 and 4095
* MODULES CALLED:  TanCot
* AUTHOR:          Capt. Steven A. Hotchkiss
* 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 TanCot RENT(Arg, Job) A 12,18;
BEGIN
  ITEM Arg    A 1,30;
  ITEM Job    B;
END

```

***** Tan Procedure *****

```

DEF PROC Tan RENT(Xx) A 12,18;
BEGIN
  DEFINE Tangent  "1B'0'";
  DEFINE Cotangent "1B'1'";

  ITEM Xx          A 1,30;
  ITEM Job         B;

  Job = Tangent;
  Tan = TanCot(Xx, Job);

  RETURN;
END
TERM

```

```

*****
*
* DATE:          6 August 1985
* VERSION:       1.0
* NAME:          TanCot
* MODULE NUMBER: 1.1
* DESCRIPTION:
*   This function is invoked by either 'Tan' or 'Cot' to compute the
*   appropriate value for a given angle. The angle 'Ang' is given in
*   pi-radians: pi-radian -> radians/pi. Since tan(x) = 1/cot(x)
*   both functions can call this routine to compute the desired values.
*   Since Tangent and Cotangent approach plus or minus infinity at
*   certain angles, a check is required to prevent degradation due to
*   underflow or overflow. plus and minus infinity for this procedure
*   are considered to be 4095 and -4095 respectively. The coefficient
*   used in the two polynomials were determined through the PADE
*   method. The original polynomial was determined through a
*   Chebyshev economization of the power series for tangent.
* PASSED VARIABLES: Ang - an angle expressed in pi-radians
*                   Job - tells whether to compute tangent or cotangent
* RETURNS:         Either the tangent or cotangent of a given angle.
*                   there are 12 integer bits and 18 fraction bits for this
*                   function.
* MODULES CALLED:  None
* AUTHOR:          Capt. Steven A. Hotchkiss
* 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 TanCot RENT(Ang, Job) A 12,18;

  BEGIN

    DEFINE Tangent "1B'0'";
    DEFINE Cotangent "1B'1'";
    DEFINE Pg "Polynomial(0)";
    DEFINE Qg "Polynomial(1)";

    ITEM Job B;
    ITEM Ang A 1,30;
    ITEM Xx A 1,30;
    ITEM Xfloat F 39;
    ITEM Gg F 39;
    ITEM Sign F 39;

    ITEM One STATIC F 39 = 1.0;

    CONSTANT ITEM PiFloat F 39 = 3.141592653590;
    CONSTANT ITEM PiFixed A 2,29 = 3.1415926536;
    CONSTANT ITEM Eps A 1,30 = 0.00007773138;
    CONSTANT ITEM UpperLimit A 12,18 = 4095.0;

```

```

        ITEM P0 STATIC F 39;
CONSTANT ITEM P1          F 39 = -1.099093361855;
CONSTANT ITEM P2          F 39 = 0.104729845797;
OVERLAY PIFloat: P0;

```

```

        ITEM Q0 STATIC F 39;
CONSTANT ITEM Q1          F 39 = -4.388961479150;
CONSTANT ITEM Q2          F 39 = 1.555953608405;
OVERLAY One: Q0;

```

```

TABLE Array (0:1);
BEGIN
    ITEM Polynomial F 39;
END

```

```

ITEM Numerator      U 8;
ITEM Denominator    U 8;
ITEM Num            B 8;
ITEM Den            B 8;

```

```

OVERLAY Numerator : Num;
OVERLAY Denominator : Den;

```

```

Sign = One;
Xx = Arg;

```

```

IF Job = Cotangent;
BEGIN
    Numerator = 1;
    Denominator = 0;
END

```

```

ELSE
BEGIN
    Numerator = 0;
    Denominator = 1;
END

```

```

IF Xx < 0.0;
BEGIN
    Xx = -Xx;
    Sign = -Sign;
END

```

```

IF Xx > 1.0;
Xx = Xx - 1.0;

```

```

IF Xx > .5;
BEGIN
    Xx = 1.0 - Xx;
    Sign = -Sign;
END

```

```

IF Xx > .25;
BEGIN
    Xx = .5 - Xx;
    BIT(Num,7,1) = NOT BIT(Num,7,1);
    BIT(Den,7,1) = NOT BIT(Den,7,1);
END;

```

```

Xfloat = (* F 39 *) (Xx);
IF Xx < Eps;
  BEGIN
    Pg = Xfloat * PFloat;
    Qg = One;
  END
ELSE
  BEGIN
    Gg = Xfloat * Xfloat;
    Pg = ((P2 * Gg + P1) * PFloat + P0) * Xfloat;
    Qg = (Q2 * Gg + Q1) * Gg + Q0;
  END

TanCot = (* A 12, 18 *) (Sign *
  Polynomial(Numerator)/Polynomial(Denominator));

