Ground Operations Aerospace Language

GOAL

Final Report

Volume I

Study Overview

Contract NAS 10-6900

(NASA-CR-136781) GROUND OPERATIONS AEROSPACE LANGUAGE (GOAL). VOLUME 1: STUDY OVERVIEW Final Report (International Business Machines Corp.) 31 p HC $3.75 CSCL 09B G3/08

N74-15887

Unclas 28965

31 July 1973

Ground Operations Aerospace Language

GOAL

Final Report

Volume I

Study Overview

Contract NAS 10-6900

Approved by: J. W. Handley

J. W. Handley, Manager
Systems Programming and
Advanced Programs

ABSTRACT

A series of NASA and Contractor studies sponsored by NASA/KSC resulted in a specification for the Ground Operations Aerospace Language (GOAL). The Cape Kennedy Facility of the IBM Corporation was given the responsibility, under existing contracts, to perform an analysis of the Language Specification, to design and develop a GOAL Compiler, to provide a specification for a data bank, to design and develop an interpretive code translator, and to perform associated application studies.

This report documents the results of the IBM tasks.

VOLUME I STUDY OVERVIEW

TABLE OF CONTENTS

VOLUME II COMPILER

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

VOLUME III DATA BANK

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

VOLUME IV INTERPRETIVE CODE TRANSLATOR

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

VOLUME V APPLICATION STUDIES

TABLE OF CONTENTS

VOLUME V APPLICATION STUDIES

TABLE OF CONTENTS (Cont)

1.0 INTRODUCTION

This document identifies and describes the results of tasks performed during the Phase II Ground Operations Aerospace Language Development and Applications Studies. These efforts were performed during approximately the first half of 1973 and were preceded by Phase I, conducted during 1972. The principal Phase II efforts involved the refinement and extension of basic Phase I capabilities and performance and analysis of selected application studies.

1.1 BACKGROUND

The Phase I activities primarily involved the development of a basic capability for the compilation, translation and execution of GOAL programs. This system enabled preliminary verification of GOAL language concepts and provided a demonstration of the use of GOAL in simulated checkout applications. Following is a brief description of the Phase I systems.

1.1.1 Compiler

This system is used in conjunction with the GOAL Data Bank to validate the user's source statements, produce compilation output listings and generate an 'Intermediate GOAL' data file. This file contains essential information derived from the user's source program and Data Bank.

1.1.2 Translator

This system is used to convert the 'Intermediate GOAL' data file to a format suitable for execution in real time. This was accomplished by translating the intermediate data to FORTRAN which was then processed by a standard FORTRAN compiler to generate the executable codes.

1.1.3 Real Time Executive

The real time executive of the Console Operators Training System (COTS), at the IBM Cape Kennedy Facility was modified and extended to support the execution of the translated GOAL programs. This enabled the demonstration of GOAL programs in checkout applications using simulation models of actual hardware systems.

1.1.4 Principal Phase I Products

- A basic Compiler/Data Bank which would process most of the GOAL Language statements and their variations.
- A FORTRAN Translator which enabled the execution of GOAL programs in a simulation environment.

1.2 PHASE II

Following Phase I, the GOAL Language specification was extended and refined. The need for a more general translator was determined and several applications areas were identified for further analysis. These factors led to the Phase II effort and definition of the principal Phase II objectives which are described in the following section.

2.0 OBJECTIVES

The IBM effort documented in this report was accomplished in two phases. Phase I period of performance began 1 November 1971 and was in effect through September 1972. Although no formal objectives were derived, a series of discussions between IBM and NASA LV-CAP resulted in a Study Plan which delineated the following tasks:

- Perform an analysis of the GOAL specification
- Identify GOAL to ATOLL implementation effectivity
- Analyze host computer candidates and the language to be used in writing the Language Processor
- Design the Language Processor
- Develop the Language Processor
- Develop the Language Converter
- Provide a demonstration of GOAL to ATOLL translation

As stated previously in the report (Section 1.0), there were changes to these tasks during the period of performance. The most notable of the changes was the substitution of a FORTRAN format rather than ATOLL for the Translator output. This change led to the subsequent demonstration being conducted on the System/360-40 and the Console Operating Training System (COTS) located at the IBM Facility.

Phase II of the GOAL Study was begun in February 1973. As in the case of Phase I, formal objectives were not stated; instead, three basic tasks were delineated. These tasks would incorporate the latest language specification and provide a more generalized software package. The tasks defined were:

- Design and implement a Translator which converts compiler output to a generalized interpretive format
- Update the GOAL Compiler to provide full support of the Language Specification
- Apply the GOAL Language in selected operational applications and develop techniques for further improvement of the Language.

With the exception of minor changes to subtasks, these are the items which were addressed during this phase to provide the results documented in this report.

3.0 TECHNICAL APPROACH

A basic ground rule in the technical approach taken for Phase II was to extend the Phase I system in such a way as to take maximum advantage of its existing capabilities. Thus, The Translator Development, Compiler Updates, and Applications Studies were performed using equipment located at the IBM Cape Kennedy Facility as in Phase I. As directed by NASA, use was made of FORTRAN in Phase II tasks requiring the development of software. The approaches taken in the performance of the indicated tasks follow.

