NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Development of a CSSL Interface to GASP IV

by

Ahmed Saied Zaki Khadr

Submitted to the School of Graduate Studies

in partial fulfillment for the requirements of

the degree of Master of Applied Science

Department of Electrical Engineering

Faculty of Science and Engineering

University of Ottawa

Ottawa, Ontario

April 1981

IN THE NAME OF GOD MOST GRACIOUS MOST MERCIFUL

AND MY SUCCESS CAN ONLY BE FROM GOD

IN HIM I TRUST,.AND

INTO HIM I LOOK.

ACKNOWLEDGMENT

I wish to extend my thanks to a number of people who have helped directly or indirectly to shape and guide this thesis.

In particular to Dr. L. G. Birta, as my supervisor for his ever-present councelling, patience, invaluable suggestions and encouragement during the cource of this thesis.

To Dr. T. Oren for his helpful suggestions and advices before and while this work was carried out.

To Mr. M. Pepin for his assistance in programming.

To Mr. Z. Hasan and Mrs. B. A. Samad for their secretarial assistance in typing this document.

The financial support received through NSERC grant A8109 throughout the course of this research, is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

GASP IV is probably the most widely used of the combined simulation languages. Although it provides the required mechanisms for carrying out simulation studies of combined continuous/discrete systems, it suffers from an inadequate user interface. In particular, the user is obliged to specify the system model at the FORTRAN subprogram level.

This thesis describes a software system which has been designed and developed to overcome this user interface problem within a limited context; namely within the context of continuous system models. This system, called ACMED.L , is a high-level modelling and simulation specification language together with a processor which generates output code according to the structural requirements of GASP IV. The ACMED.L system thus makes it possible for the user to access the simulation mechanisms of GASP IV in a far more convenient way.

CHAPTER 1

Introduction

1.1 Preliminary Remarks

The GASP IV simulation language [1,2,3,4,5,6,7,8,9,10,11,12], referred to in this thesis simply as GASP, enjoys wide usage as a tool in simulation studies of discrete, continuous and combined system models. One of its distinctive, but unattractive, features is the relativily low level at which the user is obliged to specify the system model being studied; namely the FORTRAN subroutine level. The GASP user must, therefore, have some minimal working familiarity with FORTRAN and may be diverted from his central goal by the associated programming formalities. In addition the user has to respect the prescribed format of a set of data cards that must also be submitted with this program.

Although GASP provides a collection of sound and well-tested mechanisms for simulation studies, a better user interface is necessary in order to make these mechanisms more conveniently available. The thesis addresses this problem from the point of view of continuous system models and, in particular, describes an interface (ACMED.L) developed specifically for this subclass of system models. This development parallels to some extent to the work described in [13].

The ACMED.L preprocessor (Fig 1.1) is entirely transparent to the user since the generated subprograms and data cards are passed on to the GASP processor without any user intervention. If a job fails during the translation phase, a clear error message is printed out for the user and processing is terminated.

1.2 Some Noteworthy Features of GASP

GASP is a simulation language that can be used for discrete, continuous or combined simulation. It is a collection of FORTRAN subprograms that provide the user with a host of capabilities. These capabilities include features from simulation languages such as SIMSCRIPT [14] (next-event), CSMP [15,16] (numerical integration, plotting) and DYNAMO [17] (rates, levels, delays, table look-up functions). In fact, GASP has several distinctive features that make it particularly attractive as a simulation language; e.g.

(i) GASP is written in ANSI FORTRAN and hence requires no separate compiler. Because of the widespread availability of FORTRAN compilers, GASP can be executed on a great variety of machines.

(ii) It is well documented.

(iii) It can be modified and extended to meet the needs of particular applications.

(iv) GASP provides subprograms that accomplish the following functions:

1. Automatic time advance.
2. Event scheduling and control.
3. Continuous variable integration with variable step size (of fourth order and of the Runge-Kutta type) and user-specified accuracy tolerances.
4. Discrete-continuous interaction.
5. Statistical data collection.
6. Random deviate generation.
7. Program monitoring and error reporting.

8. Information storage and retrieval.

9. Automatic statistical computation and reporting.

10. Standardized simulation reports.

11. Tabular and plotted histograms.

12. Automatic plotting routines.

13. Flexibility in output reports and other provided functions.

Through the use of these subprograms, a GASP simulation model is developed. It includes the following:

(i) A set of event programs or continuous variable equations, or both, that describe the system's dynamic behaviour.

(ii) Various predefined data structures (vectors and matrices) for storing information.

(iii) An executive routine that directs the flow of information and control within the model.

(iv) Various support routines.

1.3 The Shortcomings of GASP

To justify the development of the new high-level language, ACMED.L, some of the shortcomings of GASP need to be examined.

From an analysis of any GASP program; e.g. Pilot Ejection Program given in [8] the following can be observed:

(i) The user has to become familiar with the large number of variables that exist in the GASP labeled common areas.

(ii) The user is obliged to copy a complete GASP labeled COMMON statement even if he is using only one of its variables; e.g.

```
COMMON/GCOM1/...
```

exists in MAIN because NPRNT, NCRDR are initialized there.

(iii) The user is obliged to write many statements that have no direct relevance to his problem; e.g.

```
INTEGER NSET(..)
COMMON QSET(..)
COMMON/GCOM1/..
EQUIVALENCE (NSET(1), QSET(1))
CALL GASP
```

(iv) The user has to develop his own mechanisms for handling conditional runs and functional relations defined by a finite set of points.

(v) Special output requirements such as print-plots and tables require the user to become familiar with the conventions for writing the GASP SSAVE subroutine and this subroutine's relationship to the GASP data cards (STA,PLO,VAR).

(vi) The data cards required by the GASP processor must be prepared with great care because of the rigid format conventions (See Appendix A).

(vii) In most model specifications, the statements are numerically meaningful only if they appear in a particular

sequence. Normally, this sequence can be mechanically established, and this feature can be exploited to facilitate the task of the user by permitting him to write the program statements in an arbitrary order and then using an algorithmic method to carry out the correct sequencing (the sorting operation).

In summary, the user has to concern himself with the following:

1 - GASP declaration statements.

2 - Initialization of certain GASP variables.

3 - Organizing his model in a predefined subroutine framework.

4 - Preparing formatted data cards and ensuring their correct order of appearance.

5 - Creating his own subroutines to handle conditional runs.

6 - Creating his own subroutines to handle functions defined by a finite set of data values.

7 - Correctly sequencing subprogram statements (sorting).

Accordingly, a simple user oriented interface could be valuable in facilitating the use of GASP and in widening its capabilities; e.g., a convenient conditional run capability and a data interpolation mechanism. These issues have provided the motivation for the development of ACMED.L.

It is of interest to note that ACMED.L can be viewed as a user-oriented interface software system. Such systems are widely used in diverse applications; other examples can be found in [18,19,20 and 21].

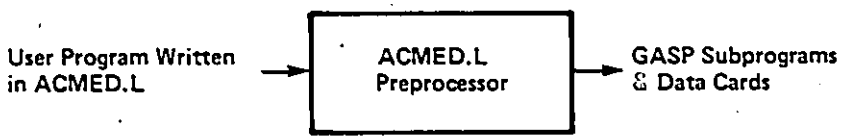


Fig 1.1 The ACMED.L Preprocessor

CHAPTER 2

The Structure of the ACMED.L. Language

ACMED.L is a CSSL-type continuous simulation language [22,23]. It has in particular, been modelled after the language constructs provided in CSMP [15,16]. The interested user can refer to these references for a more comprehensive discussion of the use of these constructs in continuous system simulation.

The basic elements of this language are the constants, symbolic names, operators, functions, and labels. A program consists of a set of statements that can be categorized into data, structure, translation control, and output control statements. These statements can be written anywhere in columns 1-72. An ACMED.L statement terminated by three dots is continued on the next card (card image).

2.1 The Basic Language Elements

2.1.1 Constants

The constants in ACMED.L are either numeric or boolean. The numeric constants are either integer or real. An integer constant is a number with one or more decimal digits without a decimal point and without embedded non-decimal characters; e.g. blanks, commas, ... etc. An integer constant can be positive, zero or negative. The following are valid integer constants:

0, +17, -9436, 157294

2.1.2 Reals

A real (floating-point) constant is a string of decimal digits with a decimal point. A real number may optionally be followed by a decimal exponent. With the exception of the character "E" for the decimal exponent, the decimal point and the algebraic signs, a real number must

not contain any embedded non-decimal characters. Real constants, like integer constants can be positive, zero or negative. The following are valid real constants:

0., 34.125, - 5934.75, 56.92E-10

The minimum and maximum values of the numeric constants are dependent on the digital computer on which this preprocessor is to be executed.

There are only two boolean constants; namely, .TRUE. and .FALSE. These are typically generated by, or used as inputs to logical or switching functions.

2.1.3 Symbolic Names

These are used to represent quantities whose value may change during the course of program execution; i.e. the problem variables. They also represent constants, procedure, subroutine names, ... etc. A symbolic name may contain from one to six alphanumeric characters (A through Z and 0 through 9). The first character must be alphabetic and none of the following special characters may be embedded in the name:

+ - * / = . ( ) ' $, or blank

A symbolic name must not be a reserved word in either ACMED.L, GASP or FORTRAN. The ACMED.L and GASP reserved words are listed in Appendix B. All symbolic names which represent variables, constants or functions are given, by default, the type REAL. This can be altered

by using either the FIXED (or INTEGER) translation control statement to establish the type INTEGER, or the LOGICAL statement to establish the type LOGICAL. Matrices and arrays must be declared using a FORTRAN DIMENSION statement or the ACMED.L STORAGE statement.

2.1.4 Operators

In ACMED.L the operators for the common arithmetic operations (addition, multiplication, exponentiation ... etc.) are the same as in FORTRAN. The FORTRAN rules for precedence also apply.

2.1.5 Functions

These are the functional blocks that perform the more complex mathematical operations such as integration, function generation, limiting, ..., etc. Table 2.1 summarized the functional blocks provided by ACMED.L. The general canonical form of each function is shown with non boolean inputs and outputs represented by X and Y, boolean inputs and outputs by L, and initial conditions by YZ and YDZ, respectively.

Both initial conditions and parameters must be either specified in the data statements (See section 2.2.1) or assigned a value in the INITIAL segment of the program (See section 2.2.2.1C)).

All functions available in the standard FORTRAN library of the user's system can also be treated as functional blocks thereby

supplementing the ACMED.L library. The user may also write his own FORTRAN FUNCTION or SUBROUTINE subprograms (See section 2.2.2.4b)).

2.1.6 Labels

The first word in any ACMED.L data or control statement is a label that identifies the purpose of the statement. Some statements contain only the label; e.g. INITIAL, END, STOP, ... etc. Others contain a label and appropriate data; e.g. to specify the admissible relative error in the integration process, the following statement must be used:

```
TIMER RELERR = .00005
```

The program checks the first four characters of a label, hence PARA, PARAM, ..., PARAMETER are equivalent. Note however that a label must be followed by at least one blank character.

2.2 The ACMED.L Statements

The ACMED.L statements can be classified as follows:

2.2.1 Data Statements

These are used to assign initial values (numeric or logical) to the constants, parameters, initial conditions and array entries that are associated with the model. A summary of their syntax is given below:

a) PARAM, CONST, and INCON

These statements are typically used to assign values to parameters, constants, and initial conditions. The general format of these statements is as follows:

PARAM *nam1* = *exp1*, *nam2* = *exp2*, ...

The format of CONST and INCON is similar to that of PARAM. Any number of variables may be initialized on a card and if necessary continuation cards may be used. Multiple PARAM, CONST, and INCON statements are permitted. The format of any of these statements is a set of assignment statements separated by commas, with the variables to be initialized on the left of the equal sign and an expression that consist of constants, variables or both on the right. However, the variable names on the right of an assignment statement must be previously initialized. The variable names and the constants must conform to their respective specification rules.

b) LIST

This statement allows the user to initialize one dimensional arrays in a convenient way. The format of this statement is as follows:

LIST *expressions*

where *expressions* are any combination of the following structures:

nam1 (*n*) = *exp1* or *nam2*(*n*-*m*) = *exp1*, K* *exp2*, ...

Note that *nam2*(*n*-*m*) is a reference to the elements *nam2*(*n*), *nam2*(*n*+1), ..., *nam2*(*m*) of the array *nam2*, and K* *exp2* means K repetitions of the value *exp2*.

Arrays initialized in the LIST statement must be dimensioned in a STORAGE or DIMENSION statement.

c) FUNCTION

This data specification construct permits the specification of the absissa-ordinate pairs of known data values. This data is subsequently used by the non-linear function generator NLFGEN (See Table 2.1). The format of the FUNCTION statement is as follows.

FUNCTION *name* = (x_1,y_1), (x_2,y_2), ...

where

name is the name given by the user to this collection of data values.

x_1,x_2, ... represent the values of the independent variable, and must be in an increasing order but the increment does not have to be of equal size.

y_1,y_2, ... represent the corresponding values of the dependent variable.

Each value of x and its corresponding y must be enclosed within parentheses. A comma is required between each pair of parentheses, and also between the two values enclosed within the parentheses. These pairs of values can be continued on consecutive cards. The maximum number of pairs per FUNCTION is twenty five. These limits are imposed by the array declarations in subroutine SPLINE which is called whenever the NLFGEN block is referenced.

d) FORTRAN Initialization Statements

The FORTRAN language provides a variety of mechanisms fof initializing variables and these can all be used in an ACMED.L program. This includes the DATA, INTEGER, REAL, and LOGICAL statements. (The DATA statement must proceed the first assignment statement).

2.2.2 Translation Control Statements

These statements provide information that is relevant to the translation phase of the program processing. In particular, they serve to define portions of the source code which need to be handled in some distinctive way.

2.2.2.1 Segments and Sections

An ACMED.L program is organized into segments; namely, INITIAL, DYNAMIC and TERMINAL. Each of these segments serves a particular function. The code within each segment may be further subdivided into SORT and NOSORT sections. In the absence of any explicit action by the user, the INITIAL and DYNAMIC segments are assumed to be SORT sections and the TERMINAL segment is assumed to be a NOSORT section. The beginning of these various subdivisions is indicated by the use of appropriate labels. A typical structure for an ACMED.L program is, therefore, as follows:

```
INITIAL
   ------
   ------
NOSORT
   ------
   ------
DYNAMIC
   ------
   ------
NOSORT
   ------
   ------
SORT
   ------
   ------
TERMINAL
   ------
   ------
END
```

The role of the labels is discussed below:

a) SORT

This label defines the beginning of a group of assignment statements which the user has written without giving any attention to the "correctness" of the sequence in which they appear. Correctness here refers to the situation where the variables on the right-hand side of each assignment statement have all been explicitly given a value in some preceding assignment statement or in one of the data declaration statements. Before such a group of statements can be considered to be in correct logical sequence, the statements must undergo a sorting operation. This is one of the steps in the translation phase.

In order to ensure feasibility of the sorting operation, there are restrictions on the type of statements which may appear in SORT sections. In particular, subprogram calls, the use of branching (DO loops, GO TO), the IF statement, and subscripted variables are not permitted.

Any particular SORT section is terminated by the beginning of a new segment or by the occurrence of the label, NOSORT. Note also that a SORT section is often referred to as "parallel" section.

b) NOSORT

No attempt is made during the translation process to alter the order of the statements in a NOSORT (or procedural) section. As a consequence, there is no restriction on the type of statements that may appear in such a section; i.e., all syntactically correct ACMED.L and FORTRAN statements are allowed.

Any particular NOSORT section is terminated by the beginning of a new segment or by the occurrence of the label, SORT.

c) INITIAL

This label identifies the beginning of the INITIAL segment of the program. The statements in this segment are executed before each run; i.e., at TIME equal to zero. This segment therefore provides a mechanism for initializing various parameters that appear in the model.