RETURN;
END
TERM

```

```

*
* DATE:          19 JULY 1985
* VERSION:       1.0
* NAME:          CotF
* MODULE NUMBER: 1.0
* DESCRIPTION:
*   This function is invoked to compute the cotangent of an angle
*   expressed in radian measure. Because of the identity function
*    $\text{Tan}(X) = 1/\text{Cot}(X)$ , the two functions 'Tan' and 'Cot' can both
*   invoke the function 'TanCot' to compute their respective values.
*   Both the argument and the result of this function are expressed
*   in 1730A extended precision floating-point representation.
* PASSED VARIABLES: Xx - the angle of interest, expressed in extended
*                     floating-point representation. The angle must
*                     lie between ( -2329349.90332, 2329349.90332 )
* RETURNS:          Cotangent of the angle Xx (-1.0 to 1.0) in extended
*                     floating-point representation
* MODULES CALLED:   TanCotF
* AUTHOR:          Capt. Steven A. Hatchkiss
* 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 TanCotF RENT (Arg, Job) F 39;
BEGIN
  ITEM   Arg      F      39;
  ITEM   Job      B;
END

```

***** Cotf Procedure *****

```

DEF PROC CotF RENT (Xx) F 39;
  BEGIN
    DEFINE   Tangent      "1B'0'";
    DEFINE   Cotangent    "1B'1'";

    ITEM     Xx           F      39;
    ITEM     Job          B;

    Job = Cotangent;
    CotF = TanCotF(Xx, Job);

    RETURN;
  END
TERM

```

```

*-----*
* DATE:          19 JULY 1985
* VERSION:       1.0
* NAME:          TanF
* MODULE NUMBER: 1.0
* DESCRIPTION:
*   This function is invoked to compute the tangent of an angle
*   expressed in radian measure. Because of the identity function
*    $\text{Tan}(X) = 1/\text{Cot}(X)$ , the two functions 'Tan' and 'Cot' can both
*   invoke the function 'TanCot' to compute their respective values.
*   Both the argument and the result of this function are expressed
*   in 1750A extended precision floating-point representation.
* PASSED VARIABLES: Xx - the angle of interest, expressed in extended
*                     floating-point representation. The angle must
*                     lie between ( -2329349.90332, 2329349.90332 )
* RETURNS:          Tangent of the angle Xx (-1.0 to 1.0) in extended
*                     floating-point representation
* MODULES CALLED:   TanCotF
* AUTHOR:           Capt. Steven A. Hatchkiss
* 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 TanCotF RENT (Arg, Job) F 39;
BEGIN
  ITEM   Arg      F      39;
  ITEM   Job      B;
END

```

***** Tanf Procedure *****

```

DEF PROC TanF RENT (Xx) F 39;

```

```

  BEGIN

```

```

    DEFINE   Tangent      "1B'0'";
    DEFINE   Cotangent    "1B'1'";

```

```

    ITEM     Xx           F      39;
    ITEM     Job          B;

```

```

    Job = Tangent;
    TanF = TanCotF(Xx, Job);

```

```

  RETURN;

```

```

END
TERM

```

```

*****
*
* DATE:          19 JULY 1985
* VERSION:       1.0
* NAME:          TanCotF
* MODULE NUMBER: 1.1
* DESCRIPTION:
*   This function is called by either 'Tan' or 'Cot' to compute the
*   tangent or cotangent of their respective angles. The result is
*   computed by using a Rational approximation [ P(X)/Q(X) ]. The
*   coefficients used were found by a Pade approximation of the
*   Chebyshev economization of the power series for tangent functions.
*   The result of this function is in 1750A extended precision
*   floating-point representation.
* PASSED VARIABLES: Xx - the angle under consideration, expressed in
*                     radians. Xx must lie between
*                     (-2329349.90332, 2329349.90332)
*                     Yy - absolute value of Xx, used to prevent overflow
*                     iflag - designates whether to compute Tan or Cot
* RETURNS:         1750A extended precision float for Tan or Cot
* MODULES CALLED:  None
* AUTHOR:          Capt. Steven A. Hotchkiss
* 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 TanCotF RENT (Arg,Job) F 39;
```

```
BEGIN
```

```

DEFINE   Tangent   "1B'0'";
DEFINE   Cotangent "1B'1'";
DEFINE   Pg        "Polynomial(0)";
DEFINE   Qg        "Polynomial(1)";

```

```

TABLE Array (0:1);
BEGIN
  ITEM Polynomial F 39;
END

```

```

TABLE Overlays (0) W 6;
BEGIN
  ITEM Xx      F 39 POS(0,0);
  ITEM Xexp    S 7  POS(8,1);
  ITEM Xbits   B 48 POS(0,0);

  ITEM Yy      F 39 POS(0,3);
  ITEM Yexp    S 7  POS(8,4);
  ITEM Ybits   B 48 POS(0,3);
END

```

```

ITEM Arg      F 39;
ITEM Job      B;

```

```

ITEM Numerator      U      8;
ITEM Denominator    U      8;
ITEM Num            B      8;
ITEM Den            B      8;

```

```

OVERLAY Numerator : Num;
OVERLAY Denominator : Den;

```

```

ITEM Gg              F 39;
ITEM Sign            F 39;

```

```

CONSTANT ITEM      P0      F 39 ==+1.0;
CONSTANT ITEM      P1      F 39 ==-1.2661071041410;
CONSTANT ITEM      P2      F 39 ==+0.2733219453881;
CONSTANT ITEM      P3      F 39 ==-7.1946857833630E-3;

```