3.1 DESIGN AND DEVELOP INTERPRETIVE CODE TRANSLATOR

As in the Phase I, input to the Translator is the 'Intermediate GOAL' data file generated by the Compiler. In addition, the compiler symbol table data is made available to the translator. These two sources of data are used to produce the interpretive data output file. This file is generated in accordance with user options specified by control cards which are also input to the translator. Output listings are generated which indicate and identify the content and location of all data items included in the Translator output file.

- o Maximum retention of data captured by the GOAL compiler.
- o Compact representation of data.
- o Minimum processing requirements for real time executive.
- o Standard formats for frequently occurring data structures.

3.2 GOAL COMPILER UPDATES

As in Phase I, the syntax equation technique was used to extend the basic compiler capabilities. The compiler updates are based on the GOAL Syntax Diagrams Handbook dated April 16, 1973. Five new statements were added to the compiler capability and about twenty modifications were made. Additional compiler routines were provided to support the new statements and existing routines were modified as required. Other compiler related modifications include the brief form processing capability, free form data bank input processing, and punched deck output. The level of compiler diagnostics was significantly increased.

3.3 APPLICATION STUDIES

A series of discussions was held with LV-CAP to determine areas of study which would enhance the applicability of GOAL. The following general areas were selected:

- o Use of Engineering Units
- o Subroutine Parameter Validation
- o Abbreviated Forms of GOAL
- o Conversion of Existing Automated Procedures

Each of these areas was analyzed in detail and methods of implementation were recommended.

3.3.1 Engineering Units

Three levels of usage/processing were considered:

- o Use as labels for readability only
- o Automatic scaling to standard set of units
- o Usage validation by compiler

A recommended system of engineering units is provided and processing techniques are described for scaling and validation.

3.3.2 Subroutine Parameter Validation Analysis

The need for validation and methods of accomplishing it are discussed relative to the subroutine originator and the subroutine user. A recommended technique involving the use of the Data Bank is described.

3.3.3 GOAL Brief Form

An alternate set of keywords was selected to reduce the amount of writing required to generate GOAL statements. This set was selected to provide a parsable set free of ambiguities resulting from a single keypunch error. The brief form keywords are expanded to standard GOAL by the updated version of the compiler.

3.3.4 Automated Procedure Conversion

The test programs IATS, IAED, and GEO1 were converted to the GOAL language. Listings are provided for the converted GOAL programs and a commentary is given relative to the conversion process.

4.0 TECHNICAL SUMMARY

4.1 SUBSYSTEMS

4.1.1 GOAL Compiler

The GOAL Compiler encompasses that portion of the GOAL System which converts symbolic procedure statements into an intermediate text format. This section will provide a summary description of the components of the GOAL Compiler. Volume II provides a detailed description of the principal elements and functions which were implemented according to specifications provided by NASA/KSC in publications TR-1228, Ground Operations Aerospace Language (GOAL) Textbook and TR-1213, Ground Operations Aerospace Language (GOAL) Syntax Diagrams Handbook both dated 16 April 1973.

The principal functions of the GOAL Language Compiler are:

1. To accept a GOAL program as input on punched cards.
2. To parse GOAL statements according to syntax diagrams.
3. To generate diagnostic messages for statement errors.
4. To generate output listings and reports.
5. To generate Intermediate GOAL (object) data.

To support these functions the following principal software elements are provided:

1. Syntax specifications (equations) for GOAL Language.
2. Syntax processor.
3. Compiler input processor.
4. Parsing routines.
5. Diagnostics routine.
6. Output report generator.
7. Intermediate GOAL data generator.

The relationship between these items is shown in Figure 4.1.1-1

Syntax Specifications

The GOAL Language is specified by a set of syntax diagrams which define all variations of GOAL statements. In accordance with the design approach for the parsing routines, these diagrams are transcribed into modified Backus Naur Form.