The INITIAL segment is optional and if it appears, then it is terminated by the label, DYNAMIC.

d) DYNAMIC

This label identifies the beginning of the code which defines the system model under investigation. Explicit use of this label is necessary only if the program includes an INITIAL segment. To be meaningful, an ACMED.L program must contain a DYNAMIC segment which includes at least one integration operation. The DYNAMIC segment is terminated either by the label TERMINAL (if the program contains a TERMINAL segment) or by the label, END.

e) TERMINAL

This label identifies the beginning of the TERMINAL segment of the ACMED.L program. The inclusion of such a segment is optional. The code in the TERMINAL segment is executed at the completion of each run; i.e., when a FINISH condition is satisfied or when FINTIM has been reached. The TERMINAL segment therefore provides a mechanism for carrying out post-run activities which could include outputing of data, changing model parameter values, or setting a flag to initiate another run (via CALL RERUN).

The TERMINAL segment is terminated by the first occurrence of the label, END.

2.2.2.2 Specification Fields

The code in an ACMED.L program up to the first END label is called the Main Specification Field, (this code will necessarily contain a DYNAMIC segment and possibly an INITIAL and a TERMINAL segment). Several additional END labels may appear and the code

enclosed between any pair of consecutive END labels is called an Auxiliary Specification Field. The sequence of Auxiliary Specification Fields is terminated by a STOP label which follows the last END label.

Each Auxiliary Specification Field causes at least one more run of the simulation model. The run(s) associated with an Auxiliary Specification Field are all carried out under the conditions specified by the data, control and output statements that appear in that particular Auxiliary Specification Field; i.e., PARAM, CONST, INCON, TIMER, FINISH, TABLE, PRTPLT. Note also that only data, control and output statements may appear in an Auxiliary Specification Field. All system and model parameters which are not explicitly assigned a value in an Auxiliary Specification Field, retain their previously assigned values. Note however, that an exception to this general "spirit" occurs in the case of the output statements TABLE and PRTPLT. Such output specifications are nullified upon entry into each new Auxiliary Specification Field and, if required, must be explicitly requested.

An example of the multiple END statement organization is shown below:

Main Specification Field

```
  .
  .
  .
PARAM ZETA = 4
  .
  .
  .
END
```

```
Auxiliary
Specification    PARAM ZETA = 5
Field
                 END

Auxiliary
Specification    PARAM ZETA = 6
Field
                 END

                 STOP
```

Apart form the possible interaction of other effects in the program, the above code would result in at least three separate runs, corresponding to the model parameter ZETA having the values 4, 5 and 6.

2.2.2.3 Declaration Statements of the FORTRAN Type

a) Type and Array Declarations

Except for the IMPLICIT statement, all other FORTRAN declaration statements such as LOGICAL, INTEGER, REAL, DIMENSION may be used in an ACMED.L program.

b) Labeled COMMON

The user is free to introduce labelled COMMON statements in his ACMED.L program. There are, however, restrictions on the variables which may appear in such a statement. Specifically, the following types of variable names must not be placed in a labelled COMMON:

- Those which are listed in a STORAGE, DIMENSION or DATA statement.
- Those which are assigned a value in a PARAM, CONST or INCON statement.
- Those which appear as the output of a function block: e.g. INTGRL, REALPL, CMPXPL, LEDLAG.

Note also that the use of an unlabelled COMMON is not permitted in an ACMED.L program.

c) EQUIVALENCE statement

The use of the FORTRAN EQUIVALENCE statement is permitted in an ACMED.L program. Its use is governed by the usual FORTRAN rules.

2.2.2.4 User Defined Functions

a) The PROCEDURE

The PROCEDURE construct provides a means for creating a collection of procedural code that is treated as a single functional block by the sorting process. The code within a PROCEDURE can include both ACMED.L structure statements and FORTRAN statements. This code is not sorted and the use of the SORT label within the body of the PROCEDURE is not allowed. A PROCEDURE is delineated by a header line which contains the name of the PROCEDURE and a terminator line which is simply the label, ENDPRO. The general format is shown below:

PROCEDURE Y_1, Y_2, --- YM = name (X_1,X_2, --- XN)

⋮

ENDPRO

where $Y_1, Y_2, \ldots$ YM represent the output variables of the PROCEDURE and $X_1, X_2, \ldots$ XN represent the input variables.

The designated output and input variables are utilized by the sorting algorithm in establishing the correct placement of the code contained in the PROCEDURE. Note also that the variable names appearing in the header line of a PROCEDURE are actual, rather than dummy, variable names. Procedures cannot be nested.

b) FORTRAN Subprograms

FORTRAN SUBROUTINE and FUNCTION subprograms may be used to augment the functional capabilities provided in ACMED.L. These should be prepared according to standard FORTRAN rules. Recall that subprogram references (calls) are permitted only from procedural (NOSORT) sections of the program. The code for these subprograms should appear after the STOP label which follows the last END label. This code must, in turn, be followed by an ENDJOB label which must always be present to define the end of the job.

2.3 Execution Control Statements

These are used to specify the values for various parameters that relate to the execution of the system model. Execution Control Statements are identified by two different labels. These statements cannot be continued on next card but multiple statements can be used if necessary.

a) TIMER

The general form of the TIMER statement is as follows:

TIMER *sys1* = *val1*, *sys2* = *val2*, ...

where *sys1*, *sys2* etc. are the names of various system parameters and *val1*, *val2* etc. are numeric constants (in real or integer format). At least one blank must follow the label TIMER. Once assigned a value via the TIMER statement, a parameter retains that value until a reassignment takes place via a subsequent TIMER statement.

A summary of the six parameters that can be specified via the TIMER statement is given below:

(i) FINTIM

This is the maximum value of the problem independent variable. If not explicitly specified, then FINTIM is given a default value of 10^{20}. Note that it is the user's responsibility to ensure that FINTIM is an even multiple of PDEL and TDEL (see below) in order that the output at the final integration step is properly captured.

(ii) ABSERR

Specifies the absolute error for the numerical integration process. If not explicitly given a value, then a default value of 10^{-4} is assigned.

(iii) RELERR

Specifies the relative error for the numerical integration process. If not explicitly given a value, then a default value of 10^{-4} is assigned.

(iv) PDEL

Specifies the increment for the print-plot output that is requested via a PRTPLT statement. If not explicitly given a value, then a default value of 5 is assigned.

(v) TDEL

Specifies the increment for the tabular output that is required via a TABLE statement. If not explicitly given a value, then a default value of 5 is given.

Note that if neither PDEL or TDEL is explicitly given a value by the user then their default values are changed to 10^{20}. In such cases the simulation run is usually terminated with no output (10^{20} is usually greater than the termination time), and accordingly it is essential to define one or the other of PDEL or TDEL.

(vi) DELMIN

Specifies the smallest allowable integration step size for the variable step-size integration algorithm used by GASP. If not explicitly given a value, then the assigned default value is 10^{-2} DELMAX where

$$\text{DELMAX} = \begin{cases} \text{TDEL; if absent, then PDEL;} \\ 10^{20} \quad \text{; if both TDEL and PDEL are absent} \end{cases}$$

b) FINISH

The FINISH statement provides a mechanism for causing the termination of a run according to conditions specified on the problem dependent variables. (Recall that the FINTIM parameter described above terminates a run on the basis of the problem independent variable). FINISH conditions are local to each specification field of the source program and hence have to be explicitly specified for each specification field where they are required.

The general form of the FINISH statement is as follows:

FINISH *exp1* α *exp2*, *exp3* α *exp4*, ...

where *exp1*, *exp2* etc. are FORTRAN expressions involving problem independent variables and constants; e.g. XDOT * (4 + YDOT) and α is either the character < (less than) or > (greater than). At least one blank must follow the keyword FINISH.

FINISH conditions are checked at half and full step size, and the algorithm, utilizing the variable step size, tries to get as close as possible to the exact point at which the two expressions are equal. However, the minimum step size controls the end of this process. In such cases if the termination time is not multiple of PDEL or TDEL (as appropriate), the output of the last integration step will be missed. Note also that when using the FINISH condition, data output at the termination time will occur only if the termination time is a multiple of PDEL or TDEL (as appropriate).

2.4 Output Control Statements

These are used to specify the variables that are to be displayed on the line printer (either in a tabular format or as a print-plot) using the problem independant variable (usually TIME) as the reference co-ordinate. Output control statements, like execution control statements, cannot be continued but multiple statements can be used if necessary. The total number of tables and print-plots in any particular run must not exceed ten and the number of variables specified in any particular print-plot or table statement must not exceed ten. If no output control statements are specified in an auxiliary specification field, then no output will be obtained for the run(s) associated with that particular auxiliary specification field.

All identifiers appearing in the model including constants, parameters, initial conditions, whether integer or real, subscripted or unsubscripted, can be displayed.

Output control statements are identified by the two following labels:

a) TABLE

This label is used to specify a list of variables whose values are to be tabulated at each TDEL increment of the problem independent variable. Each TABLE statement specifies a single table. The format of the TABLE statement is as follows:

TABLE *list*

where *list* is a sequence of up to 10 identifiers separated by commas; e.g.

TABLE A,B,X,Y,Z

b) PRTPLT

This label is used to specify a list of variables whose values are to be tabled and plotted at each PDEL increment of the problem independent variable. The format of this card is: the label PRTPLT followed by at least one blank then any combination of the following:

(i) A list of variables not enclosed within parantheses. A separate table and plot will be generated for each of these; e.g.

A,B,C, ...

(ii) A list of variables enclosed within parantheses. This will generate a single table and plot containing data for all the variables in the list; e.g.

(A,B)

For the purpose of print-plot output a variable can be scaled between limits by simply following it by two constants, separated by a comma, and enclosed within parenthesis. The two constants specify the lower and upper bounds for the corresponding print-plot; e.g.

A(10,25.5), X(.12,3.1)

If one of the bounds is not desired, then its position is left blank; e.g.

BT(10.5,) ,X12 (,.127)

Examples of various combinations of the above alternatives are given below:

PRTPLT A,XL, (A,B), D(,.92)

PRTPLT (X,Y(10,7.5), W(.5,)), Z

Denote by n_j the sum of the number of PRTPLT statements and TABLE statements in the jth specification field; then existing constraints impose the requirement that $n_j \leq 10$. Each PRTPLT and TABLE statement in the jth specification field is associated with a distinct file and these files must be properly defined by the user (See Appendix C). The total number of files required for the execution of an ACMED.L program is given by $\max\{n_1, n_2, \ldots n_m\}$ where m is the number of specification fields in the program.

2.5 Multiple Runs

A simulation experiment invariably involves multiple runs (executions) of the model where one or more model (or possibly integration) parameters change from run to run. Such situations fall into two broad categories; namely cases where the parameter set is known beforehand and cases where the parameter values are implicitly generated during the execution of the program; e.g. optimization studies and two point boundary value problems. It is therefore important to provide convenient mechanisms to accomodate both these situations.

ACMED.L provides two mechanisms in this respect. The first is based on a structural property of the source program and uses the idea of "Specification fields" (See section 2.2.2.2))

A second run control mechanism provided in the ACMED.L preprocessor is the CALL RERUN feature. Whenever the program flow in the TERMINAL segment

encounters this particular "call", another execution of the model occurs as a result of code set up by the ACMED.L preprocessor. This provides a particularly versatile and convenient method for controlling repetitive model execution. A typical usage of this feature is shown below:

```
      TERMINAL
         IF (ℓ) GO TO 50
            {code to alter the value(s) of one or more model parameters}
         CALL RERUN
      50 CONTINUE
```

If ℓ is FALSE, then the code for altering value(s) is executed together with the CALL RERUN statement. Execution of the statements between 50 CONTINUE and the first END statement in the source program follows and then control is passed to the INITIAL segment to initiate a further run (with the newly assigned parameter value(s)). Alternately if ℓ is TRUE, then control transfers directly to 50 CONTINUE and then the statements through to the first END are executed. Assuming that the statements following 50 CONTINUE do not contain a CALL RERUN, then no further runs will occur unless there remains an unprocessed auxiliary specification field.

Fig. 2.1 shows subprogram RERUN, which manipulates some of the GASP internal variables to force another execution of the model.

```
      SUBROUTINE  RERUN
      COMMON/UCONIO/RIRUN
      COMMON/MISC/IIECK,IIPOF,IIPOS,IISUM,IIPIR,JJFIL,IIFIN,NRTOT,JBEGG
      COMMON/GCOMS/IIEVT,IISED(6),JJBEG,JJCLR,MMNIT,MMON,NNAME(3),NNCFI,
     *NNDAY,NNPT,NNSET,NNPRJ,NNPRM,NNRNS,NNRUN,NNSTR,NNYR,SSEED(6)
      IIFIN    =1
      RIRUN    =1
      NNRNS    =NNRNS+1
      NNRUN    =NNRUN-1
      RETURN
      END
```

Fig. 2.1. Subprogram RERUN

Syntax	Interpretation
Dynamic Blocks	
Y=INTGRL(YZ,X)	$\dot{y}(t)=x(t)$ with $y(0)=y_0$
Y=MODINT(YZ,L1,L2,X)	$\dot{y}(t)=x(t)$ while L1=true $y(t)=y_0$ while L1=true and L2=false $y(t)=\bar{y}$ while L1=false and L2=false where $\bar{y}$ is the output value at the time when the L1=false and L2=false condition occurs
Y=REALPL(YZ,T,X)	$y(t)=\frac{1}{T}(x(t)-y(t))$ with $y(0)=y_0$ i.e. $\frac{y(s)}{x(s)}=\frac{1}{Ts+1}$
Y=LEDLAG(T1,T2,X)	$T_1\dot{y}(t)+y(t)=T_2\dot{x}(t)+x(t)$ with $T_1y(0)=T_2x(0)$ i.e. $\frac{y(s)}{x(s)}=\frac{T_1s+1}{T_2s+1}$
Y=CMPXPL(YZ,YDZ,A,B,X)	$\ddot{y}(t)+2AB\dot{y}(t)+B^2y(t)=x$ with $y(0)=y_0$ and $\dot{y}(0)=\dot{y}_0$ i.e. $\frac{y(s)}{x(s)}=\frac{1}{s^2+2ABs+B^2}$
Switch Blocks	
Y=FCNSW(C,X1,X2,X3)	$y(t)=\begin{cases}x_1(t) & \text{for } C<0\\ x_2(t) & \text{for } C=0\\ x_3(t) & \text{for } C>0\end{cases}$
Y=INSW(L,X1,X2)	$y(t)=\begin{cases}x_1(t) & \text{for L=false}\\ x_2(t) & \text{for L=true}\end{cases}$
Y1,Y2=OUTSW(L,X)	$y_1(t)=x(t)$, $y_2(t)=0$ for L=false $y_1(t)=0$, $y_2(t)=x(t)$ for L=true
L=COMPAR(X1,X2)	$L(t)=\begin{cases}\text{false for } x_1(t)<x_2(t)\\ \text{true for } x_1(t)\geq x_2(t)\end{cases}$

Syntax	Interpretation
Logic Blocks	
L=AND(L1,L2)	$L=\begin{cases}\text{true if both } L_1 \text{ and } L_2 \text{ are true}\\ \text{false otherwise}\end{cases}$
L=NAND(L1,L2)	$L=\begin{cases}\text{false if both } L_1 \text{ and } L_2 \text{ are true}\\ \text{true otherwise}\end{cases}$
L=IOR(L1,L2)	$L=\begin{cases}\text{true if } L_1 \text{ or } L_2 \text{ is true}\\ \text{false otherwise}\end{cases}$
L=NOR(L1,L2)	$L=\begin{cases}\text{false if } L_1 \text{ or } L_2 \text{ is true}\\ \text{true otherwise}\end{cases}$
L=EOR(L1,L2)	$L=\begin{cases}\text{true if } L_1 \text{ is true and } L_2 \text{ is false}\\ \text{true if } L_1 \text{ is false and } L_2 \text{ is true}\\ \text{false otherwise}\end{cases}$
L=NOT(L1)	$L=\begin{cases}\text{true if } L_1 \text{ is false}\\ \text{false otherwise}\end{cases}$
L=EQUIV(L1,L2)	$L=\begin{cases}\text{true if } L_1 \text{ and } L_2 \text{ are true}\\ \text{true if } L_1 \text{ and } L_2 \text{ are false}\\ \text{false otherwise}\end{cases}$
Function Generator Blocks	
Y=STEP(T)	$y(t)=\begin{cases}0 & \text{for } t<T\\ 1 & \text{for } t\geq T\end{cases}$
Y=RAMP(T)	$y(t)=\begin{cases}0 & \text{for } t<T\\ (t-T) & \text{for } t\geq T\end{cases}$
Y=LIMIT(A,B,X) (A<B)	$y(t)=\begin{cases}A & \text{for } x(t)<A\\ x(t) & \text{for } A\leq x(t)\leq B\\ B & \text{for } x(t)>B\end{cases}$
Y=NLFGEN(F,X)	$y(t)=f(x(t))$

Table 2.1 Functional Building Blocks

CHAPTER 3

The ACMED.L Preprocessor

3.1 The Preprocessor Organization

The overall task of the ACMED.L preprocessor can be divided into two subtasks; namely:

(i) Generation of the subprograms required by GASP to execute the simulation study specified by the source program.