```

CONSTANT ITEM      Q0      F 39 ==+1.0;
CONSTANT ITEM      Q1      F 39 ==-4.5559752378380;
CONSTANT ITEM      Q2      F 39 ==+2.2740008937200;
CONSTANT ITEM      Q3      F 39 ==-0.2003996971354;

```

```

CONSTANT ITEM      Ymax     F 39 ==+0.23293499033203E+7;
CONSTANT ITEM      Zero     F 39 == 0.00;
CONSTANT ITEM      One      F 39 == 1.00;
CONSTANT ITEM      OneHalf  F 39 == 0.50;
CONSTANT ITEM      OneFourth F 39 == 0.25;
CONSTANT ITEM      TwoDivByPi F 39 ==+0.6366197723675;
CONSTANT ITEM      PiFloat  F 39 ==+3.1415926535898;
CONSTANT ITEM      Eps      F 39 ==+0.1348699152348E-5;

```

```

Sign = One;
Vy(0) = Zero;
Xx(0) = Arg * TwoDivByPi;

```

```

IF Job = Cotangent;
BEGIN
  Numerator = 1;
  Denominator = 0;
END

```

```

ELSE
BEGIN
  Numerator = 0;
  Denominator = 1;
END

```

```

IF Xx(0) < Zero;
BEGIN
  Xx(0) = -Xx(0);
  Sign = -Sign;
END

```

```

IF (Xexp(0) > 0) AND (BIT(Xbits(0),Xexp(0),1) = 1B'1');
BEGIN
  Sign = -Sign;
  BIT(Num,7,1) = NOT BIT(Num,7,1);
  BIT(Den,7,1) = NOT BIT(Den,7,1);
END;

```

```

IF Xexp(0) > 1;
  BEGIN
    Yexp(0) = Xexp(0);
    BIT(Ybits(0),0,Xexp(0)+1) = BIT(Xbits(0),0,Xexp(0)+1);
  END;
Xx(0) = Xx(0) - Yy(0);

IF Xx(0) <> Zero;
  Xexp(0) = Xexp(0) - (* S 7 *) ( 1 );

IF Xx(0) > OneFourth;
  BEGIN
    Xx(0) = OneHalf-Xx(0);
    BIT(Num,7,1) = NOT BIT(Num,7,1);
    BIT(Den,7,1) = NOT BIT(Den,7,1);
  END

IF Xx(0) < Eps;
  BEGIN
    Pg = Xx(0) * PiFloat;
    Qg = One;
  END
ELSE
  BEGIN
    Gg = Xx(0) * Xx(0);
    Pg = ((P3 * Gg + P2) * Gg + P1) * Gg * PiFloat + P0) * Xx(0);
    Qg = ((Q3 * Gg + Q2) * Gg + Q1) * Gg + Q0;
  END

TanCotF = Sign * Polynomial(Numerator)/Polynomial(Denominator);

RETURN;

```

END
TERM

```

*****
*
* DATE:          21 August 1985
* VERSION:       1.0
* NAME:          ACos
* MODULE NUMBER: 1.0
* DESCRIPTION:
*
*               This program is used to call ASinCos. ASinCos is the
*               routine that actually computes the inverse cosine for
*               argument Xx. They are coded this way because both ASin
*               ACos can use the same routine for their computations.
*               This is because of the identity function
*               ACos(X) = pi/2 - ASin(x)
*
* PASSED VARIABLES: Xx - the cosine for which an angle is to be
*                   computed.
*
* RETURNS:        ACos - expressed in fixed-point pi-radians
*
* MODULES CALLED: ASinCos
*
* AUTHOR:         Capt. Steven A. Hotchkiss
*
* 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 ASinCos RENT(Arg, Job) A 1,30;
BEGIN
  ITEM Arg A 1,30;
  ITEM Job U 8;
END

```

***** ACos Procedure *****

```

DEF PROC ACos RENT(Xx) A 1,30;
BEGIN

```

```

  DEFINE ArcSine "0";
  DEFINE ArcCosine "1";

```

```

  ITEM Xx A 1,30;
  ITEM Job U 8;

```

```

  Job = ArcCosine;
  ACos = ASinCos(Xx, Job);

```

```

  RETURN;

```

```

END
TERM

```

```

*****
*
* DATE:          21 August 1985
* VERSION:       1.0
* NAME:          ASin
* MODULE NUMBER: 1.0
* DESCRIPTION:
*
*           This program is used to call ASinCos. ASinCos is the
*           routine that actually computes the inverse sine for
*           argument Xx. They are coded this way because both ASin
*           ACos can use the same routine for their computations.
*           This is because of the identity function
*            $ACos(X) = \pi/2 - ASin(x)$ 
*
* PASSED VARIABLES: Xx - the fixed-point inverse for which an angle is
*                   is to be computed
*
* RETURNS:        ASin - the angle in pi-radians.
*
* MODULES CALLED:  ASinCos
*
* AUTHOR:         Capt. Steven A. Hotchkiss
*
* 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 ASinCos RENT(Ang, Job) A 1,30;
BEGIN
  ITEM Ang A 1,30;
  ITEM Job U 8;
END

```

***** ASin Procedure *****