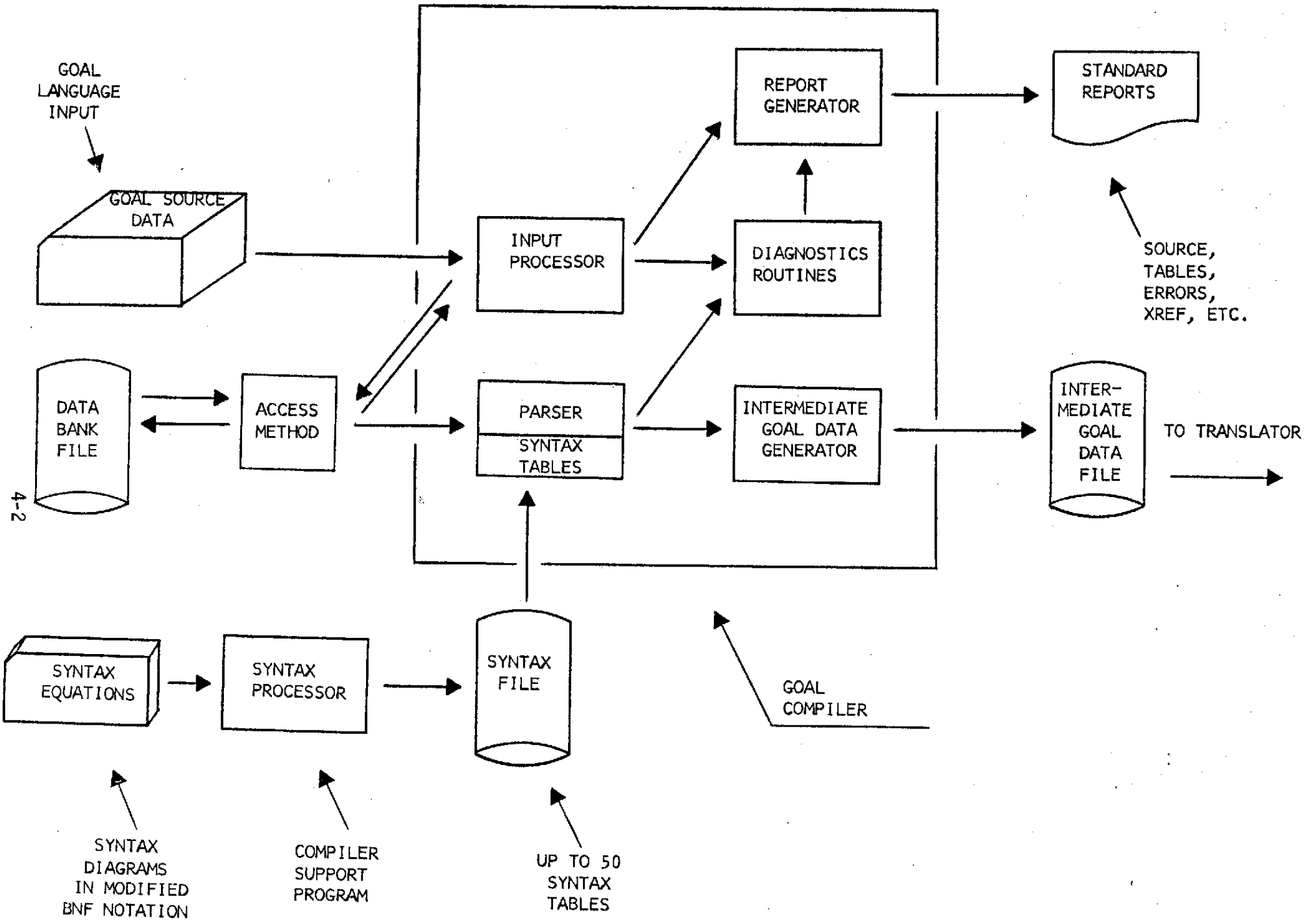

GOAL
LANGUAGE
INPUT
GOAL SOURCE
DATA
INPUT
PROCESSOR
REPORT
GENERATOR
STANDARD
REPORTS
DIAGNOSTICS
ROUTINES
SOURCE,
TABLES,
ERRORS,
XREF, ETC.
DATA
BANK
FILE
ACCESS
METHOD
PARSER
SYNTAX
TABLES
INTERMEDIATE
GOAL DATA
GENERATOR
INTER-
MEDIATE
GOAL
DATA
FILE
TO TRANSLATOR
SYNTAX
EQUATIONS
SYNTAX
PROCESSOR
SYNTAX
FILE
GOAL
COMPILER
SYNTAX
DIAGRAMS
IN MODIFIED
BNF NOTATION
COMPILER
SUPPORT
PROGRAM
UP TO 50
SYNTAX
TABLES

Figure 4.1.1-1.

Modified BNF statements are processed by the GOAL SYNTAX GENERATION PROGRAM to generate syntax tables which are subsequently used by the GOAL Compiler to parse GOAL Language statements.

The set of statements describing the GOAL syntax equations are arranged as a group. The order is not important, however, the last statement must be 'END'.

Each statement may use up to 20 cards. The end of statement is indicated by a semicolon, ';'.

Each statement, (except 'END'), defines a syntactical element of the GOAL Language. This element may be referenced by other statements. The symbolic name of the syntactical element being defined appears on the left of the character, '=', in the Modified BNF statement. Each element including the 'ROOT' element must be defined ONCE in the input group. The 'ROOT' element is the top of the 'syntax tree'. All valid GOAL statement variations can be derived starting with the 'ROOT' element.

Each definition statement is in either SEQUENTIAL or ALTERNATE form. The sequential form indicates that all items specified on the right of the character '=' must be processed or identified to satisfy the definition of the syntactical element associated with that statement. The alternate form indicates that any item identified on the right of the character, '='. will satisfy the definition. In both cases comparison proceeds from left to right. Items in the alternate form are separated by the character, '|'. Sequential and alternate forms cannot be mixed in the same statement.

Syntax Processor

The syntax processor is a stand-alone program which accepts input in the form of syntax equations and converts them to syntax tables which are stored in the Syntax Table File for subsequent use by the GOAL Compiler parsing routines. This relationship is shown in Figure 4.1.1-2. The Syntax File may contain up to 50 different syntax tables of moderate size. This enables the use of language subsets or experimental versions of a language syntax.