(ii) Generation of the GASP data cards relating to run control, integration parameters and output.

From an operational point of view, the preprocessor carries out its function in three phases:

3.1.1 Phase I

The following functions are performed during this phase:

(i) The deletion of blank cards, comment cards (those with a * in column 1) and various header cards (SORT, NOSORT, PROCEDURE, ENDPRO, etc.).

(ii) The migration of state variable statements (those assignment statements having a dynamic function block on the right hand side) to the bottom of the DYNAMIC segment.

(iii) Sorting the statements in all appropriate sections.

(iv) The creation of the declaration statement:

COMMON/UCOM9/list of a-names, list of b-names

where:

1- An a-name is the variable name that appears on the left-hand-side of an assignment statement which contains a LEDLAG.

2- A *b*-name is a variable name which appears in a PRTPLT or TABLE statement and which is not a state variable and which is not assigned a value in the INITIAL segment.

(v) The creation of the type declaration statement:

LOGICAL list of *ℓ*-names

where an *ℓ*-name is the name of a variable which appears on the left-hand-side of an assignement statement that has a logic functional block on the right-hand-side (See Table 2.1).

3.1.2 Phase II

The processing in phase I results in the creation of a modified version of the original source program. This modified source program is passed to Phase II where the main translation activity takes place; specifically,

(i) The preprocessor parameters, tables and character strings are initialized.

(ii) Five statements required by the GASP processor are generated (See section 1.3 (iii)).

(iii) ACMED.L statements are translated to the equivalant GASP statements.

3.1.3 Phase III

The various code segments generated during phase II are merged into their respected frameworks to produce the FORTRAN subprograms required by GASP.

3.2 The File Structures

The preprocessor uses the following files:

a) The System file (F0)

The file F0 is a direct access file which uses the variable INF0 as a record pointer. It is maintained as an integral part of the preprocessor and contains the structural framework for each of the GASP subprograms that will be generated; e.g. headers, array declarations, COMMON declarations etc. In addition, this file contains the keyword tables, the character manipulation strings (See section 3.3 (iv)), and some organizational data relating to the temporary files.

b) The Temporary files (IN and F2)

(i) File IN is a sequential file which stores the ACMED.L program which is produced during phase I (See sections 3.1.1 , 3.4.1).

(ii) File F2 is a direct access file with record pointer INF2 used to temporarily store the statements of the generated GASP subprograms. The executable statements of each subprogram are stored starting at a record number predefined by data read from the System file F0 into array INDX(5). Similary, the declaration statements; e.g. (INTEGER, REAL, ...), EQUIVALENCE, and COMMON, are stored starting at locations INDEC, INEQV and INCOM respectivily. These eight pointers are updated as processing advances, and new statements are generated. Fig. 3.1 shows the organization of the temporary file F2.

c) The Main Output Files (F1 and F3)

(i) File F1 is a direct access file with record pointer INF1. It stores the data cards required by the GASP processor.

(ii) File F3 is a sequential file which stores the generated GASP source program. The creation of file F3 represents the main function of phase III.

Fig. 3.2 shows the phases and file structure of the ACMED.L preprocessor.

3.3 The Data Structures

A brief description of the key data structures used in the preprocessor is provided below:

(i) BUFFER (one dimensional, length = 80, type = INTEGER*2)
An input buffer.

(ii) CARD (one dimensional, length = 80, type = INTEGER*2)
An intermediate buffer for generated card images before they are stored on the files F1 and/or F2.

(iii) EQSTRN (one dimensional, length = 200, type = INTEGER*2).
This stores the generated EQUIVALENCE strings separated by commas. Each string consists of "("user variable","GASP state variable")". INSTRN points to the location following the last EQUIVALENCE string. In addition,

up to six small strings of characters are stored starting at location INSTRN. INSTR points to the first unoccupied location on EQSTRN. The six strings have GASP oriented data card information that is generated from the TIMER card. Each string is stored as follows:

1- A pointer to a location in EQSTRN that is initialized to zero, marking the absence of the string. Six such pointers exist i.e. INFINT, INPRTD, INPLTD, INDEL, INABSE, and INRELE.

2- The referenced location in EQSTRN contains the string length.

3- The string itself starts at the next location in EQSTRN.

Fig. 3.3 shows the EQSTRN array.

(iv) Character manipulation strings (type=INTEGER*2)

These strings are used in the translation process to aid in the generation of the required GASP code. Fig. 3.4 shows the various character manipulation strings and their respective lengths.

(v) Keyword Tables

Each keyword table is a two dimensional array with a set of keywords in the first column, and a set of associated indices in the second. The keyword tables are as follows:

1- KIWRD1 (two dimensional array, length=M1, type=INTEGER)
The keywords in this table are the labels with which the different ACMED.L statements are detected; i.e. column 1 of Table 3.1.

2- KIWRD2 (two dimensional array, length=M2, type=INTEGER
The keywords in this table are those of the functional building blocks; i.e. column 1 of Table 3.2.

3- KIWRD3 (two dimensional array, length=M3, type=INTEGER)
The keywords in this table are those which may appear on the TIMER statement.

(vi) FNSTOR (two dimensional array, length M, type=INTEGER)
This stores information about each FUNCTION, defined by the user, in the source program. Specifically the entries in the first column are FUNCTION names and the corresponding second column entries are the number of data points that have been provided.

(vii) INDX (one dimensional, length=5, type=INTEGER*2)
The entries in this vector store pointers to five distinct locations in the file F2. These locations are the "first available position" in each of the five sections of this file that are respectively allocated to the generated code for subprograms MAIN, INTLC, STATE, SSAVE, OTPUT.

3.4 The General Program Flow

As mentioned earlier, the preprocessor operates in three phases. The general program flow of each of these phases is described below.

3.4.1 Phase I

The processing associated with Phase I* takes place during the execution of the subprogram SORTER, which in turn calls the following subprograms:

AXSPEC, LEXY, LYNAME, MNSPEC, NAMID, SORT, STMTYP

The sorting procedure is organized to detect "unsortable" code; e.g. the pair of statements A = B, B = A, and to appropriately inform the user.

3.4.2 Phase II

The processing in this phase takes place on the "massaged" source program produced by Phase I.

The general program flow in the ACMED.L preprocessor is shown in Fig 3.5. This phase is carried out in two steps.

a) The Initiation Step

(i) The parameters, tables and character strings that are used during the processing are initialized as follows:

1 - The parameters are initialized in the preprocessor mainline. Table 3.3 lists these parameters together with their initial values and functions.

2 - The tables and character strings are read from the system file F0.

*The SORTER subroutine and its associated subprograms have been supplied by Mr. R. Haria and are based on his work described in [24]

(ii) Five statements that are required by the GASP main program and/or OTPUT subprogram are generated by invoking the following subprograms: STATM1, STATM2, STATM3, STATM4, STATM5 (See Table 3.4)

b) The Translation Step

In this step, the source program statements are sequentially read from file IN and processed. Fig. 3.6 shows the Pseudo-Code for this portion of the preprocessor program. The translation step is handled as follows:

(i) The preprocessor mainline (See Fig. 3.7)

1 - parses and interprets the first word (terminated by a blank)

2 - detects ACMED.L labeled statements

3 - branches to the appropriate processing subprograms

4 - processes "simple" statements; e.g. INITIAL, DYNAMIC, and TERMINAL

5 - branches to subprogram DYNMIC to handle the other statements

6 - on encountering the last statement (ENDJOB), it branches to the appropriate subprograms controlling the processing of phase III.

(ii) Subprogram DYNMIC (See Fig. 3.8)

1 - copies the FORTRAN labeled statements to file F2

2 - parses and interprets the first word after "="

3 - detects the ACMED.L function block statements

4 - branches to the appropriate processing subprograms

5 - processes the FORTRAN assignment statements

6 - in addition this subprogram sets a trace flag "on" if a TRACE statement was encountered in the source code.

3.4.3 Phase III

When the ENDJOB card is encountered in the ACMED.L source program, the preprocessor mainline transfers control to subprogram WRITER. This subprogram creates on file F3 the GASP programs from the various blocks on file F2 and the system file F0. The subprograms on file F3 and the data cards' images on file F1 represent an executable GASP module. The steps involved are as follows:

(i) Create a labeled COMMON statement having all constants, parameters, and initial conditions which are initialized in the INTLC subprogram block on file F2.

(ii) Create an EQUIVALENCE statement from the equivalence string array EQSTRN.

(iii) Merge the statements of the various blocks into their respective frameworks.

3.5 Error Detection and Messages

The ACMED.L preprocessor analyses each of the allowed structures and checks its syntax. Accordingly the source program statements will be checked twice; that is:

(i) During the translation phase the functional building blocks are checked, and as soon as the first error is detected the translation phase is abandoned and the job is terminated. In such a case the user is provided with the already translated statements terminated by the faulty one and an error message indicating the error number. If the error is not apparent the user should refer to the error table; i.e. Table 3.6. A trace feature is incorporated in ACMED.L to facilitate the location of (as yet) undetected "bugs" within the ACMED.L preprocessor (See Appendix D).

(ii) During the execution under GASP, the FORTRAN compiler performs the checks of FORTRAN syntax.
Note that a job is never transfered to the GASP processor if an illegal ACMED.L statement has been detected.

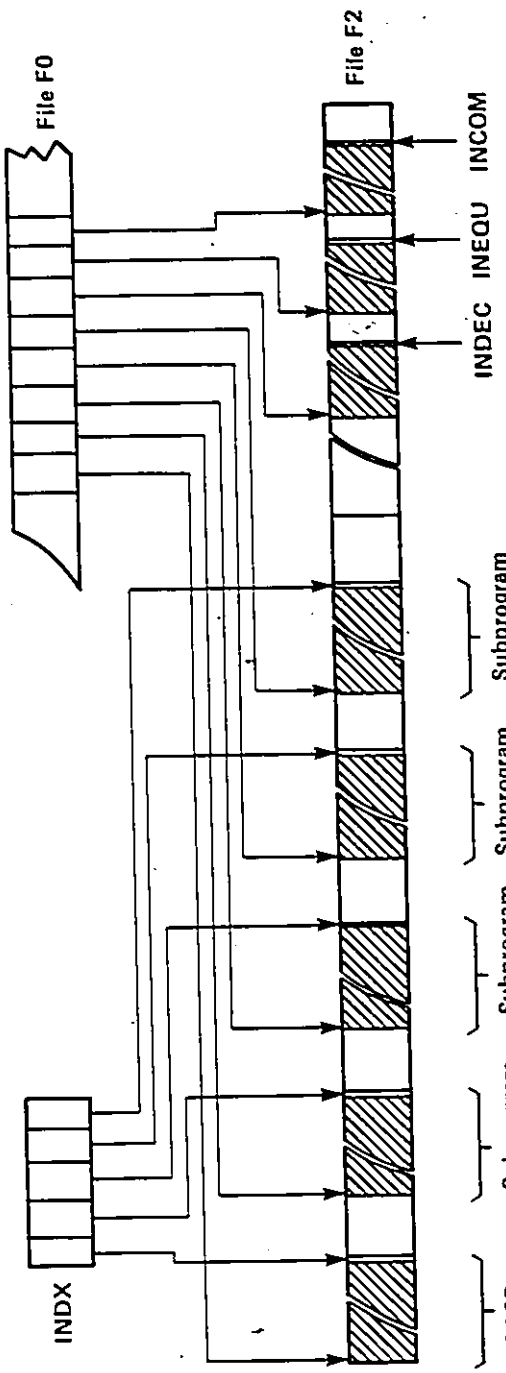
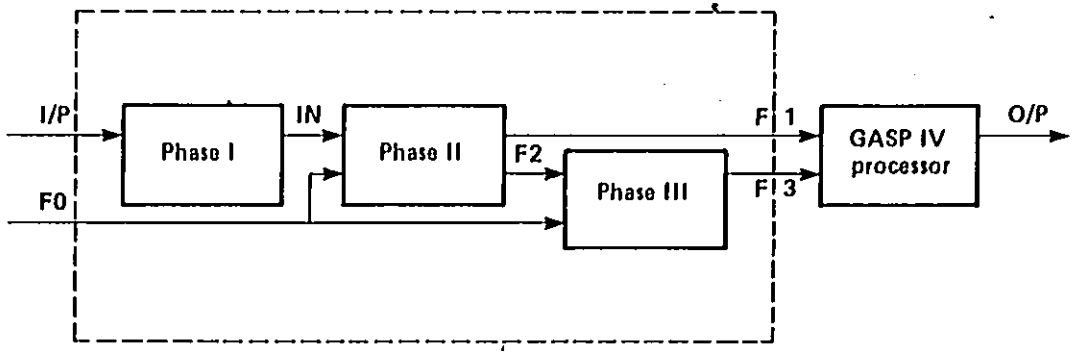


Fig. 3.1 The Organization of File F2



I/P ≡ Source Program From Card Reader or Terminal
F0 ≡ System File
IN ≡ Massaged Input File
F1 ≡ GASP Data File
F2 ≡ GASP Subprogram Statement Blocks File
F3 ≡ GASP FORTRAN Subprograms File
O/P ≡ Output to Line Printer or Terminal

Fig. 3.2 The Phases and File Structure of ACMED.L Preprocessor

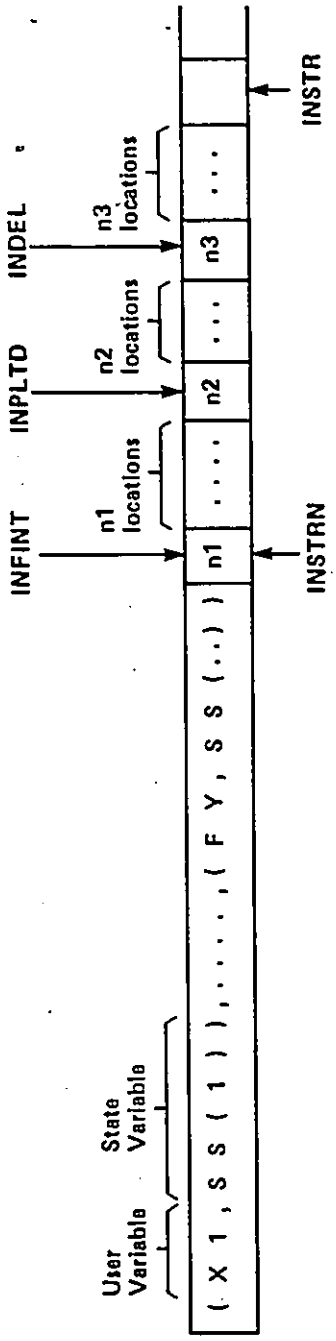


Fig. 3.3 The Format of the EQSTRN Array

STRING NAME	CONTENTS	LENGTH
ATOZ	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	26
TRUFLS	· T R U E · F A L S E ·	12
LOGI	· A N D · O R · N O T ·	12
DIGIT	0 1 2 3 4 5 6 7 8 9	10
CARCTR	( ) ' , * = :	7