```

DEF PROC ASin RENT(Xx) A 1,30;
BEGIN
  DEFINE ArcSine "0";
  DEFINE ArcCosine "1";

  ITEM Xx A 1,30;
  ITEM Job U 8;

```

```

  Job = ArcSine;
  ASin = ASinCos(Xx, Job);

```

```

  RETURN;
END
TERM

```

```

*****
*
* DATE:          21 August 1985
* VERSION:       1.0
* NAME:          ASinCos
* MODULE NUMBER: 1.1
* DESCRIPTION:
*   This program is written to return the correct angle for a user
*   passed argument. The argument represents the Sine or Cosine of
*   the angle to be returned. The returned angle is in pi-radian
*   measure (pi-radian -> radians/pi). This routine can be called
*   by either ASin or ACos because of the identity function
*    $ACos(x) = \pi/2 - ASin(x)$ . The coefficients were determined by
*   of the method of Chebyshev expansion as described in 'A First
*   Course in Numerical Analysis' by Anthony Ralston.
* PASSED VARIABLES: Arg - the sine or cosine of the angle to be
*                       determined. Variable is in fixed-point
*                       Job - tells whether to compute for ACos or ASin
* RETURNS:             The angle representation for the argument. The angle
*                       is in pi-radians. Legal values fall in the range
*                       (-1.0, 1.0)
* MODULES CALLED:     None
* AUTHOR:             Capt. Steven A. Hotchkiss
* 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 Sqrt      RENT(Xx)      F 39;
BEGIN
ITEM   Xx          F 39;
END

```

***** AsinCos Procedure *****

```

DEF PROC ASinCos RENT(Arg, Job) A 1,30;

```

BEGIN

```

DEFINE ArcSine      "0";
DEFINE ArcCosine    "1";
DEFINE Positive      "0";
DEFINE Negative      "1";

ITEM Job            U 8;
ITEM Ii            U 8;
ITEM Arg           A 1,30;
ITEM Result        A 1,30;
ITEM Pg            F 39;
ITEM Qg            F 39;

```

TABLE Overlays(0) W 6;

BEGIN

ITEM Yy F 39 POS(0,0);

ITEM Yexp S 7 POS(8,1);

ITEM Gg F 39 POS(0,3);

ITEM Gexp S 7 POS(8,4);

END

CONSTANT TABLE Constants (0:1) W 4 = 0.0, 1.0, 0.5, 0.5;

BEGIN

ITEM Aa A 1,30 POS(0,0);

ITEM Bb A 1,30 POS(0,2);

END

CONSTANT ITEM Eps F 39 = 9.587379924290E-5;

CONSTANT ITEM OneOverPi F 39 = 0.31830988618379;

CONSTANT ITEM One F 39 = 1.0;

CONSTANT ITEM OneHalf F 39 = 0.5;

CONSTANT ITEM OneInt S 7 = 1;

CONSTANT ITEM Xmax A 1,30 = 1.999999999;

CONSTANT ITEM P1 F 39 = -0.27516555290596E+1;

CONSTANT ITEM P2 F 39 = +0.29058762374859E+1;

CONSTANT ITEM P3 F 39 = -0.59450144193246E+0;

CONSTANT ITEM Q0 F 39 = -0.18509933202424E+2;

CONSTANT ITEM Q1 F 39 = +0.24864728969164E+2;

CONSTANT ITEM Q2 F 39 = -0.10333867072113E+2;

CONSTANT ITEM Q3 F 39 = +0.10000000000000E+1;

Yy(0) = (* F 39 *) (ABS(Ang));

IF Yy(0) <= OneHalf;

BEGIN

li = Job;

IF Yy(0) < Eps;

BEGIN

Result = (* A 1,30 *) (Yy(0) * OneOverPi);

GOTO L1;

END

Gg(0) = Yy(0) * Yy(0);

END

ELSE

BEGIN

li = 1 - Job;

IF Yy(0) > One;

BEGIN

ASinCos = Xmax;

ABORT;

END

Gg (0) = One - Yy(0);

Gexp(0) = Gexp(0) - OneInt; " Gg = Gg/2 or Gg = Gg * 2 ** -1 "

Yy (0) = -Sqrt(Gg(0));

Yexp(0) = Yexp(0) + OneInt; " Yy = Yy*2 or Yy = Yy * 2 ** 1 "

END

```

Pg      = ((P3 * Gg(0) + P2) * Gg(0) + P1) * Gg(0);
Qg      = ((      Gg(0) + Q2) * Gg(0) + Q1) * Gg(0) + Q0;

Result = (* A 1,30 *) ((Yy(0) + Yy(0) * Pg / Qg) * OneOverPi );

L1:  IF Job = ArcSine;
      BEGIN
        Result = (* A 1,30 *) ( Result + Aa(ii) );

        IF Arg < 0.0;
          Result = -Result;
        END
      ELSE                                     "ELSE Job = ArcCosine"
        IF Arg < 0.0;
          Result = (* A 1,30 *) ( Bb(ii) + Result );
        ELSE
          Result = (* A 1,30 *) ( Aa(ii) - Result );