The syntax processor employs a basic parsing routine and a special syntax table which is generated by the syntax file initialization program. In general, the initialization program need only be used once. The initialization program also reads in a character set record which is used by the GOAL Compiler to identify all letters, numerals, and symbols used in the GOAL Language. This technique enables functional substitution of one character for another, (in compiler input), without regeneration of the GOAL Compiler itself.

The operation of the syntax file initialization program and the syntax processor is shown in Figure 4.1.1-2.

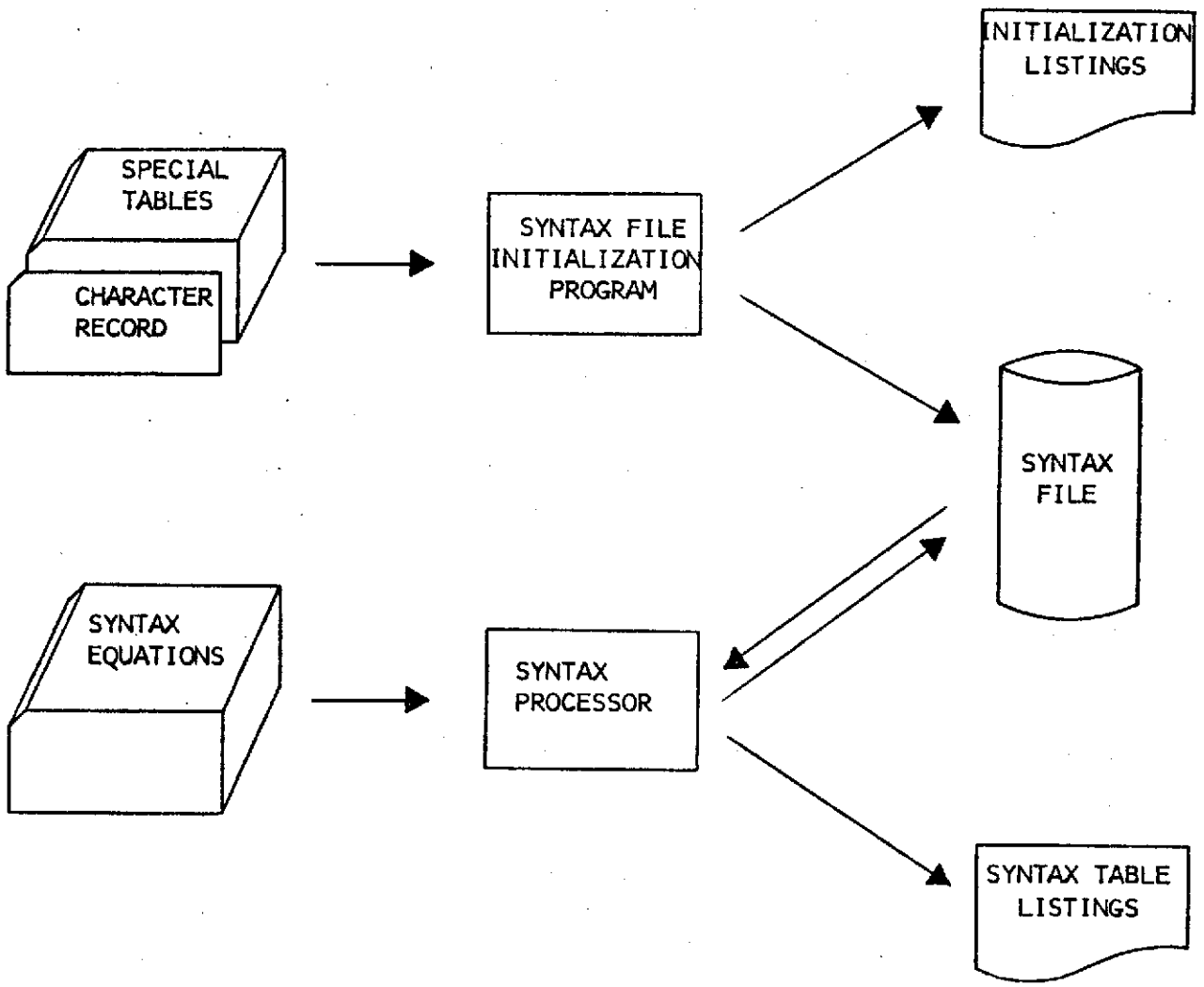

INITIALIZATION
LISTINGS
SPECIAL
TABLES
CHARACTER
RECORD
SYNTAX FILE
INITIALIZATION
PROGRAM
SYNTAX
FILE
SYNTAX
EQUATIONS
SYNTAX
PROCESSOR
SYNTAX TABLE
LISTINGS

Figure 4.1.1-2.

Parsing Routines

The GOAL Compiler will utilize a table guided top-down parsing algorithm. The parsing routines are of two types:

1. General purpose - These routines form the basic parser. They are used by all syntax tables. They perform the input statement scan according to the top-down technique, flag unrecognizable constructions, and cue the execution of action routines according to the structure of the input statement.