Fig. 3.4 The Character Manipulation Strings

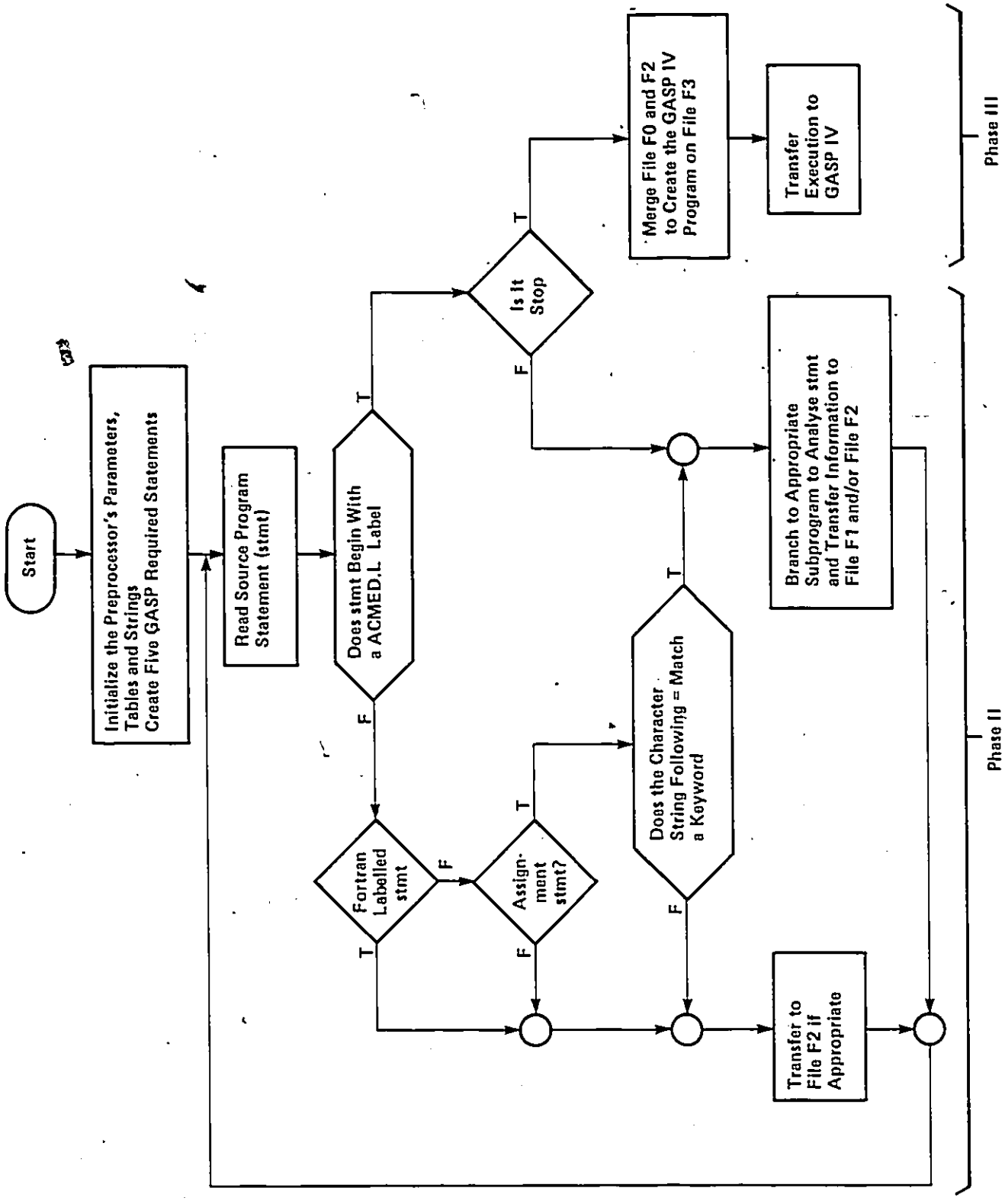


Fig. 3.5 Program Flow in the ACMED.L Preprocessor Phase II and III

```
WHILE not end of file DO
   read a statement
   IF statement starts with a label (Table 3.1)
      THEN
         IF lable is not STOP
            THEN
               - get index associated with label (entry to computed GO TO)
               - branch to appropriate subprogram (Table 3.1) to analyse statement.
               - transfer information to file F1 and/or F2
            ELSE
               - copy any user supplied FORTRAN subprograms to file F3
               - merge Files F0 and F2 to create the GASP program on file F3
               - utilize the JCL information to execute the GASP program.
               - STOP
         ENDIF
      ELSE
         IF statement is a FORTRAN labeled statement or not an assignment statement
            THEN
               Copy to file F2
            ELSE
               IF character string following "=" matches a keyword (Table 3.2)
                  THEN
                     - get index associated with keyword
                     - branch to appropriate subprogram (Table 3.2)
                     - transfer information to file F1 and/or F2
                  ELSE
                        Copy to file F2
               ENDIF
         ENDIF
   ENDIF
ENDWHILE
```

Fig. 3.6 A Simplified Pseudo-Code for the Translation and GASP Program Generation

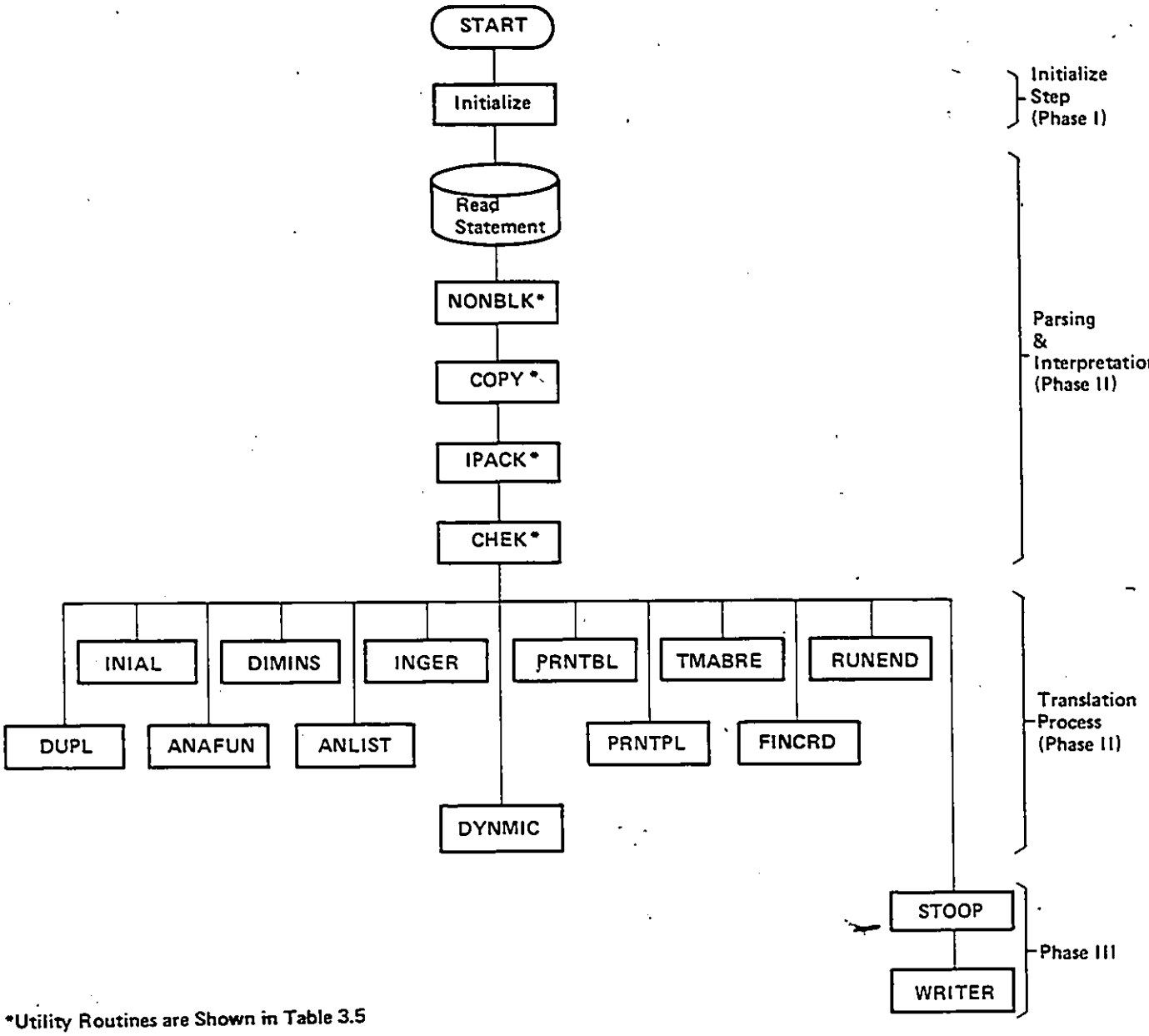


Fig. 3.7 The Program Structure for the ACMED.L Preprocessor Mainline

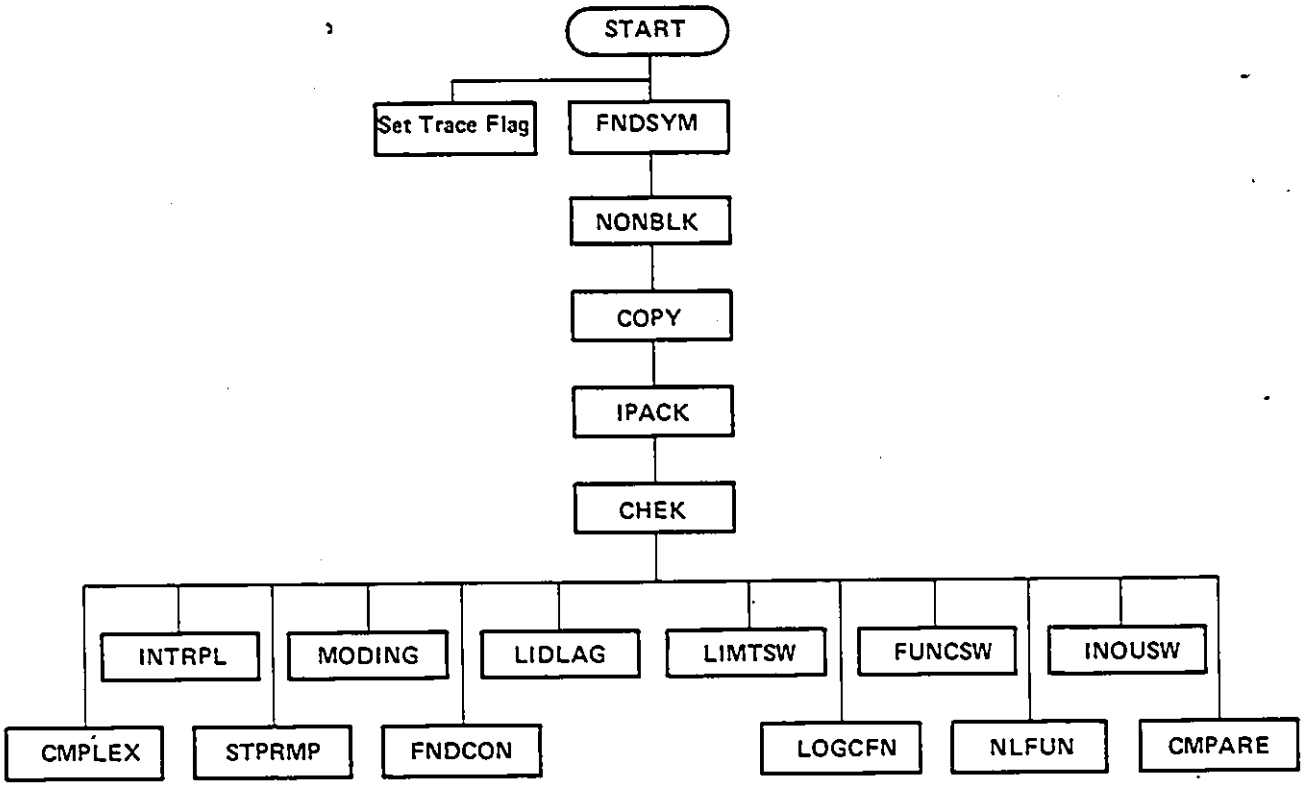


Fig. 3.8 The Program Structure for Subprogram DYNMIC

KIWRD1 table		Subprogram called for processing	Function of the Subprogram	Destination of Generated data		
Keyword/* LABEL	Index			file F1 (data statement type)	file F2 (data block	other files
INITIAL	1		flags the beginning of the INTLC subprogram data block			
PARAMETER	2	INITIAL	analyses PARAM statement		INTLC	
CONSTANT	2	INITIAL	analyses CONS statement		INTLC	
INCON	2	INITIAL	analyses INCON statement		INTLC	
FIXED	3	INGER	creates an INTEGER statement		declaration	
STORAGE	4	DIMINS	creates a DIMENSION statement		declaration	
DIMENSION	4	DIMINS	creates a DIMENSION statement		declaration	
INTEGER	5		copies the statement		declaration	
REAL	5		copies the statement		declaration	
LOGICAL	5		copies the statement		declaration	
COMMON	6		copies the statement		COMMON	
EQUIVALENCE	7		copies the statement		EQUIVALENCE	
FUNCTION	8	ANAFUN	analyses FUNCTION statement		INTLC	
LIST	9	ANLIST	analyses LIST statement		INTLC	
DYNAMIC	10		flags the beginning of the STATE subprogram data block			
TERMINAL	11		flags the beginning of the OTPUT subprogram data block			
TABLE	12	PRNTBL	analyses TABLE statement	PIO, VAR	SSAVE	
PRTPLT	13	PRNTPL	analyses PRTPLT statement	PIO, VAR	SSAVE	
TIMER	14	TMABRL	analyses TIMER statement and passes information to File F1	PIO, COM, INI		
FINISH	15	FINCRD	analyses FINISH statement		STATE	
END	16	RUNEND	detects the end of the run specification	SIM		
STOP	17	STOOP	- detects model termination. Copies any user supplied FORTRAN subprograms to file F3 - detects ENDJOB	FIN		F3
		WRITER	- merges files F0, F1 and F2 to generate GASP IV program on F3			F3

(*) Only the first 4 characters are relevant.

Table 3.1 KIWRD1 (labels) table and associated translation control subprograms.

KIWRD2 table		Subprogram called for processing	Function of the Subprogram
Keyword	Index		
INTGRL	1	INTRPL	analyses the INTGRL function
REALPL	1	INTRPL	analyses the REALPL function
MODINT	2	MODING	analyses the MODINT function
CMPXPL	3	CMPLEX	analyses the CMPXPL function
LEDLAG	4	LIDLAG	analyses the LEDLAG function
STEP	5	STPRMP	analyses the STEP function
RAMP	5	STPRMP	analyses the RAMP function
LIMIT	6	LIMTSW	analyses the LIMIT function
INSW	7	INOUSW	analyses the INSW function
OUTSW	7	INOUSW	analyses the OUTSW function
FCNSW	8	FUNCSW	analyses the FCNSW function
COMPAR	9	CMPARE	analyses the COMPAR function
NAND	10	ANONOT	analyses the NAND function
NOR	11	ANONOT	analyses the NOR function
NOT	12	ANONOT	analyses the NOT function
AND	13	ANONOT	analyses the AND function
IOR	14	ANONOT	analyses the IOR function
EOR	15	EOREQV	analyses the EOR function
EQUIV	16	EOREQV	analyses the EQUIV function
NLFGEN	17	NLFUN	analyses the NLFGEN function

Notes: 1 - The calls to ANONOT and EOREQV take place indirectly via subprogram LOGCFN

2 - The generated data is destined to the appropriate data blocks on file F2

Table 3.2 KIWRD2 table and associated translation control subprograms.