        ASinCos = Result;

      RETURN;
    END
  TERM

```

```

*****
*
* DATE:          21 August 1985
* VERSION:       1.0
* NAME:          ACosf
* MODULE NUMBER: 1.0
* DESCRIPTION:
*
*           This program is used to call ASinCosf. ASinCosf is the
*           routine that actually computes the inverse cosine for
*           argument Xx. They are coded this way because both ASinf
*           ACosf can use the same routine for their computations.
*           This is because of the identity function
*            $ACos(X) = \pi/2 - ASin(x)$ 
*
* PASSED VARIABLES: Xx - the cosine for which an angle is to be
*                   computed.
*
* RETURNS:        ACosf - expressed in floating-point
*
* MODULES CALLED: ASinCosf
*
* AUTHOR:         Capt. Steven A. Hotchkiss
*
* 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 ASinCos  RENT(Arg, Job) F 39;
BEGIN
  ITEM Arg  F  39;
  ITEM Job  U   8;
END

```

***** ACosf Procedure *****

```

DEF PROC ACos  RENT(Xx)  F 39;
BEGIN

  DEFINE ArcSine  "0";
  DEFINE ArcCosine "1";

  ITEM Xx  F  39;
  ITEM Job  U   8;

```

```

  Job = ArcCosine;
  ACosf = ASinCosf(Xx, Job);

```

```

  RETURN;
END
TERM

```

```

*****
*
* DATE:          21 August 1985
* VERSION:       1.0
* NAME:          ASinf
* MODULE NUMBER: 1.0
* DESCRIPTION:
*
*               This program is used to call ASinCosf. ASinCosf is the
*               routine that actually computes the inverse sine for
*               argument Xx. They are coded this way because both ASinf
*               ACosf can use the same routine for their computations.
*               This is because of the identity function
*                $ACos(X) = \pi/2 - ASin(x)$ 
*
* PASSED VARIABLES: Xx - the floating-point inverse for which an angle is
*                   is to be computed
*
* RETURNS:        ASinf - the the angle .
*
* MODULES CALLED:  ASinCosf
*
* AUTHOR:         Capt. Steven A. Hetchkiss
*
* 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 ASinCosf RENT(Ang, Job) F 39;
BEGIN
  ITEM Ang F 39;
  ITEM Job U 8;
END

```

***** ASinf Procedure *****

```

DEF PROC ASin RENT(Xx) F 39;

```

BEGIN

```

  DEFINE ArcSine "0";
  DEFINE ArcCosine "1";

```

```

  ITEM Xx F 39;
  ITEM Job U 8;

```

```

  Job = ArcSine;
  ASinf = ASinCosf(Xx, Job);

```

RETURN;

END
TERM

```

*****
*
* DATE:          21 August 1985
* VERSION:       1.0
* NAME:          ASinCosf
* MODULE NUMBER: 1.1
* DESCRIPTION:
*   This program is written to return the correct angle for a user
*   passed argument. The argument represents the Sine or Cosine of
*   the angle to be returned. The returned angle is in radian
*   measure. This routine can be called
*   by either ASinf or ACosf because of the identity function
*    $ACos(x) = \pi/2 - ASin(x)$ . The coefficients were determined by
*   of the method of Chebyshev expansion as described in 'A First
*   Course in Numerical Analysis' by Anthony Ralston.
* PASSED VARIABLES: Arg - the sine or cosine of the angle to be
*                       determined. Variable is in floating-point
*                       Job - tells whether to compute for ACosf or ASinf
* RETURNS:             The angle representation for the argument. The angle
*                       is in radians. Legal values fall in the range
* MODULES CALLED:      None
* AUTHOR:             Capt. Steven A. Hotchkiss
* 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 Sqrt      RENT(Xx)      F  39;
BEGIN
ITEM  Xx           F  39;
END

```

***** ASinCos Procedure *****

```

DEF PROC ASinCos RENT(Arg, Job) F 39;

BEGIN

DEFINE ArcSine      "0";
DEFINE ArcCosine    "1";
DEFINE Positive      "0";
DEFINE Negative      "1";

ITEM  Job           U  8;
ITEM  Ii            U  8;
ITEM  Arg           F 39;
ITEM  Result        F 39;
ITEM  Pg            F 39;
ITEM  Qg            F 39;

```

TABLE Overlays(0) W 6;

```
BEGIN
ITEM Yy      F    39 POS(0,0);
ITEM Yexp    S    7  POS(8,1);
ITEM Gg      F    39 POS(0,3);
ITEM Gexp    S    7  POS(8,4);
END
```

CONSTANT TABLE Constants (0:1) W 6 =
0.0, 1.570796326795, 0.7853981633974, 0.7853981633974;

```
BEGIN
ITEM Aa      F    39 POS(0,0);
ITEM Bb      F    39 POS(0,3);
END
```

```
CONSTANT ITEM Eps      F    39 = 9.587379924290E-5;
CONSTANT ITEM OneOverPi F    39 = 0.31830988618379;
CONSTANT ITEM One      F    39 = 1.0;
CONSTANT ITEM OneHalf  F    39 = 0.5;
CONSTANT ITEM OneInt    S    7 = 1;
CONSTANT ITEM Xmax      F    39 = MAXFLOAT(39);
```