2. Special purpose - These are the action routines which provide processing to support recognition and testing of specific syntactical elements such as labels, variables, and macros.

The primary function of the general purpose parsing routines is to scan statements in the input stream and recognize permissible constructions according to the contents of the syntax tables. The recognition criterion is simple appearance. As constructions are recognized, the parse continues until the entire statement has been processed. If a construction is not recognized, alternate definitions are tested. If none is satisfied, the invalid field is marked and a diagnostic message is given according to the last error checkpoint processed from the syntax tables. When a construction is successfully identified in the input stream, these routines may cue the execution of a special purpose action routine specified in the syntax tables.

The special purpose parsing routines are the action routines cued by the general purpose parsing routines. They may perform any of the following types of functions.

1. Specialized compiler support such as macro definition and expansion.

2. Symbol table operations for definition and reference.

3. Usage validation for any syntactical element of the GOAL Language.

The special purpose parsing routines may signal a no-compare condition to the basic parser. In this case alternate definitions will be tested or a diagnostic message will be given. In this way the special purpose routines may resolve the difference between syntactical elements which have similar appearance but different meanings.

Compiler Diagnostics

Two basic types of errors are recognized in the input statements to the GOAL Compiler. These are:

1. Syntax Errors - The appearance of the statement does not conform to any permissible variation described in the GOAL Language Syntax diagrams. In this case the parse is terminated for the current statement.

2. Usage Errors - The statement is syntactically correct, however, some valid construction is incorrectly used. In this case the parse may continue to process the remainder of the current statement.

In both cases the statement is flagged in the expanded source statement listing. A mark is placed under the field in which the error occurred and all relevant data is logged for use in the diagnostic summary report.

Compiler Output Reports

The following reports will be provided, on request, by the GOAL Compiler.

1. Source Listing

2. Expanded Statement Listing

3. Statement Label Cross-Reference

4. Symbolic Name Cross-Reference

5. Function Designator Summary

6. Diagnostics Summary.

These reports may be requested using compiler directives described in Volume II, Section 2.6.7.

Intermediate GOAL Data Generator

The intermediate text output from the compiler is a data set which represents in a tabular fashion all of the informational content of the GOAL source program. This data set is sequentially processable and it contains logical records of varying length. Each logical record consists of a fixed header portion followed by a varying length data portion. Total record content will be such that it can be read, written, and processed in a FORTRAN array of the INTEGER type. Each element of the array is capable of storing a signed number or a single character.

Detailed formats are given in Volume II, Section 3.2.2.

4.1.2 Data Bank

To insure compatibility with a maximum number of applications, a systematic and error-free method of referencing command/response (analog and digital) hardware measurements is a principal feature of the GOAL language. Central to the concept of requiring the test language to be independent of launch complex test equipment and terminology is that of addressing measurements via symbolic names that have meaning directly in the hardware units being tested. To form the link from test program through test system interfaces to the units being tested the concept of a data bank has been introduced. The data bank is actually a large cross-reference table that provides pertinent hardware data such as interface unit addresses, data bus routings, or any other system values required to locate and access measurements.

Three aspects of future aerospace operations can be considered key when determining whether single or multiple data bank(s) can be justified. First, the management of a very large number of sensor based measurements at distinct facilities constitutes a large engineering management effort. Secondly, verification and launch checkout of future ground and flight hardwares will require use of this large measurements base in conjunction with the generation of a large body of automation and test program software packages. A third problem area is then created when combining certain aspects of the first two - when hardware modifications are required as a result of normal or abnormal operations these changes must be carried over into the software programs also. The solutions to any or all of these problems are further degraded when confronted with the potential space shuttle environment of tight launch schedules involving multiple vehicles.

Little can be accomplished in alleviating the problems of the large measurements base or the large number of required test programs as these items are controlled by engineering factors not under direct control of ground data processing systems. Data processing equipments can be effectively used in the support of the ground checkout and launch "business" just as they have been in other applications of the scientific and commercial world. The GOAL test programming language has evolved as an attempt at alleviating the problem of generating and managing a large number of test programs. The use of a data bank in conjunction with the GOAL language is vital since its sole purpose is to assist the test procedure writer in dealing with a large and variable set of measurements and to make as easy as possible the introduction of the effects of hardware changes into the test programming system.

The disadvantages of dealing directly with the hardwares of test system interfaces were well known when specifications were first drafted for GOAL. The conceptual use of a data bank as a repository for all sensor-base hardware procedural, addressing, and routing data accompanied the

first implementation of GOAL. In the original version, the data bank took the form of a disk-resident file from which the GOAL compiler could randomly access data relative to all available measurements. GOAL also introduced a meaningful symbolic naming scheme for all hardware measurements and classifies such devices as "function designators." As an illustration, the discrete signal that activates/deactivates 28 volt stage power might have been addressed in the SATURN/APOLLO software via its hardware address, e.g., DISCRETE OUTPUT #414. The same measurement could be addressed in a GOAL program by the sybolic function designator name "28V STAGE POWER." Using the function designator name as a search argument, the GOAL compiler would query the data bank during compilation and determine that a discrete output measurement at hardware address 414 was being referenced. This symbolic name usage and the automatic lookup and resolution of system hardware dependent data eliminates many of the frustrations of accommodating changes to the hardware systems. All such changes are introduced to the GOAL system by an update to the data bank; recompilation of the appropriate test program will then propagate the measurement changes into the software.