Parameter	Initial value	Role of the Parameter
BLKFLG	2	Set to 1 when the INITIAL section is being processed Set to 2 when the DYNAMIC section is being processed Set to 3 when the TERMINAL section is being processed
TRFLAG	.FALSE.	Set to .FALSE. when the trace option has not been requested by the user. Set to .TRUE. when the trace option has been requested by the user.
DTFLAG	.FALSE.	Set to .FALSE. if no data card images have yet been generated. Set to .TRUE. when one or more data cards' image have been generated.
INF0	1	File F0 record pointer
INF1	1	File F1 record pointer
INF2	1	File F2 record pointer
INDEC	1001	File F2 record pointer to the beginning of the declaration block (See section 3.2 b) 2))
INEQV	1026	File F2 record pointer to the beginning of the EQUIVALENCE block (See section 3.2 b) 2))
INCOM	1051	File F2 record pointer to the beginning of the COMMON block (See Section 3.2 b) 2))
INDAT2	2	File F3 record pointer to the STA data card
INSTRN	1	Pointer to the location next to the last EQUIVALENCE string
INSTR	1	Pointer to the first unused location in EQSTRN.

Table 3.3 The ACMED.L preprocessor parameters and their initial values.

Parameter	Initial value	Role of the Parameter
INFNST	1	Pointer to the first unused entry in the FNSTOR table
INFINT	0	A zero value indicates that a user specified value for FINTIM has not yet been located. A non-zero value is a pointer to EQSTRN where the user specified value for FINTIM is stored
INPRTD	0	Analogous to INFINT but relevant to TDEL rather than FINTIM
INPLTD	0	Analogous to INFINT but relevant to PDEL rather than FINTIM
INDEL	0	Analogous to INFINT but relevant to DELMIN rather than FINTIM
INABSE	0	Analogous to INFINT but relevant to ABSERR rather than FINTIM
INRELE	0	Analogous to INFINT but relevant to RELERR rather than FINTIM
NOMCOM	10	Stores the next available number to be used as a suffix in the sequence of system generated labelled common statements; e.g. /UCOM10/, /UCOM11/ etc.
NOASK	100	Stores the next available number to be used as a suffix in the sequence of system generated variable names; e.g. ASK100, ASK101, etc.
INTLAB	9111	Stores the next available statement number for labelled statements required in the INTLC block of file F2
SAVLAB	9111	Analogous to INTLAB but relevant to SSAVE block of file F2
NOPLOT	1	Stores the value $(N_{tp}+1)$ where N_{tp} is the number of generated TABLES and/or PRTPLT

Table 3.3 Cont.

Parameter	Initial value	Role of the Parameter
NOPFIL	11	File number associated with TABLE and/or PRTPLT.
STVRNO	1	A counter whose value is (N_U+1) where N_U is the number of state variable statements generated "to-date"
RUNNUM	1	The number of the run currently being set up

Table 3.3 Cont.

Subprogram	Function	Generated data destination	
		File F1 (Data Card Type)	File F2 Subprograms
ANORNOT	Processes AND, NAND, OR, NOR and Not Logical functions.		IIC*
ANTBL1	Processes the LIST name(n)=exp statement.		
ANTBL2	Processes the LIST name(n-m)=exp1, K*exp2, ... statement.		
ARHLAR	Processes the right hand side of the second type of LIST statement		
BLDCOM	Creates a labelled COMMON statement using the COMMON block string.		COMMON
BLDEQV	Creates an EQUIVALENCE statement from the equivalence array EQSTRN.		EQUIVALENCE
DAT1	Creates the GEN card image.	GEN	
DAT2	Creates the STA card image.	STA	
DAT3	Creates the LIM card image.	LIM	
DAT7	Creates the PLO card image.	PLO	
DAT8	Creates the VAR card image.	VAR	
DAT10	Creates the CON card image.	CON	
DAT12	Creates the INI card image.	INI	
EOREQV	Processes both EOR and EQUI logical function.		IIC
PLOTST	Creates a PLOT statement.		SSAVE
PTMMMX	Puts the minimum and maximum of the plots and tables on CON card.	CON	
STATM1	Creates the statement RIRUN=0		MAIN,OTPUT
STATM2	Creates the statement MMQ(1)=0		MAIN
STATM3	Creates the statement NCRDR=9		MAIN
STATM4	Creates the statement NPRNT=6		MAIN
STATM5	Creates the statement IIFIN=0		OTPUT
WRTCGP	Creates the statement CALL GPLOT(PLOT,THOM,N) where N=NOPLOT-1		SSAVE
DTCHEK	Checks for the existance of any data card image on file F1		
FNINTB	Puts a function symbolic name and its length in FNSTOR table		

Table 3.4 Subsidiary subprograms

IIC* In-Line-Code. During Phase II, certain ACHED.L source statements are replaced, "in-line", by one or more lines of equivalent FORTRAN code; that is, if a source statement falls in the INITIAL section, then the generated data destination is the INTIC subprogram. If it falls in the DYNAMIC or TERMINAL sections then the generated data destination is subprogram STATE.

Subprogram name	Function
ALFCHK	Checks whether a character is alphabetic.
ALTONO	Converts an alphanumeric to an equivalent number.
BLANK	Blanks a part of a buffer.
BLDIF	Forms the structure of an IF statement.
CHEK	Checks whether a word exists in a keyword list.
CHKCON	Checks whether a statement has a continuation.
COPY	Copies a part of a buffer into another buffer.
DUPL	Writes an input statement into a file.
FLNACP	Finds the length of a string of characters and copies it to a buffer.
FNDCON	Finds the continuation of a statement.
FNDSYM	Finds a character in a string.
IPACK	Packs four or less characters into a single word.
GETIN	Reads a statement.
GETOUT	Writes a statement.
ERROR	Writes an error message.
NONBLK	Gets the first/or last non-blank character in a string.
NOTOBF	Converts a number to a character string, and copies it into a buffer.
PREPAR	Gets the first token in a buffer which must be preceeded by "(".
PTIFGO	Creates a statement "IF(NNRUN.NE.*run-number*)GOTO lable" and copies it into a file.
PTIFRR	Creates the statements "IF(RIRUN.NE.*rerun-flag*) GOTO *label*" and "IF(NNRUN.NE *run-number*] GOTO *label1* " and copies them to a file.
PTCONT	Creates the statement "*label* CONTINUE" and copies it into a file.
RMVEND	Removes the last characters of a record.
SPLITER	Splits a sequence of assignment statements separated by commas to separate ones which are then copied sequentially to a file.
SVINEQ	Saves an equivalence string; e.g. (xxx, yyy) in the array EQSTRN.
TESTIN	Reads the first non-comment card and gets its first non blank character.
WRDEQL	Creates the structure "XYZ$n_1n_2n_3$ =" where X,Y,Z are characters and n_1,n_2,n_3 are numbers. The blank before the "=" sign would be ")" if Z is "(".

Table 3.5 Utility subprograms

Error #	Invalid ACMED.L statement	Where error message is generated
150	FUNCTION	ANAFUN
175	LIST	ANATAB,ANTBL1,ANTBL2,ARMLAR
200	INTGRL	INTRPL
225	REALPL	INTRPL
250	CMPXPL	CMPLEX
275	LEDLAG	LIDLAG
300	MODINT	MODING
325	FCNSW	FUNCSW
350	INSW	INOUSW
375	OUTSW	INOUSW
400	LIMIT	LIMTSW
425	STEP	STPRMP
450	RAMP	STPRMP
475	NLFGEN	NLFUN
500	AND or NAND or IOR or NOR or NOT	ANONOT
525	EOR or EQUIV	EOREQV
550	COMPAR	CMPARE
575	FINISH	FINCRD
600	TIMER	TMABRL
625	TABLE	PRNTBL
650	PRTPLT	PRNTPL,PTMNMX
675	STOP	MAIN: Input error or no STOP statement
700		DYNMIC: missing right hand side of an assignment statement
725		CHKCON: invalid continuation statement
750		LOGFUN: missing right hand side of a logical function block.
	ENDJOB	If ENDJOB statement is missing, an abnormal termination occurs. EOF error.

Table 3.6 ACMED.L Error Codes

CHAPTER 4

The Translation Process

4.1 Introduction

The generated output for each source statement is a set of character strings which are placed in appropriate areas in the files F1, F2 and/or F3. These character strings can be divided into two categories, namely (a) statements (in the FORTRAN sense) and (b) expressions. Each statement "belongs" to one of the FORTRAN subprograms that is being synthesized (i.e. MAIN, INTLC, STATE, SSAVE or OTPUT). Each expression, on the other hand, "belongs" to either a labeled COMMON statement, or an EQUIVALENCE statement or to the input data file, F1. Note also that each labeled COMMON or EQUIVALENCE statement that is created by the translator is inserted into every synthesized subprogram. The steps in the translation process for each source statement are as follows:

(i) Create a set of pointers to mark the tokens of the source statement. According to the type of the statement, a token can be a single word, an expression or an assignment statement. Tokens are usually separated by one of the following characters:

, ( ) = - blank or three consecutive dots.

(ii) Sequentially build the set of GASP statements equivalent to the ACMED.L source statement. Update various counters and tables as necessary.

(iii) A created character string is stored into its appropriate block on files F1, F2, F3 and/or in the equivalence string array, EQSTRN.

4.2 The Inputs and Outputs

This section provides a detailed account of the translation process as it relates to the various ACMED.L source statement. This information is summarized in Tables 4.1 - 4.10 which show the code generated for each ACMED.L statement. These tables use the following terminology:

name, *nam1*, *nam2*,... ≡ symbolic names

exp, *exp1*, *exp2*, ... ≡ constants or arithmatic expressions (in the FORTRAN sense)

log, *log1*, *log2*, ... ≡ logical expressions (in the FORTRAN sense)

runj ≡ current run flag { 0 predefined run; 1 conditional run }

runn ≡ current run number (a positive integer greater than 0)

ASK*j* ≡ a generated variable name (*j* ≡ positive integer greater than 100)

UCOM*i* ≡ a generated labeled COMMON *name* (*i* ≡ positive integer greater than 9)

label	≡ a generated statement label (a positive integer greater than 9110)
SS(*n*)	≡ standard GASP designation for state variables
DD(*n*)	≡ standard GASP designation for the time derivative of SS(*n*) (*n* correspond to the preprocessor internal variable called STVRNO)
ILC	≡ See footnote in Table 3.4
Run Data Cards	≡ data cards that relate to a specific run
Global Data Cards	≡ data cards that relate to the overall simulation program; e.g. GEN and FIN

Note the following:

(i) The blocks dealt with on file F2 correspond to the GASP MAIN, the INTLC, STATE, OTPUT, SSAVE subprograms, and the declaration, common and equivalence blocks. File F1 holds the data card images. The final executable GASP program is generated in file F3.

(ii) The three labels PARAM, INCON and CONS are treated in an identical manner. Table 4.1 considers the PARAM case only.

(iii) The syntax of the PRTPLT statement permits a variety of different forms. A representative form which serves to illustrate the salient feature of the translation procedure is shown in Table 4.7(b).

(iv) The three labels INIT, DYNA and TERM do not give rise to any new code in the target FORTRAN subprograms. Rather they influence where the generated code is placed. Specifically, after the label INIT is encountered, all subsequently generated statements are placed into the INTLC subprogram. The labels DYNA and TERM have an analogous effect with respect to the STATE and OTPUT subprograms respectively. If the first assignment statement is encountered before an occurence of either the INIT or DYNA labels, then it is assumed that these labels are absent and that the current source code "belongs" to the DYNAMIC segment of the source program.

(v) The labels SORT/NOSORT and PROC/ENDPRO are removed from the source code at the completion of Phase I. Hence these labels never appear in the modified source program seen by the translation phase.

(vi) On encountering an END card, the code shown in Table 4.9 is generated and in addition the following preprocessor internal variables are updated:

1 - The run number (RUUNUM) is incremented by 1.

2 - The number of tables and print-plots for the new run is initialized to 1.

(vii) After generating the code associated with the STOP card (See Table 4.9), the preprocessor checks for the existence of user-written subprograms. If present, these subprograms are copied to file F3. If the ENDJOB card is not present after the last user-written subprogram, then an abnormal termination occurs.

(viii) The detection of the ENDJOB statement, causes the generation of the code shown in Table 4.9. In addition, the processing of Phase III is initiated.

The final Phase III is concerned with the creation on file F3 of the executable GASP program from code fragments located in various files. This GASP program consists of (i) a main program, (ii) the five subprograms INTLC, STATE, OTPUT, SSAVE and RERUN, (iii) the two subprograms SPLINE and TRIDIA (Fig. 4.1), which handle the interpolation process associated with the NLFGEN block, (iv) user supplied subprograms and (v) the GASP data cards. Items (i) and (ii), are created by fusing together various data blocks that have been generated on files F0 and F2, and the various constituent elements involved here are summarized in Table 4.10. The definitions of Blocks A, B, and C are given below:

Block A:

```
      INTEGER  JEVNT,MFA,MFE,MLE,MSTOP,NCRDR,NNAPO,NNAPT,NNATR,NNFIL,NNG,
     *NNTRY,NPRNT,ISEES,LFLAG,NFLAG ,NNEOD,NNEGS,NNEGT,IIEVT,IISED,JJBEG
     *,JJCLR,MMNIT,MMON,NNAME,NNCFI,NNDAY,NNPT,NNSET,NNPRJ,NNPRM,NNRNS,
     *NNRUN,NNSTR,NNYR
      INTEGER  IIECK,IIPOF,IIPOS,IISUM,IIPIR,JJFIL,IIFIN,NRTOT,JEEGG
```

Block B:

```
      COMMON /GCOM1/ ATRIB(25),JEVNT,MFA,MFE(100),MLE(100),MSTOP,NCRDR,N
     *NAPO,NNAPT,NNATR,NNFIL,NNG(100),NNTRY,NPRNT,PPARM(50,4),TNOW,TTBEG
     *,TTCLR,TTFIN,TTRIE(25),TTSET
      COMMON /GCOM2/ DD(100),DDL(100),DTFUL,DTNOW,ISEES,LFLAG(50),NFLAG,
     *NNEOD,NNEQS,NNEGT,SS(100),SSL(100),TTNEX
      COMMON/GCOM5/IIEVT,IISED(6),JJBEG,JJCLR,MMNIT,MMON,NNAME(3),NNCFI,
     *NNDAY,NNPT,NNSET,NNPRJ,NNPRM,NNRNS,NNRUN,NNSTR,NNYR,SSEED(6)
      COMMON/MISC/IIECK,IIPOF,IIPOS,IISUM,IIPIR,JJFIL,IIFIN,NRTOT,JEEGG
```

Block C:

```
      COMMON/MISC/IIECK,IIPOF,IIPOS,IISUM,IIPIR,JJFIL,IIFIN,NRTOT,JEEGG
      COMMON/GCOM5/IIEVT,IISED(6),JJBEG,JJCLR,MMNIT,MMON,NNAME(3),NNCFI,
     *NNDAY,NNPT,NNSET,NNPRJ,NNPRM,NNRNS,NNRUN,NNSTR,NNYR,SSEED(6)
      IIFIN   =1
      RIRUN   =1
      NNRNS   =NNRNS+1
      NNRUN   =NNRUN-1
```