```
CONSTANT ITEM P1      F    39 = +0.8537216436677E+1;
CONSTANT ITEM P2      F    39 = -0.1342870791343E+2;
CONSTANT ITEM P3      F    39 = +0.5968315761775E+1;
CONSTANT ITEM P4      F    39 = -0.6540406699934E+0;
```

```
CONSTANT ITEM Q0      F    39 = +0.5122329862011E+2;
CONSTANT ITEM Q1      F    39 = -0.1036227318640E+3;
CONSTANT ITEM Q2      F    39 = +0.6871959765381E+2;
CONSTANT ITEM Q3      F    39 = -0.1642955755750E+2;
CONSTANT ITEM Q4      F    39 = +0.1000000000000E+1;
```

Yy(0) = ABS(Arg);

```
IF Yy(0) <= OneHalf;
BEGIN
  Ii = Job;
  IF Yy(0) < Eps;
  BEGIN
    Result = Yy(0);
    GOTO L1;
  END
  Gg(0) = Yy(0) * Yy(0);
END
```

```
ELSE
BEGIN
  Ii = 1 - Job;

  IF Yy(0) > One;
  BEGIN
    ASinCosf = Xmax;
    GOTO L2;
  END
```

```
Gg (0) = One - Yy(0);
Gexp(0) = Gexp(0) - OneInt;   " Gg = Gg/2  or  Gg = Gg * 2 ** -1 "
Yy (0) = -Sqrt(Gg(0));
```

```

Yexp(0) = Yexp(0) + OneInt;   " Vy = Vy*2 or Vy = Vy * 2 ** 1 "
END

```

```

Pg      = (((P4 * Gg(0) + P3) * Gg(0) + P2) * Gg(0) + P1) * Gg(0);
Qg      = (((      Gg(0) + Q3) * Gg(0) + Q2) * Gg(0) + Q1) * Gg(0) + Q0;

```

```

Result = Vy(0) + Vy(0) * Pg / Qg;

```

```

L1:  IF Job = ArcSine;
      BEGIN
        Result = Result + Aa(ii);

        IF Arg < 0.0;
          Result = -Result;
        END
      ELSE
        "ELSE Job = ArcCosine"
        IF Arg < 0.0;
          Result = Bb(ii) + Result;
        ELSE
          Result = Aa(ii) - Result;
        END
      ASinCos = Result;

L2:  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. Hatchkiss 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 RLog      Rent(Arg) F      39;
  BEGIN
  ITEM Arg F      39;
  END

REF PROC RLog10    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:                loRefs
* 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 loRefs;
```

```

"
' 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 CALF STATIC C 2;
OVERLAY Carriage: CALF;

```

```

"
' 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 Overlaid 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:     FitToChar
* 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 FitToChar (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 = FitToChar(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 = ' 0.000000000000E+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

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

```


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 address 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
* WriteRec
*
* 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(:ItsRec,Eof);

      " Put Loader Info into Contiguous Memory Locations "
      WdsInRec = Cnt1(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) + WdsInRec <=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 90 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;
      RcdType1 = AsciiB;
      WriteRcd;
    END

  END

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

    IF LdPt = LdRd(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;
          RcdType1 = AsciiB;
          WriteRcd;

          " Set the load point for this new loader record "
          LdRd(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
        END
      END
    END
  END

```

```

END

END "Same record not enough 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 "

      RcdType1 = AsciiB;
      WdsInRcd = BufPtr(Buff) - 1;
      WriteRcd;
      END
      END " end not first pass "

FirstPass = False;

" Set the load point for this loader record "
LdRcd(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
  RcdType1 = 8261; "blank E"
  OutBuff(0) = Laddr(0);
  OutBuff(1) = 30; " ascii record separator "
  WriteRcd;
  END

END "end execution address record "

END "end while loop "

" Write end of file loader record "

```

```
RedType1 = 8252; " blank F "  
OutBuff(0) = 30; " ascii record separator "
```

```
" 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 taks 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

```

!CONPOOL('RfMtCpl');
!CONPOOL ('IoData');

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

DEF PROC WriteRcd;
  BEGIN

    LoopCnt = WdsInBuffer;

    If RcdTyp1 = AsciiB;
      BEGIN

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

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

```

```

FOR ix: 1 BY 1 WHILE ix <= LoopCnt;
  BEGIN
    OUTBUFF(ix+1) = BufId(ix);
    ChkSum = ChkSum XOR OutBufFB(ix+1);
    END

    OutBufFB(ix+1) = ChkSum;

    Printf(RcdTyp1, OutFid);

  END
ELSE
  Printf(RcdTyp1, OutFid);

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

  RETURN;
END
TERM

```

```

*****
*
* DATE:          10 October 1985
* VERSION:       1.0
* NAME:          loCalls
* 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 loCalls;

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 CInUp;
  !LINKAGE FORTRAN;
  BEGIN
  END