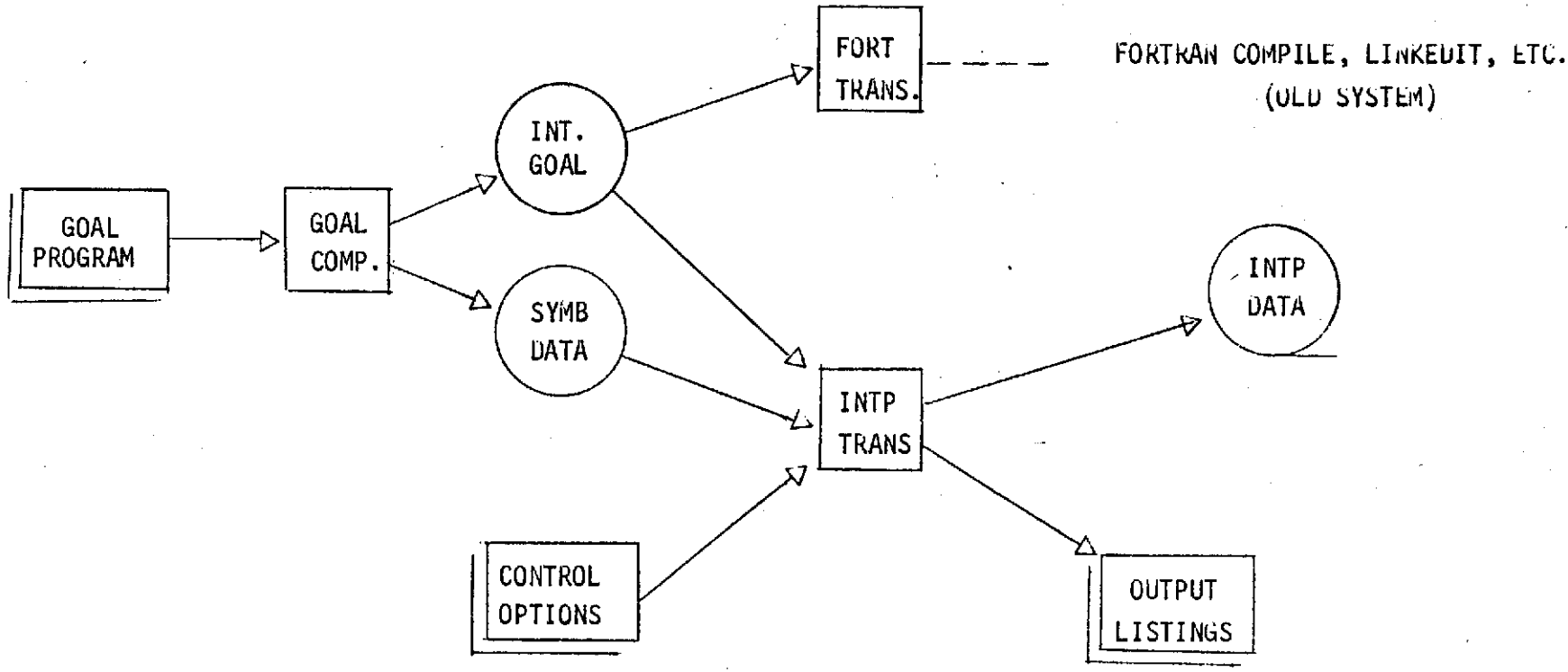
4.1.3 Interpretive Code Translator

The principal function of the Interpretive Code Translator is to convert the 'Intermediate GOAL' and 'SYMBOL TABLE' data files generated by the compiler into a highly general binary format which is suitable for interpretive execution.

The relationship between the Interpretive Code Translator and the GOAL Compiler/Data Bank system is shown in Figure 4.1.3-1. Note that significant differences between the Interpretive Translator and the Phase I FORTRAN Translator include the processing of symbolic data captured by the compiler and the provision of user options to control the contents and format of the output interpretive data.

The control options provided by the Translator program enable the user to specify the word size, character set, characters per word, and words per numeric value relative to the target real time system. The user may also select the number of target machine words to be included in each block written on either seven or nine track tape. The inclusion of symbolic names tables in the interpretive data is also optional.

The Interpretive Code Translator Program provides output listings which identify and describe the contents and location of all items contained in the output data. The use of the interpretive data is discussed in Volume IV, Appendix A, Interpretive Code Guideline.



GOAL INTERPRETIVE CODE TRANSLATOR

FIGURE 4.1.3-1

4.2 Integrated System

The relationship between the previously described components of the system is summarized in Figure 4.2-1. Following is a brief description of the principal functions of the system elements identified in this diagram:

Syntax Data - The logic of GOAL statements as described in the syntax diagrams is converted to modified BNF and transcribed to punched cards for input to the Syntax Generation Program.