Table 4.11 provides an indication of the data which appears on the synthesized GASP data cards and the origin of this data in the source ACMED.L program.

```
      FUNCTION      SPLINE(N,X,ALF)
      DIMENSION     X(N,2),C(4,25),DF(25),DX(25),
     *              SUB(24),DIA(24),SUP(24),B(24)
      NPI       =N+1
      NMI       =N-1
      DO 2 K=1,NMI
      DX(K)     =X(K+1,1)-X(K,1)
      DF(K)     =(X(K+1,2)-X(K,2))/DX(K)
2     CONTINUE
      DIA(1)    =2.
      SUP(1)    =1.
      B(1)      =3.*DF(1)
      SUB(N)    =1.
      DIA(N)    =2.
      B(N)      =3.*DF(NMI)
      DO 4 I=2,NMI
      SUB(I)    =DX(I)
      DIA(I)    =2.*(X(I+1,1)-X(I-1,1))
      SUP(I)    =DX(I-1)
      B(I)      =3.*(DF(I-1)*DX(I)+DF(I)*DX(I-1))
4     CONTINUE
      CALL      TRIDIA(N,SUB,DIA,SUP,B)
      DO 6 I=2,N
      C(1,I)    =X(I-1,2)
      C(2,I)    =B(I-1)
      C(3,I)    =( 3.*DF(I-1)-2.*B(I-1)-B(I))/ DX(I-1)
      C(4,I)    =(-2.*DF(I-1)+   B(I-1)+B(I))/(DX(I-1)**2)
6     CONTINUE
      C(1,1)    =X(1,2)
      C(2,1)    =B(1)
      C(3,1)    =0.
      C(4,1)    =0.
      C(1,NPI)=X(N,2)
      C(2,NPI)=B(N)
      C(3,NPI)=0.
      C(4,NPI)=0.
      K         =NPI
      DO 8 I=1,N
      AX        =ALF-X(K-1,1)
      IF(AX.GE.0.)                    GO TO 12
      K         =K-1
8     CONTINUE
12    OUTPUT    =((C(4,K)*AX+C(3,K))*AX+C(2,K))*AX+C(1,K)
      SPLINE    =OUTPUT
      RETURN
      END



      SUBROUTINE    TRIDIA(N,SUB,DIA,SUP,B)
      DIMENSION     SUB(N),DIA(N),SUP(N),B(N)
      DO 10 K=2,N
      DIA(K)  =DIA(K)-SUB(K)*SUP(K-1)/DIA(K-1)
      B  (K)  =B  (K)-SUB(K)*B  (K-1)/DIA(K-1)
10    CONTINUE
      B  (N)  =B(N)/DIA(N)
      N1        =N-1
      DO 20 M=1,N1
      K         =N-M
      B  (K)  =(B  (K)-SUP(K)*B  (K+1))/DIA(K)
20    CONTINUE
      RETURN
      END
```

Fig. 4.1 Interpolation Program Using a Cubic Natural Spline Function [25]

Source Input	Generated Code	Generated Code Destination
PARAM *nam1=exp1, nam2=exp2,...*	COMMON/UCOM*i*/*nam1,nam2,...* IF (RIRUN.NE.*nunf*) GO TO *label* IF (NNRUN.NE.*nunn*) GO TO *label* *nam1=exp1* *nam2=exp2* : *label*: CONTINUE	All subprograms INTLC
LIST *nam1(n1)=exp1,nam2(n2)=exp2,...* *nam3(n3-n4)=exp3,exp4,...,n*expm* where: *n1,n2,...,n* are INTEGER values	*nam1(n1)=exp1* *nam2(n2)=exp2* *nam3(n3)=exp3* *nam3(n3+1)=exp4* : *nam3(l)=expm* : *nam (l+n-2)=expm* *nam (n4)=expm* where: *nam1,nam2,...* are dimensional arrays (vectors)	INTLC
FUNCTION *name=(n1,m1),(n2,m2),...(nr,mr)* where: the data values (*nj,mj*) may be either INTEGER or REAL values	COMMON/UCOM*i*/*name(n,2)* *name(1,1)=n1* *name(1,2)=m1* *name(2,1)=n2* *name(2,2)=m2* : *name(n,1)=nr* *name(n,2)=mr*	All subprograms INTLC

Table 4.1 Translation of the Data Statements

Source Input	Generated Code	Generated Code Destination
name=INTGRL(*exp1*,*exp2*)	COMMON/UCOM*i*/*name* EQUIVALENCE(SS(*n*),*name*) *name*=*exp1* DD(*n*)=*exp2*	All subprograms INTLC STATE
name=REALPL(*exp1*,*exp2*,*exp3*)	COMMON/UCOM*i*/*name* EQUIVALENCE(SS(*n*),*name*) *name*=*exp1* DD(*n*)=1./*exp2*(*exp3*-*name*)	All subprograms INTLC STATE
name=MODINT(*exp1*,*log1*,*log2*,*exp2*)	EQUIVALENCE(SS(*n*),*name*) DD(*n*)=0 IF(.NOT.*log1*.AND.*log2*) SS(*n*)=*exp1* IF(*log1*) DD(*n*)=*exp2*	All subprograms STATE
name=CMPXPL(*exp1*,*exp2*,*exp3*,*exp4*,*exp5*)	COMMON/UCOM*i*/*name*,ASK*j* EQUIVALENCE(SS(*n*),*name*), (SS(*n*+1),ASK*j*) *name*=*exp1* ASK*j*=*exp2* DD(*n*)=ASK*j* DD(*n*+1)=-2.*exp3*exp1*ASK*j*-*exp1***2*name+exp5	All subprograms INTLC STATE
name=LEDLAG(*exp1*,*exp2*,*exp3*)	SS(*n*)=0 ASK*j*=*exp3* DD(*n*)=(ASK*j*-SS(*n*))/*exp2* *name*=(ASK*j*-SS(*n*))*exp1/exp2+SS(*n*)	INTLC STATE

Table 4.2 Translation of the Dynamic Functional Blocks

Source Input	Generated Code	Generated Code Destination
name=FCNSW(*exp1*,*exp2*,*exp3*,*exp4*)	*name*=*exp2* IF(*exp1*.EQ.0) *name*=*exp2* IF(*exp1*.GT.0) *name*=*exp3*	
name=INSW(*log*,*exp1*,*exp2*)	*name*=*exp1* IF(*log*)*name*=*exp2*	
nam1,*nam2*=OUTSW(*log*,*exp*)	*nam1*=*exp* *nam2*=0 IF(*log*) *nam1*=0 IF(*log*) *nam2*=*exp*	I/C
log=COMPAR(*exp1*,*exp2*)	*log*=.TRUE. IF(*exp1*.GT.*exp2*) *log*=.FALSE.	

Table 4.3 Translation of the Switching Blocks

Source Input	Generated Code	Generated Code Destination
ℓog=AND(ℓog1,ℓog2)	ℓog=ℓog1.AND.ℓog2	ILC
ℓog=NAND(ℓog1,ℓog2)	ℓog= .NOT.(ℓog1.AND.ℓog2)	
ℓog=IOR(ℓog1,ℓog2)	ℓog=ℓog1.OR.ℓog2	
ℓog=NOR(ℓog1,ℓog2)	ℓog=.NOT.(ℓog1.OR.ℓog2)	
ℓog=NOT(ℓog1)	ℓog= .NOT.(ℓog1)	
ℓog=EQUIV(ℓog1,ℓog2)	ℓog=(ℓog1.AND.ℓog2).OR.(.NOT.ℓog1.AND..NOT.ℓog2)	
ℓog=EOR(ℓog1,ℓog2)	ℓog=.NOT.((ℓog1.AND.ℓog2).OR.(.NOT.ℓog1.AND..NOT. ℓog2))	

Table 4.4 Translation of the Logic Blocks

Source Input	Generated Code	Generated Code Destination
name=LIMIT(exp1,exp2,exp3)	*name*=exp3 IF(exp3.LT.exp1) *name*=exp1 IF(exp3.GT.exp2) *name*=exp2	ILC
name=STEP(exp)	*name*= 1 IF(TNOW.LT.exp) *name*=0	
name=RAMP(exp)	*name*=TNOW-exp IF(TNOW.LT.exp) *name*=0	
nam1=NLFGEN(*nam2*,*n*)	*nam1*=SPLINE(*m*,*nam2*,*n*) where: *nam2* is FUNCTION symbolic name *n* is INTEGER/REAL constant *m* is INTEGER, number of entries in the FUNCTION statement associated with *nam2*	

Table 4.5 Translation of the Function Generator Blocks

Source Input	Data Value and Destination		
	Data	Card Type	Field number
TIMER FINT=n1,DELM=n2,PDEL=n3,TDEL=n4,ABSE=n5,RELE=n6	n3 or n4*	PLO	7
	n5	CON	4
	n6	CON	5
	n2	CON	7
	n3 or n4	CON	8
	n1	INI	6

Table 4.6(a) Translation of the TIMER Execution Control Statement

Source Input	Generated Code	Generated Code Destination
FINISH exp1 α1 exp2, exp3 α2 exp4,...	IF(NNRUN.NE.*nnnn*) GO TO *label* IF(exp1.*rel1*.exp2) MSTOP=-1 IF(exp3.*rel2*.exp4) MSTOP=-1 : IF(MSTOP.EQ.-1) ISEES=-1 *label*: CONTINUE	STATE
where: α1, α2, ... are one of the relational operators; i.e. < or >	where: *rel1*,*rel2* are one of the FORTRAN relational operators; i.e. LT or GT note: *rel1* and *rel2* correspond to α1 and α2 respectively	

Table 4.6(b) Translation of the FINISH Execution Control Statement

(*)See section 2.3.1

Source Input	Generated Code	Generated Code Destination
TABLE *nam1,nam2,... nam m ...*	IF(NNRUN.NE.*nnnn*) GO TO *label* PLOT(1)=*nam1* PLOT(2)=*nam2* ⋮ PLOT(m)=*namm* CALL GPLOT(PLOT,TNOW,*i*) *label*: CONTINUE PLO,*i*,TIME,*j*,*m*,1,*val** VAR,*i*,1,,*nam1** VAR,*i*,2,,*nam2** ⋮ VAR,*i*,*m*,,*namm* where: *m* is INTEGER*10; number of variables to be plotted simultaneously *i* is INTEGER; order of a specific table between other tables and print-plots *j* is integer; plotting file number (1 sequentially up to 20) *val* is PDEL or TDEL from TIMER card	SSAVE Run Data Cards

Table 4.7(a) Translation of the TABLE Output Control Statement

Source Input	Generated Code	Generated Code Destination
PRTPLT *nam1*, *nam2* (*min1*, *max1*), (*nam3*, *nam4* (*min2*,), ...) where: *min1*, *min2*, ... are REAL constants; value of the plot minimum coordinate *max1*, ... are REAL constants; value of the plot maximum coordiante	IF(NNRUN.NE.*nunn*) GO TO *label* PLOT(1)=*nam1* CALL GPLOT(PLOT,TNOW,*i*) PLOT(1)=*nam2* CALL GPLOT(PLOT,TNOW,*i*+1) PLOT(1)=*nam3* PLOT(2)=*nam4* ⋮ PLOT(*m*)=*namn* CALL GPLOT(PLOT,TNOW,*i*+2) ⋮ *label*: CONTINUE PLO,*i*,TIME,*j*,1,2,*val** VAR,*i*,1,,*nam1** PLO,*i*+1,TIME,*j*+1,1,2,*val** VAR,*i*+1,1,,*nam2*,1,1,*min1*,*max1** PLO,*i*+2,TIME,*j*+2,*m*,2,*val** VAR,*i*+2,1,,*nam3** VAR,*i*+2,2,,*nam4*,1,,*min2** ⋮ VAR,*i*+2,*m*,,*namn**	SSAVE Run Data Cards
	where: all the variables are as defined on the TABLE statement *m* is INTEGER≤10; number of variables to be plotted simultaneously	

Table 4.7(b) Translation of the PRTPLT Output Control Statement

Source Input		Generated Code	Generated Code Destination
INTEGER	*nam1,nam2,...*	INTEGER *nam1,nam2,...*	All subprograms
FIXED	*nam1,nam2,...*	INTEGER *nam1,nam2,...*	
REAL	*nam1,nam2,...*	REAL *nam1,nam2,...*	
LOGICAL	*nam1,nam2,...*	LOGICAL *nam1,nam2,...*	
EQUIVALENCE	*(nam1,nam2),...*	EQUIVALENCE *(nam1,nam2),...*	
COMMON	*/name/nam1,nam2,...*	COMMON/*name*/*nam1,nam2,...*	
DIMENSION	*nam1(n1,n2),...*	COMMON/UCOM*i*/*nam1(n1,n2),...*	
STORAGE	*nam1(m1,m2),...*	COMMON/UCOM*i*/*nam1(m1,m2),...*	

Table 4.8 Translation of the Declaration Statements

Source Input	Generated Code	Generated Code Destination
END	*label:* CONTINUE STA,(5),*j** LIM,(8),*k** CON,,,*n5*,*n6*,*n2*,*n3* or *n4** INI,1,(6),*n1** SIM* where: *j* is number of tables and print-plot requested in current run *k* is number of first order differential equations in the system model *n1*,*n2*,*n3*,*n4*,*n5*,*n6*=See TIMER statement format given in Table 4.6(a)	SSAVE Run Data Cards
STOP	GEN,(7)*runn** FIN*	Global Data Cards
ENDJOB	COMMON/UCOM*i*/RIRUN,*nam1*,*nam2*,... EQUIVALENCE *string* where: *nam1*,*nam2*,... are parameters, constants and initial conditions initialized in the INTLC subprogram *string* is the equivalence string as currently existing in the EQSTRN array	All subprograms

Table 4.9 Translation of the Translation Control Statements

Source File	DATA BLOCK	main	Subpro. INTLC	Subpro. STATE	Subpro. OTPUT	Subpro. SSAVE	Subpro. RERUN
F0	c GASP MAIN	✓					
F0	SUBROUTINE INTLC		✓				
F0	SUBROUTINE STATE			✓			
F0	SUBROUTINE OTPUT				✓		
F0	SUBROUTINE SSAVE					✓	
F0	SUBROUTINE RERUN						✓
F0	IMPLICIT REAL (A-Z)	✓	✓	✓	✓	✓	
F0	INTEGER NSET (2000)	✓					
F0	block A	✓	✓	✓	✓	✓	
F2	declaration block	✓	✓	✓	✓	✓	
F0	DIMENSION PLOT(10)					✓	
F0	COMMON QSET(1)	✓	✓	✓	✓	✓	
F0	COMMON/UCOM/RIRUN						✓
F2	common block	✓	✓	✓	✓	✓	
F0	block B	✓	✓	✓	✓	✓	
F0	block C						✓
F0	EQUIVALENCE(NSET(1),QSET(1))	✓					
F2	equivalence block	✓	✓	✓	✓	✓	
F2	main block	✓					
F2	INTLC block		✓				
F2	STATE block			✓			
F2	OTPUT block				✓		
F2	SSAVE block					✓	
F0	CALL GASP	✓					
F0	CALL EXIT	✓					
F0	RETURN		✓	✓	✓	✓	✓
F0	END	✓	✓	✓	✓	✓	✓

Table 4.10 The GASP program formation (Phase III)

Card type	Field Number	GASP IV Parameter name	Source of the data value(s) in the source program
GEN	7	NNRNS	Count of the number of END cards
STA	5	NNPLT	Count of the number of tables and plots specified via TABLE and PRTPLT cards
LIM	8	NNEQD	Count of the number of first order differential equations implied by the statements in the DYNAMIC segment
PLO	2-6	IP, LLABP(IP), IITAP(IP), NNVAR(IP), LLPLT	Specifications appearing on the TABLE and PRTPLT cards
	7	DTPLT(IP)	PDEL or TDEL (TIMER card)
VAR	2-9	IP, IJ, LLSYM(IJ), LLABP(IJ), LLPLO(IJ), LLPHI(IJ), PPLO(IJ),PPHI(IJ)	Specifications appearing on the TABLE and PRTPLT cards
CON	4	AAERR	Value assigned to ABSERR (TIMER card)
	5	RRERR	Value assigned to RELERR (TIMER card)
	7	DTMAX	Value assigned to DTMAX (TIMER card)
	8	DTSAV	Value assigned to TDEL or PDEL (TIMER card)
INI	6	TTFIN	Value assigned to FINTIM (TIMER card)

Table 4.11 GASP Data Card Synthesis

CHAPTER 5

Examples

To illustrate the use of ACMED.L, two example problems are described in this chapter. In each case, the mathematical model under study is outlined and the corresponding source code for ACMED.L is given. The subroutines generated by the ACMED.L for subsequent use by GASP are shown, together with the associated GASP data cards.