REF PROC IntFil;
  !LINKAGE FORTRAN;
  BEGIN
  END

```

TERM

```

*****
*
* DATE:                10 October 1985
* VERSION:             1.0
* NAME:                loData
* MODULE NUMBER:       8
* DESCRIPTION:
*
* This compool defines all data required for the JOVIAL
* routine RefMat and 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 loData;

```

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

```

```

DEF TABLE ItsTable(0) M 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 M;
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 RcdTypeI S 15;

```

DEF ITEM RedTyp C 2;
OVERLAY RedTyp1: RedTyp;

OVERLAY Infil, OutFil, Filnam, Header, ItsRed, OutFid, Eof, RedTyp;

TERM

```

*****
*
* 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 lx          U 8;
DEF ITEM ly          U 8;
DEF ITEM WdsInAcd     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(8,0);
  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

Appendix D

```

$ | ASH1730 -- Assemble a 1730 source module
$ |
$ | 0ASH1730    file
$ |
$ |              file = input source name of module    file.SI
$ |
$ | Create Assembler input file UI that designates M11-Std-1730A as the target
$ | rather than the alternate 1730A target
$ |
$ CREATE 'P1'.UI
$ ASSEMBLE    TARGET=M1730A
$ TY 'P1'.UI
$ |
$ ASSIGN 'P1'.UI          UI | UPDATE INPUT COMMANDS FILE (INPUT)
$ ASSIGN 'P1'.SI          SI | 1730A 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_1730A OI | LIBRARY INPUT
$ |
$ SET VERIFY
$ M1730A
$ |
$ DERASSIGN SI
$ DERASSIGN UI
$ DERASSIGN OO
$ DERASSIGN SO
$ DERASSIGN LO
$ DERASSIGN 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'

```

```

$ | LINK1730 — Link one or more 1730A target object modules.
$ |
$ | @LINK1730    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_1730A OI         | LIBRARY OBJECT FILE (INPUT)
$ |
$ | ITS LINK                | RUN 1730A 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:RME,GROUP:RM,WORLD:RME)/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
$ | SO                :== 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 JOUAL LIBRARY FOR AUTOMATIC SEARCHING FOR VAX TARGET
$ |
$ | ASSIGN JOULIBU:JOULIBU.OLB LNK$LIBRARY
$ |
$ | The following defines the 1750A support tools pseudo-commands:
$ |
$ | LINK30A :== LINKITS
$ | RAIDX  :== $TOOLS:RAID
$ |
$ | END LOGIN.COM
$ |
$FINISH:
$ | EXIT

```

Appendix E

```

with TEXT_10;
use TEXT_10;
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_10 is new INTEGER_10 (integer);
use INT_10;

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_10 is new INTEGER_10 (integer);  
use INT_10;  
package FLT_10 is new FLOAT_10 (float);  
use FLT_10;
```

```
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_10 is new INTEGER_10 (integer);  
use INT_10;  
package FLT_10 is new FLOAT_10 (float);  
use FLT_10;
```

```
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_10 is new INTEGER_10 (integer);  
use INT_10;  
package FLT_10 is new FLOAT_10 (float);  
use FLT_10;
```

```
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 —Tchebyshev Economization.

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

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

COMPUTE_POWERS_OF_TCHEBYSHEF;
new_line;
put ("Powers of Tchebyshev");
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;

—Accumulate 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_10; use TEXT_10;

package TCHEBYSHEF_PACKAGE is

```

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

```

—Unconstrained type declarations

type MATRIX is array (integer range <>, integer range <>) of integer;

—Matrix of integer values, used to contain the Tchebysheff polynomials

type FLOAT_MATRIX is array (integer range <>, integer range <>) of float;

—Matrix of floating point values, used to contain the powers of
— Tchebysheff

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 Tchebysheff 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_10 is new INTEGER_10(integer);
use INT_10;

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;

```

```

and 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_10 is new INTEGER_10<INTEGER>;
  use INT_10;

  package FLT_10 is new FLOAT_10<LONG_FLOAT>;
  use FLT_10;

  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_CHK;

      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(1);
        put(" ==> ");
        put(C<DEN,1>);
        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: MATRIX;

A: PADE_MATRIX;

D: VECTOR;

M: integer;

K: integer;

N: integer;

MACLAURIN: VECTOR;

COEFFICIENT: string(1..33);

EPS: LONG_FLOAT;

C: MATRIX;

end GLOBAL_DATABASE;

- Matrix containing the coefficients of
- different powers of Tchebyshev polynomials
- Used to contain the series of PADE approx
- $R(S, N \text{ or } D, C)$
- 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
- Error values of PADE approximations
- Power of the numerator polynomial
- Power of the denominator polynomial
- Power of the initial power series
- Contains the coefficients for the
- different powers of "X" for the power
- series expansion of a function
- Used to contain user entered coefficients
- a power series expansion
- Convergent epsilon
- Final rational approximation

--
 -- 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_10;          use TEXT_10;
package body COMMON_PROCS is

    package INT_10 is new INTEGER_10(integer);
    use INT_10;

    package FLT_10 is new FLOAT_10(LONG_FLOAT);
    use FLT_10;

    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 convergance.");
            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 separate
  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(LONG_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(N,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 := N + 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(N,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 := N + 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(N,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 := N + K + 2;