Syntax Generation - The modified BNF syntax equations are validated and converted to tabular format for storage in the Syntax File.

Syntax File - This file may contain up to fifty different syntax tables. These are identified by an assigned number ranging in value from 1 to 50.

Data Bank Input - A list of test and control points is obtained for the system under test. For each measurement the name, type, and hardware address are transcribed to punched cards for input to the Data Bank Update program.

Data Bank Update - The measurements input data is sorted and entered in the Data Bank file. A directory is created to enable efficient retrieval of the data using the measurement name as a keyword.

Data Bank - This file may contain up to thirty individual 'GOAL program Data Banks'. These are used by the compiler to validate references to measurements in the GOAL Language statements.

GOAL Program - System test requirements and criteria are converted to a test procedure written in the GOAL Language. This procedure is transcribed to punched cards for input to the GOAL Compiler.

GOAL Compiler - The input GOAL program is validated according to the syntax equations and measurement definitions used for the particular compilation. Output listings include GOAL statements, cross-reference tables, and diagnostics. The Intermediate GOAL and Symbol table files are generated for input to the Interpretive code Translator.

Translator Control Cards - The user determines the required word size, character set, block size, output tape, and optional tables required in his real time application. These requirements are transcribed to punched cards for input to the Interpretive Code Translator.

Interpretive Code Translator - The Intermediate GOAL and Symbol table data from the compiler are processed according to the specified user control options to generate the Interpretive Data output tape. Listings are generated which identify and describe the location and content of all output Interpretive Data.