5.1 Example 1 :Water Regulatory System

This example is concerned with a simulation study of the body-water regulatory system as formulated by Harris [26]. The mathematical model for the system is summarized below:

$$
\begin{aligned}
ds/dt &= d_r(t) - k_1 s(t) && ; \quad s(0) = 0 \\
dg/dt &= k_1 s(t) - k_2(t) g(t) && ; \quad g(0) = 0 \\
dp/dt &= k_2(t) g(t) - 0.06u(t) && ; \quad p(0) = 3 \\
da/dt &= \delta(t) && ; \quad a(0) = 10 \\
k_2(t) &= 3(c_1 r(t) + c_2)/p(t) && ; \quad c_1 = 0.0097, \\
& && \qquad\quad c_2 = 3.0 \\
a_c(t) &= a(t)/p(t)
\end{aligned}
$$

where

s = volume of water in stomach (liters)
g = volume of water in intestine (liters)
p = volume of plasma (liters)
a = level of ADH (antidiuretic hormone) in plasma (milliunits)
k_2= intestinal loss rate parameter (hours^{-1})
a_c= concentration of ADH in plasma (milliunits/liter)
d_r= drinking rate (liters/hour)
r = osmolarity of ingested drink (milliosmols)
u = urine flow rate (milliliters/minute)

The rate of change of ADH level, $\delta(t)$ is taken to be

$$
\delta(t) = \begin{cases} -133.3p(t) + 420 - 2a(t) & \text{for } p(t) \leq 3 \\ -50p(t) + 170 - 2a(t) & 3 < p(t) \leq 3.4 \\ -2a(t) & p(t) > 3.4 \end{cases}
$$

and the urine flow rate is assumed to be given by

$$u(t) = \begin{cases} -6a_c(t)+20 & \text{for } a_c(t) \leq 3.17 \\ 1.0 & \text{otherwise} \end{cases}$$

The drinking rate $d_\pi(t)$ is taken to be constant over a ten minute period, after which it falls to zero;i.e.,

$$d_\pi(t) = \begin{cases} 6V & \text{for } 0 \leq t \leq 0.167 \text{ (hours)} \\ 0 & \text{otherwise} \end{cases}$$

where V is the total water consumed and is taken to be 1 liter.

It is presumed that the study is concerned with investigating the effects of changes in π (osmolarity) on the various system variables. The source ACMED.L program for the problem is given in Fig. 5.1. It has been written to provide plotted output for the cases of π = 300, 200 and 100. In addition, the resposes for these same three π values are obtained again for the case where $a(0)$ is changed from its nominal value of 10, to 8. The program illustrates the use of the CALL RERUN mechanism and the use of the multiple specification fields. Note also that the REALPL construct is used to illustrate an alternative means for specifying the $\delta(t)$ equation. The FORTRAN programs generated by the ACMED.L for the problem are shown in Fig. 5.2 and the data cards associated with the two specification fields in the source program, are shown in Fig. 5.3.

5.2 Example 2 :Pilot Ejection Problem

This example is the pilot ejection problem which has been widely used as a test problem for continuous system simulation software [8, 16, 27, 28]. The objective of the simulation study is to determine, for each of 10 aircraft velocities, the least altitude at which the pilot can eject and be sure to avoid a collision with the aircraft's vertical stabilizer. The central influence in this process is the aerodynamic drag on the ejected pilot which depends on his velocity and the air density (hence altitude).

The ejection process takes place in two phases. In the first the pilot and his seat travel along rails at a constant velocity V_e and at an angle θ_e backward from the vertical. Once the pilot-seat combination leaves the rails (this occurs when the vertical displacement equals y_1), it follows a ballistic trajectory. The mathematical model for this process is given below:

$$dV/dt = \begin{cases} 0 & \text{for } 0 \leq y(t) < y_1 \\ (D/m) - g \sin\theta(t) & \text{for } y(t) \geq y_1 \end{cases}$$

$$d\theta/dt = \begin{cases} 0 & \text{for } 0 \leq y(t) < y_1 \\ -g\cos\theta(t)/V(t) & \text{for } y(t) \geq y_1 \end{cases}$$

$$dx/dt = V(t)\cos\theta(t) - V_a$$

$$dy/dt = V(t)\sin\theta(t)$$

$$D = 0.5\, C_D S \rho V^2(t)$$

where

$x(t)$ = pilot horizontal position, relative to the aircraft (hence $x(0) = 0$)

$y(t)$ = pilot vertical position, relative to the aircraft (hence $y(0) = 0$)

V_a = aircraft velocity (assumed to be horizontal)

from the rail geometry it can be established that:

$$V(0) = [\ (V_e \cos\theta_e)^2 + (V_a - V_e \sin\theta_e)^2\ ]^{.5}$$

$$\theta(0) = \tan^{-1}[\ (V_e \cos\theta_e)/(V_a - V_e \sin\theta_e)\ ]$$

The remaining data for the problem is:

m =7 slugs , C_D =1 ,

S =10 ft^2 , y_1 = ft ,

g =32.2 ft/sec^2 , θ_e =15° ,

V_e =40 ft/sec ,

ρ =$\rho(h)$, the air density function which is dependent on altitude h and is provided in tabular form (slugs/ft^3).

The specific objective is to determine for each V_a value in the set (100, 200, ... 1000), the least value of h (in steps of 100 ft), for which the pilot's trajectory will miss the vertical stabilizer by at least 8 ft; the stabilizer location is taken to be 30 ft behind the cockpit.

The source ACMED.L program for this problem is given in Fig 5.4. The program illustrates, in particular, the use of the NLFGEN block (for accomodating the ρ function), the PROCEDURE construct (for handling the discontinuity that occurs when the pilot-seat combination leaves the rails) and the FINISH card (for specifying run termination conditions based on the position of the pilot-seat combination). The FORTRAN subprograms generated by the ACMED.L for GASP and related GASP data cards are shown in Fig. 5.5 and 5.6 respectivly.

```
*      RETRIEVE FROM *DRINKR
PARAM K1=4.K3=2.C1=-.0097.C2=3.P2=3.P3=3.41
PARAM V=1.TIN=0.167.R=300
INCON SZ=0.GZ=0.PZ=3.AZ=10
DYNAMIC
          S=REALPL(SZ.1/K1.DR/K1)
          G=INTGRL(GZ.K1*S-K2*G)
          P=INTGRL(PZ.K2*G-0.06*U)
          A=INTGRL(AZ.F)
          AC=A/P
          K2=3*(C1*R+C2)/P
*GENERATION OF F
          F=F1S+F2S-K3*A
          F1S=INSW(L1.0.F1)
          F2S=INSW(L2.0.F2)
          F1=-50*P+170
          F2=-83.3*P+250
          L1=COMPAR(P3.P)
          L2=COMPAR(P2.P)
*GENERATION OF U
          U=INSW(L3.U1.1)
          U1=20-6*AC
          L3=COMPAR(AC.3.17)
*GENERATION OF DR
          DR=INSW(L.6*V.0)
          L=COMPAR(TIME.TIN)
*
TIMER FINTIM=5.DTMAX=0.01.PDEL=0.1
PRTPLT S.G.P.A.U
*
TERMINAL
          IF(R.LT.150) GO TO 10
          R=R-100
          CALL RERUN
  10      CONTINUE
END
PARAM  AZ=8. R=300
PRTPLT S.G.P.A.U
END
STOP
ENDJOB
```

Fig.5.1 Source ACMED.L Program for Example 1

```
      INTEGER JEVNT,MFA,MFE,MLE,MSTOP,NCRDR,NNAPO,NNAPT,NNATR,NNFIL,NNQ,
     *NNTRY,NPRNT,ISEES,LFLAG,NFLAG ,NNEOD,NNEQS,NNEQT,IIEVT,IISED,JJBEG
     *,JJCLR,MMNIT,MMON,NNAME,NNCFI,NNDAY,NNPT,NNSET,NNPRJ,NNPRM,NNRNS,
     *NNRUN,NNSTK,NNYR
      INTEGER IIECK,IIPOF,IIPOS,IISUM,IIPIR,JJFIL,IIFIN,NRTOT,JBEGG
      LOGICAL L1,L2,L3,L
      COMMON QSET(50000)
      COMMON/UCOM9/AC,K2,F,F1S,F2S,F1,F2,L1,L2,U,U1,L3,DR,L
      COMMON/UCOM10/RIRUN,K1,K3,C1,C2,P2,P3,V,TIN,R,SZ,GZ,PZ,AZ
      COMMON /GCOM1/ ATRIB(25),JEVNT,MFA,MFE(100),MLE(100),MSTOP,NCRDR,N
     *NAPO,NNAPT,NNATR,NNFIL,NNQ(100),NNTRY,NPRNT,PPARM(50,4),TNOW,TTBEG
     *,TTCLR,TTFIN,TTRIB(25),TTSET
      COMMON /GCOM2/ DD(100),DDL(100),DTFUL,DTNOW,ISEES,LFLAG(50),NFLAG,
     *NNEOD,NNEQS,NNEQT,SS(100),SSL(100),TTNEX
      COMMON/GCOM5/IIEVT,IISED(6),JJBEG,JJCLR,MMNIT,MMON,NNAME(3),NNCFI,
     *NNDAY,NNPT,NNSET,NNPRJ,NNPRM,NNRNS,NNRUN,NNSTR,NNYR,SSEED(6)
      COMMON/MISC/IIECK,IIPOF,IIPOS,IISUM,IIPIR,JJFIL,IIFIN,NRTOT,JBEGG
      EQUIVALENCE  (NSET(1),QSET(1))
      EQUIVALENCE  (SS(  1),A),(SS(  2),P),(SS(  3),G),(SS(  4),S),(T
     *            IME,TNOW)
```

Fig. 5.2(a) Declaration Statements for Example 1

```
      C     MAIN PROGRAM
            IMPLICIT REAL (A-Z)
            INTEGER NSET(50000)
                  :
                  :
          Declaration Statements
                  :
                  :
            RIRUN   =0
            NNQ(1)  =0
            NCRDR   =9
            NPRNT   =6
            CALL    GASP
            CALL    EXIT
            END
```

Fig. 5.2(b) MAIN Program for Example 1

```
      SUBROUTINE INTLC
      IMPLICIT REAL (A-Z)
C
C     Declaration Statements
C
      IF(RIRUN.NE.0  )                GO TO 9111
      IF(NNRUN.NE.   1)               GO TO 9111
      K1=4
      K3=2
      C1=-.0097
      C2=3
      P2=3
      P3=3.41
 9111 CONTINUE
      IF(RIRUN.NE.0  )                GO TO 9112
      IF(NNRUN.NE.   1)               GO TO 9112
      V=1
      TIN=0.167
      R=300
 9112 CONTINUE
      IF(RIRUN.NE.0  )                GO TO 9113
      IF(NNRUN.NE.   1)               GO TO 9113
      SZ=0
      GZ=0
      PZ=3
      AZ=10
 9113 CONTINUE
      A=AZ
      P=PZ
      G=GZ
      S=SZ
      IF(RIRUN.NE.0  )                GO TO 9114
      IF(NNRUN.NE.   2)               GO TO 9114
      AZ=8
      R=300
 9114 CONTINUE
      RETURN
      END
```

Fig. 5.2(c) Subroutine INTLC for Example 1

```
      SUBROUTINE   OTPUT
      IMPLICIT REAL (A-Z)
C
C     Declaration Statements
C
      RIRUN   =0
      IIFIN   =0
      IF(R.LT.150) GO TO 10
      R=R-100
      CALL RERUN
   10 CONTINUE
      RETURN
      END
```

Fig. 5.2(d) Subroutine OTPUT for Example 1

```
      SUBROUTINE STATE
      IMPLICIT REAL (A-Z)
        Declaration Statements
      AC=A/P
      K2=3*(C1*R+C2)/P
      F1=-50*P+170
      F2=-83.3*P+250
      L1=.TRUE.
      IF(P3.LT.P)     L1=    .FALSE.
      F1S=0
      IF(L1)     F1S=    F1
      L2=.TRUE.
      IF(P2.LT.P)     L2=    .FALSE.
      F2S=0
      IF(L2)     F2S=    F2
      F=F1S+F2S-K3*A
      U1=20-6*AC
      L3=.TRUE.
      IF(AC.LT.3.17)     L3=    .FALSE.
      U=U1
      IF(L3)     U=    1
      L=.TRUE.
      IF(TIME.LT.TIN)     L=    .FALSE.
      DR=6*V
      IF(L)     DR=    0.
      DD(   1)=F
      DD(   2)=K2*G-0.06*U
      DD(   3)=K1*S-K2*G
      DD(   4)=1./(1/K1)*(DR/K1-S)
      RETURN
      END
```

Fig. 5.2(e) Subroutine STATE for Example 1

```
      SUBROUTINE SSAVE
      IMPLICIT REAL (A-Z)
        Declaration Statements
      IF(NNRUN.NE.  1)                GO TO 9111
      PLOT(  1)=S
      CALL      GPLOT(PLOT,TNOW,  1)
      PLOT(  1)=G
      CALL      GPLOT(PLOT,TNOW,  2)
      PLOT(  1)=P
      CALL      GPLOT(PLOT,TNOW,  3)
      PLOT(  1)=A
      CALL      GPLOT(PLOT,TNOW,  4)
      PLOT(  1)=U
      CALL      GPLOT(PLOT,TNOW,  5)
 9111 CONTINUE
      IF(NNRUN.NE.  2)                GO TO 9112
      PLOT(  1)=S
      CALL      GPLOT(PLOT,TNOW,  1)
      PLOT(  1)=G
      CALL      GPLOT(PLOT,TNOW,  2)
      PLOT(  1)=P
      CALL      GPLOT(PLOT,TNOW,  3)
      PLOT(  1)=A
      CALL      GPLOT(PLOT,TNOW,  4)
      PLOT(  1)=U
      CALL      GPLOT(PLOT,TNOW,  5)
 9112 CONTINUE
      RETURN
      END
```

Fig. 5.2(f) Subroutine SSAVE for Example 1

***** GENERATED DATA CARDS *****

```
GEN.(7)  2*
STA.(5)  5*
LIM.(8)  4*
PLO. 1.TIME.11. 1.2.0.1*
VAR. 1. 1..S*
PLO. 2.TIME.12. 1.2.0.1*
VAR. 2. 1..G*
PLO. 3.TIME.13. 1.2.0.1*
VAR. 3. 1..P*
PLO. 4.TIME.14. 1.2.0.1*
VAR. 4. 1..A*
PLO. 5.TIME.15. 1.2.0.1*
VAR. 5. 1..U*
CON.....0.01.0.1*
INI.1.(6)5*
SIM*
STA.(5)  5*
PLO. 1.TIME.11. 1.2.0.1*
VAR. 1. 1..S*
PLO. 2.TIME.12. 1.2.0.1*
VAR. 2. 1..G*
PLO. 3.TIME.13. 1.2.0.1*
VAR. 3. 1..P*
PLO. 4.TIME.14. 1.2.0.1*
VAR. 4. 1..A*
PLO. 5.TIME.15. 1.2.0.1*
VAR. 5. 1..U*
CON.....0.01.0.1*
INI.1.(6)5*
SIM*
FIN*
```