— Compute the initial approximating polynomial

for l in 0..25 loop

MACLAURIN(l) := 0.0;

end loop;

for l in 1..((N+1)/2) loop

MACLAURIN(l*2-1) := PRODUCT(1, ((1-2)*2+1),2) /

PRODUCT(2, (1*2-2),2) *

LONG_FLOAT(1*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(N,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 := N + 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(N,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 := N + 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;

AD-A163 985

DEVELOPMENT OF A RUN TIME MATH LIBRARY FOR THE 1750A
AIRBORNE MICROCOMPUTER(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
S A HOTCHKISS DEC 85 AFIT/GCS/HA/85D-5

3/3

UNCLASSIFIED

F/G 9/2

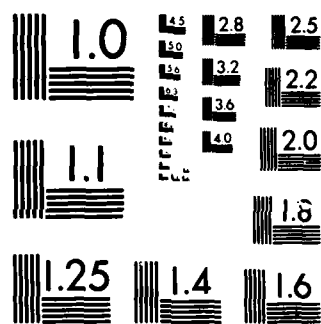
NL



END

FILED

DTIC



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(N,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 := N + 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. Hetchkiss 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_CHK;

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
      R(SERIES, NUM_DEN, COEFFICIENT) := 0.0;
    end loop;
  end loop;
end loop;

for I in 0..N 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;
      end loop SETUP;
    end if;
  end loop;
end loop;

```

```

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(a,k)
--  $D(I+J+1) = \sum_{L=0}^J (\text{MacLaurin}(I+J+1-L) * B(L))$ 
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;

and COMPUTE_PADE_APPROXIMATIONS;

```

procedure COMPUTE_CHK is

A: Integer := 0;
B: Integer := 1;
LANDA: VECTOR;

begin

-- Compute the Landas (alpha=1)

LANDA(0) := -(D(N+K+1) * T(N+K,0)) / (2.0**<N+K>);

for J in 0..<N+K-1> loop

if D(J+1) /= 0.0 then

LANDA(J+1) := (D(N+K+1) * T(N+K+1,J+1)) / (<2.0 **<N+K>> * D(J+1));

else

LANDA(J+1) := 0.0;

end if;

end loop;

-- Load Pa(X) and Qa(X) with their A and B coefficients respectively

for I in 0..N loop

C(A,I) := R(N+K,A,I);

end loop;

for I in 0..K loop

C(B,I) := R(N+K,B,I);

end loop;

-- Compute coefficients "A" of numerator and "B" of denominator

for J in 0..<N+K-1> loop

for K in 0..25 loop

R(J,A,K) := R(J,A,K) * LANDA(J+1);

C(A,K) := C(A,K) + R(J,A,K);

R(J,B,K) := R(J,B,K) * LANDA(J+1);

C(B,K) := C(B,K) + R(J,B,K);

end loop;

end loop;

C(A,0) := C(A,0) + LANDA(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_CHK;

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

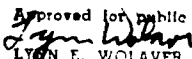
Captain Steven A. Hotchkiss was born on 8 February 1952 in Eureka, Kansas. In May of 1970, he graduated from High School at Smith Center, Kansas. He later entered the Navy as an Aviation Antisubmarine Warfare Operator, and earned his Naval Aircrew Wings in November of 1974. After being released from the Navy in January of 1977, he attended Kansas State University from which he received the degree of Bachelor of Science in Computer Science. Upon graduating in May of 1980, he received a commission in the USAF through the ROTC program. He was stationed at Beale AFB, California as a Missile Warning System Programming Officer, and was responsible for the maintenance and development of tactical software for the phased array sea-launched ballistic missile early warning system-- PAVE PAWS. While stationed there, he also attended classes in a University of Southern California residency program, and in December of 1983 he received the degree of Master of Science in System Management. He then entered the School of Engineering, Air Force Institute of Technology, in June of 1984.

Permanent address: 829 Prospect Street
Osage City, Kansas 66523

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

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-5			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.						
			<table border="1"> <tr> <td>PROGRAM ELEMENT NO.</td> <td>PROJECT NO.</td> <td>TASK NO.</td> <td>WORK UNIT NO.</td> </tr> </table>			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.						
11. TITLE (Include Security Classification) See Box 19									
12. PERSONAL AUTHOR(S) Steven A. Hotchkiss, M.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 201					
16. SUPPLEMENTARY NOTATION									
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)						
FIELD	GROUP	SUB. GR.							
12	01		Functions (Mathematics), Approximations,						
09	01		Computer Programs, MIL-STD-1750A						
19. ABSTRACT (Continue on reverse if necessary and identify by block number)									
Title: DEVELOPMENT OF A RUN TIME LIBRARY FOR THE 1750A AIRBORNE MICROCOMPUTER Thesis Chairman: Panna B. Nagarsenker Associate Professor of Mathematics and Computer Science									
Approved for public release: LAW AFR 190-11  LYNN E. WOLAVER 16 JAN 86 Dean for Research and Professional Development Air Force Institute of Technology (AFIT) 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 circular trigonometric functions and their inverses. In addition, the steps required for the performance analysis of the math library have been outlined.

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

END

FILMED

3-86

DTIC