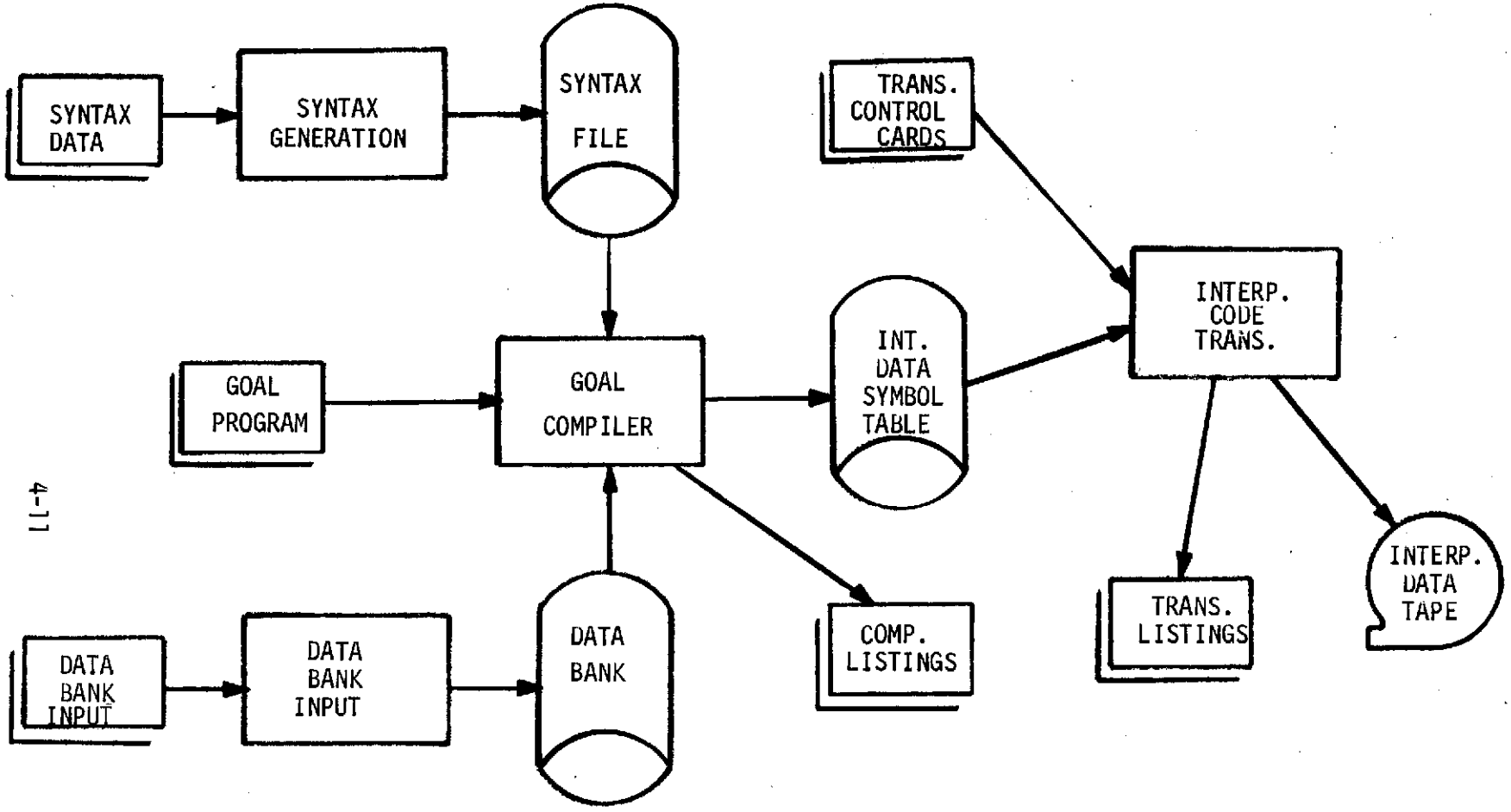

SYNTAX DATA
SYNTAX GENERATION
SYNTAX FILE
TRANS. CONTROL CARDS
INTERP. CODE TRANS.
GOAL PROGRAM
GOAL COMPILER
INT. DATA SYMBOL TABLE
DATA BANK INPUT
DATA BANK INPUT
DATA BANK
COMP. LISTINGS
TRANS. LISTINGS
INTERP. DATA TAPE

FIGURE 4.2-1

Interpretive Data Tape - The user may select either a seven or nine track tape to contain the output interpretive data generated by the translator. This tape is written in binary mode with the specified number of target machine words per record.

Translator Listings - The listings generated by the translator identify all user specified options and identify all information contained in the output tape. Most of the listings are optional and may be suppressed via user control option.

4.3 SPECIAL STUDIES

4.3.1 GOAL Short Forms

GOAL is a high level test engineer oriented language encompassing subsystem and system testing. It is compatible with a variety of engineering disciplines and preflight operations.

The GOAL vocabulary selected provides a high degree of readability and retainability. These highly desirable benefits are achieved through a burden placed on the procedure writer. This burden is simply the length and number of words and phrases which appear in the GOAL syntax. This realization led to a study to provide the test engineer more economical means of writing GOAL statements, while retaining the benefits of readability and retainability.

The first step in the study was to determine the feasibility of modifying the syntax equations in such a manner that they could accommodate alternatives to the GOAL words and phrases. The syntax equations were analyzed and each embedded word or phrase (text) was identified. Then each word and phrase was assigned a symbolic name. The syntax equations for each of these symbolic names were written such that optional alternates could be parsed. An alternative vocabulary was developed using acronyms and mnemonics. A ground rule for development of the alternative vocabulary was, no single keypunch error in an alternate would produce a parsable statement. The alternative vocabulary was written into the GOAL vocabulary equations and a Syntax Table generated. A GOAL test procedure was then written and compiled to test the vocabulary syntax equations. The testing provided grounds for modifications to the alternative vocabulary and the syntax equations. At this point it was clear that a short form would provide a more economical means of writing GOAL statements. It was also clear that each user group could develop a dialect suitable for their own needs.

The next step was to provide the procedure writer with the capability of automatically translating any dialect to the standard GOAL language. To achieve this end a subroutine was written to operate under control of the vocabulary syntax equations and the parser. This subroutine enables the user to convert a test procedure written in a dialect to the standard GOAL language. The implementation of the dialect-to-GOAL converter has further enhanced the utility of GOAL.

The subject of implementing an alternative vocabulary is discussed in VOLUME V, Section 2.

4.3.2 Engineering Units

The purpose of this study was to identify the general problems associated with using engineering units in the GOAL language. In order to accomplish this task, a candidate set of units was identified that would support operations involving electrical, mechanical, and thermal terminology. Both English and Metric systems units were considered in the derivation of a method selected for possible implementation of engineering units.

The method recommended for implementation of engineering units is one in which each unit is expressed as a combination of five fundamental quantities. The fundamental quantities are length, mass, time, permeability of a vacuum, and temperature (l, m, t, μ, θ). By expressing each unit in this manner, it will be required that all units be converted to a common internal working system. The working system recommended is the MKS System. By using the MKS system and the five fundamental quantities, input/output scaling operations can be automatically performed and unit usage can be validated.

The study concludes that implementation of engineering units is possible and recommends that the five fundamental quantities and the MKS System of units be used.

A three phase implementation plan is identified. The results of the study are in Volume V, Section 3.0.

4.3.3 Subroutine Parameter Validation

The purpose of this report was to identify validation requirements and techniques for passing parameters to subroutines.

Validation procedures are defined for subroutine writers, subroutine users, and for the configuration control group responsible for the System Data Bank. Implementation techniques are defined for the System Data Bank and the GOAL Compiler. The report is contained in Volume V, Section 4.0.

4.3.4 Selected Applications

The objective of the application analysis portion of this study was to verify the applicability and adaptability of the GOAL language to the launch vehicle checkout environment. Current automated checkout procedures were used for this study, extrapolating where necessary to project requirements for future applications. The method used to study these requirements was to manually convert these test programs to the GOAL language. The programs converted to GOAL included the Emergency Detection System (IAED) and the Terminal Count Sequence (IATS) which were written in the ATOLL language, and the GE01 platform alignment program, written in machine language for the RCA-110A computer.

The conversion of these programs to GOAL verified the basic design concepts of this language. Nearly all required tests could be performed with GOAL as it now exists. Principle features of the language, such as readability, ease of learning, and applicability to the engineering environment were also demonstrated. The study contains a summary of the various instructions used to make up the GOAL language, and comments on their applicability to test requirements.