Fig. 5.3 Generated Data Cards for Example 1

```
*     RETRIEVE FROM #PILOTR

CONSTANTS M = 7.0.    CD = 1.0.    S = 10.0.   H=0    .   VA=0   .   ...
          Y1 = 4.0. G = 32.2.   TE = 15..   VE = 40..   PI = 3.14159

FUNCTION RHO = (0.0 . 2.377E-3) . (1.E3 . 2.308E-3) ....
               (2.E3 . 2.241E-3) . (4.E3 . 2.117E-3) ....
               (6.E3 . 1.987E-3) . (1.E4 . 1.755E-3) ....
               (1.5E4. 1.497E-3) . (2.E4 . 1.267E-3) ....
               (3.E4 . 0.891E-3) . (4.E4 . 0.587E-3) ....
               (5.E4 . 0.364E-3) . (6.E4 . .2238E-3)
INITIAL
          VZERO = SQRT ((VA-VES)**2 + VEC**2)
          TZERO = ATAN (VEC/(VA - VES))
          VEC=VE*COS(TERAD)
          VES=VE*SIN(TERAD)
          TERAD = TE/57.3
          K=0.5*CD*S*KK
          KK=NLFGEN(RHO,H)
NOSORT
          IF(TZERO.LT.0.) TZERO=TZERO + PI
*THE PRECEDING STMT ENSURES THE CORRECT HANDLING OF THE CASE WHERE VA LT VES

DYNAMIC
          X = INTGRL(0., VX - VA)
          Y = INTGRL(0., VY)
          V = INTGRL(VZERO,VDOT)
          THETA = INTGRL(TZERO,THEDOT)
          VX=V*COS(THETA)
          VY=V*SIN(THETA)
          D = K*(V**2)

PROCEDURE VDOT,THETADOT=F(D,Y,V,THETA)
          IF (Y .LT. Y1) GO TO 1
          GX=G*COS(THETA)
          GY=G*SIN(THETA)
          THEDOT = - GX / V
          VDOT = (-D/M) - GY
          GO TO 2
    1     THEDOT = 0.
          VDOT = 0.
    2     CONTINUE
ENDPRO

TERMINAL
          MISS = Y - 12.
          IF (MISS.GE.8) GO TO 3
          IF (X .GT. -30.0)  GO TO 3
          H = H + 1000.
          IF(H.LE.60000.) GO TO 6
          WRITE(6,8)
          GO TO 7
    3     WRITE(6,4) VA,H,MISS,TIME,X,Y
    5     IF(VA.GE.1000.) GO TO 7
          VA = VA + 100.
    6     CALL RERUN
    7     CONTINUE
    4     FORMAT(' VA =',F7.1,5X,'H =',F10.1,5X,'MISS =',F7.2,5X,...
         *'TIME =',F7.2,5X,'X =',F7.2,5X,'Y =',F7.2)

    8     FORMAT(///,' HEIGHT NOW GREATER THAN 60,000 FEET')

TIMER FINTIM= 5   , DTMAX=0.01
FINISH  X < -30.0.   Y<-0.5
END
STOP
ENDJOB
```

Fig. 5.4 Source ACMED.L Program for Example 2

```
      INTEGER JEVNT,MFA,MFE,MLE,MSTOP,NCRDR,NNAPO,NNAPT,NNATR,NNFIL,NNQ,
     *NNTRY,NPRNT,ISEES,LFLAG,NFLAG,NNEOD,NNEQS,NNEQT,IIEVT,IISED,JJBEG
     *,JJCLR,MMNIT,MMON,NNAME,NNCFI,NNDAY,NNPT,NNSET,NNPRJ,NNPRM,NNRNS,
     *NNRUN,NNSTR,NNYR
      INTEGER IIECK,IIPOF,IIPOS,IISUM,IIPIR,JJFIL,IIFIN,NRTOT,JBEGG
      COMMON QSET(50000)
      COMMON/UCOM9/VX,VY,D
      COMMON/UCOM10/RIRUN,M ,CO ,S ,H,VA,YI ,G ,TE ,VE ,PI ,RHO (12,2),
     *TERAD ,VES,VEC,VZERO ,TZERO ,KK,K
      COMMON /GCOM1/ ATRIB(25),JEVNT,MFA,MFE(100),MLE(100),MSTOP,NCRDR,N
     *NAPO,NNAPT,NNATR,NNFIL,NNQ(100),NNTRY,NPRNT,PPARM(50,4),TNOW,TTBEG
     *,TTCLR,TTFIN,TTRIB(25),TTSET
      COMMON /GCOM2/ DD(100),DDL(100),DTFUL,DTNOW,ISEES,LFLAG(50),NFLAG,
     *NNEOD,NNEQS,NNEQT,SS(100),SSL(100),TTNEX
      COMMON/GCOM5/IIEVT,IISED(6),JJBEG,JJCLR,MMNIT,MMON,NNAME(3),NNCFI,
     *NNDAY,NNPT,NNSET,NNPRJ,NNPRM,NNRNS,NNRUN,NNSTR,NNYR,SSEED(6)
      COMMON/HISC/IIECK,IIPOF,IIPOS,IISUM,IIPIR,JJFIL,IIFIN,NRTOT,JBEGG
      EQUIVALENCE  (NSET(1),QSET(1))
      EQUIVALENCE  (SS(  1),THETA ),(SS(  2),V ),(SS(  3),Y ),(SS(  4
     *            ),X ),(TIME,TNOW)
```

Fig. 5.5(a) Declaration Statements for Example 2

```
C     MAIN PROGRAM
      IMPLICIT REAL (A-Z)
      INTEGER NSET(50000)
                :
                :
      Declaration Statements
                :
                :
      RIRUN   =0
      NNQ(1)  =0
      NCRDR   =9
      NPRNT   =6
      CALL    GASP
      CALL    EXIT
      END
```

Fig. 5.5(b) MAIN Program for Example 2

```
      SUBROUTINE INTLC
      IMPLICIT REAL (A-Z)
                :
                :
        Declaration Statements
                :
                :
      IF(RIRUN.NE.0  )                GO TO 9111
      IF(NNRUN.NE.  1)                GO TO 9111
      M = 7.0
      CD = 1.0
      S = 10.0
      H=0
      VA=0
      Y1 = 4.0
      G = 32.2
      TE = 15.
      VE = 40.
      PI = 3.14159
 9111 CONTINUE
      RHO (01,1)=0.0
      RHO (01,2)=2.377E-3
      RHO (02,1)=1.E3
      RHO (02,2)=2.309E-3
      RHO (03,1)=2.E3
      RHO (03,2)=2.241E-3
      RHO (04,1)=4.E3
      RHO (04,2)=2.117E-3
      RHO (05,1)=6.E3
      RHO (05,2)=1.987E-3
      RHO (06,1)=1.E4
      RHO (06,2)=1.755E-3
      RHO (07,1)=1.5E4
      RHO (07,2)=1.497E-3
      RHO (08,1)=2.E4
      RHO (08,2)=1.267E-3
      RHO (09,1)=3.E4
      RHO (09,2)=0.891E-3
      RHO (10,1)=4.E4
      RHO (10,2)=0.587E-3
      RHO (11,1)=5.E4
      RHO (11,2)=0.364E-3
      RHO (12,1)=6.E4
      RHO (12,2)=.2236E-3
      TERAD = TE/57.3
      VES=VE*SIN(TERAD)
      VEC=VE*COS(TERAD)
      VZERO = SQRT ((VA-VES)**2 + VEC**2)
      TZERO = ATAN (VEC/(VA - VES))
      KK=SPLINE(12,RHO,H)
      K=0.5*CD*S*KK
      IF(TZERO.LT.0.) TZERO=TZERO + PI
      THETA =TZERO
      V =VZERO
      Y =0.
      X =0.
      RETURN
      END
```

Fig. 5.5(c) Subroutine INTLC for Example 2

```
      SUBROUTINE   OTPUT
      IMPLICIT REAL (A-Z)
                        :
                        :
                Declaration statements
                        :
                        :
      RIRUN   =0
      IIFIN   =0
      MISS = Y - 12.
      IF (MISS.GE.8) GO TO 3
      IF (X .GT. -30.0)  GO TO 3
      H = H + 1000.
      IF(H.LE.60000.) GO TO 6
      WRITE(6,8)
      GO TO 7
    3    WRITE(6,4) VA,H,MISS,TIME,X,Y
    5    IF(VA.GE.1000.) GO TO 7
      VA = VA + 100.
    6    CALL RERUN
    7    CONTINUE
    4    FORMAT(' VA =',F7.1,5X,'H =',F10.1,5X,'MISS =',F7.2,5X,'TIME ='
   4*,F7.2,5X,'X =',F7.2,5X,'Y =',F7.2)
    8    FORMAT(///,' HEIGHT NOW GREATER THAN 60,000 FEET')
      RETURN
      END
```

Fig. 5.5(d) Subroutine OTPUT for Example 2

```
      SUBROUTINE STATE
      IMPLICIT REAL (A-Z)
                        :
                        :
            Declaration statements
                        :
                        :
      VX=V*COS(THETA)
      VY=V*SIN(THETA)
      D = K*(V**2)
      IF (Y .LT. Y1) GO TO 1
      GX=G*COS(THETA)
      GY=G*SIN(THETA)
      THEDOT = - GX / V
      VDOT = (-D/M) - GY
      GO TO 2
    1    THEDOT = 0.
      VDOT = 0.
    2    CONTINUE
      DD(   1)=THEDOT
      DD(   2)=VDOT
      DD(   3)= VY
      DD(   4)= VX - VA
      IF(NNRUN.NE.   1)                    GO TO 9112
      IF(X .LE.-30.0)     MSTOP=-1
      IF(Y.LE.-0.5)     MSTOP=-1
      IF(MSTOP.EQ.-1)   ISEES=-1
 9112 CONTINUE
      RETURN
      END
```

Fig. 5.5(e) Subroutine STATE for Example 2

```
***** GENERATED DATA CARDS *****


GEN,(7)   1*
LIM,(8)   4*
CON,,,,,,0.01,*
INI,1,(6) 5   *
SIM*
FIN*
```

Fig. 5.6 Generated Data Cards for Example 2

CHAPTER 6

Conclusions and Further Work

There is, without question, sufficient need for such software packages as ACMED.L to justify their development. ACMED.L enables the user to specify, in a high level language, the continuous simulation problem under study. Translation of this source program into a GASP IV program is entirely automatic. The user is thus relieved from many of the tedious and error prone aspects of GASP IV program preparation.

The ACMED.L language provides the user with a number of useful model specification features and frees the user from most of the formalities that are otherwise required in preparing a GASP IV program for a continuous system simulation study. The following are specifically noted:

a) The user is free to use meaningful identities for the problem state variables and their derivatives;

b) The COMMON and EQUIVALENCE statements in sub-programs; MAIN, INITIAL, STATE, SSAVE, OTPUT, are automatically generated;

c) The model specification statements can appear in arbitrary order (a sorting mechanism ensures correct re-ordering);

d) A variety of functional building blocks are provided for the user; e.g. INTGRL, REALPL, etc.

e) The PROCEDURE construct provides a mechanism for encapsulating a group of source statements so that they are treated as a single functional block.

f) Flexible mechanisms are provided for carrying out multiple runs.

g) An easy-to-use interpolation procedure is provided to handle functional relations defined by a finite set of data points.

A significant foundation has been established in this thesis project for further development of the ACMED.L preprocessor.

The following is a list of the possible further extensions to ACMED.L.

a) Extending the SORT routines to sort the absissa-ordinate pairs' independent variable in an ascending order (the FUNCTION statement).

b) Inclusion of a "vector" integration routine.

c) Inclusion of a "MACRO" feature.

d) Inclusion of some contemporary concepts; e.g. a separate section to specify the experiments which are distinct from the model specification section.

e) Some naive user-proffing might be a worthwhile enhancement; e.g. the TDEL specification could have a default value to protect the user.

In addition, a particular project that merits consideration is the extension of the current system to include an equivalent interface for handling discrete models.

APPENDIX A

GASP Data Cards

Description of GASP Data Cards [5]

Card type GEN contains heading, run identification, and data suppression information.

Card types STA and LIM contain variables describing the size of the model being run.

Card types PLO and VAR provide information for plotting output through the use of subroutine GPLOT. The PLO card defines the variables applicable to the plot or table. VAR cards are associated with each of the dependent variables to be plotted or tabled.

Card type CON is associated with state variables. Because a step can never be greater than DTSAV, DTMAX is equated to DTSAV if it is larger than a positive DTSAV.

Card type INI contains basic run control variables. The variables MSTOP, JJCLR, TTBEG, and TTFIN relate to the method of stopping the simulation, clearing of statistics, the beginning time for the simulation and the ending time for the simulation.

A SIM card is used to indicate the end of GASP data for one run. Data read in subroutine INTLC would follow a SIM or FIN card. A FIN card indicates that there are no more GASP data cards.

Table A shows in detail each data card's structure for the case of a continuous simulation model.

Card Type	Field Number	GASP Variable Initialized	Definition	Default
GEN	1			
	2	NNAME	User's name	WHOM
	3	NNPRJ	Project number	580
	4	MMON	Month number	1
	5	NNDAY	Day number	1
	6	NNYR	Year number	2001
	7	NNRNS	Number of simulation runs to be made	1
	8	NPRT2	Number of a scratch tape to be used in processing free form input. This number should not be the number of any tapes used for plotting (IITAP(IP)).	7
	9	IIECH	Print suppression key: YES → print listing of input cards, input error messages if any, and GASP IV echo check; NO → skip printing	Y
	10	IIPOF	Print suppression key: YES → print initial entries in files if any; NO → skip printing	Y
	11	IIPOS	Print suppression key: YES → print initial values of SS(.) and DD(.) if any; NO → skip printing	Y
	12	IISUM	Print suppression key: YES → call SUMRY at end of simulation to obtain final report; NO → skip printing of final report	Y
	13	IIPIR	Print suppression key: YES → print heading "INTERMEDIATE RESULTS"; NO → skip printing of heading	Y
STA	1			

Table A.1

Definitions for GASP Variables Initialized by Data Inputs

Card Type	Field Number	GASP Variable Initialized	Definition	Default
STA	5	NNPLT	Number of plots/tables	largest IP
LIM	8	NNEQD	Number of first order differential equations in the model.	0
PLO	1			
	2	IP	Index	
	3	LLABP(11)	Label associated with independent variable for IPth plot	IND VAR
	4	IITAP(IP)	Index of the tape on which data for plot IP is stored. If IITAP(IP)=0, data is stored in QSET	0
	5	NNVAR(IP)	Number of variables to be plotted/ tabled in plot/table IP	largest IJ
	6	LLPLT	Key for specifying type of table/ plot. Values for LLPLT: 0→plot only; 1→table only; > 1→ table and plot	0
	7	DTPLT(IP)	The increment of the independent variable between successive plot points for plot IP	5
VAR	1			
	2	IP	Plot number	
	3	IJ	Index for IJth dependent variable	
	4	LLSYM(IJ)	Plot symbol for variable IJ for plot IP	IJth letter
	5	LLABP(IJ)	Label associated with variable IJ of plot IP	IJth letter
	6	LLPLQ(IJ)	Key for specifying the lower limit of the scale for variable IJ of plot IP. Values of LLPLO(IJ) are: 0 → use minimum from simulation; 1 → use PPLO(IJ); 2 → use minimum from simulation rounded to nearest PPLO(IJ)	0
	7	LLPHI(IJ)	Key for specifying upper limit of the scale for variable IJ of plot IP. Keys are similar to LLPLO(IJ) above.	0
	8	PPLO(IJ)	Value associated with lower limit of plot ordinate	0
	9	PPHI(IJ)	Value associated with upper limit of plot ordinate	0

Table B.1 (cont.)

Card Type	Field Number	GASP Variable Initialized	Definition	Default
CON	1			
	3	LLERR	Dey to indicate severity of error when accuracy cannot be maintained by the Runge-Kutta integration package -1→proceed without warning message 0→proceed with warning message +1→fatal error Accuracy is required to be less than or equal to AAERR+RRERR*SS(I) for all I.	0
	4	AAERR	Absolute local truncation error allowed in Runge-Kutta integration	.00001
	5	RRERR	Relative error allowed in Runge-Kutta integration	.00001
	6	DTMIN	Minimum step size permitted	.01*DTMAX
	7	DTMAX	Maximum step size permitted	$\min[DTSAV, 10^{20}]$
	8	DTSAV	Time between communication points if positive. If DTSAV=0, communication to occur at each accepted update point. If DTSAV < 0, communication only at event time.	10^{20}
INI	1			
	2	MSTOP	Key for specifying method of stopping: 0→end of simulation event provided by user; 1 → stop at TTFIN	1
	4	JJBEG	Key for initializing DRAND,TNOW and state variables: NO → do not initialize YES → initialize	Y

Table B.1 (cont.)

Card Type	Field Number	GASP Variable Initialized	Definition	Default
INI	5	TTBEG	Initial value of TNOW	0
	6	TTFIN	Ending time of simulation if MSTOP -> 0	10^{20}
	7	JJFIL	Key for initializing file system: NO → do not initialize YES → initialize	Y

Table B.1 (cont.)

APPENDIX B

The Reserved Words

B.1 ACMED.L Reserved Words

(i) Reserved variable names:

Since the ACMED.L preprocessor checks only the first four characters of a keyword, a user must avoid using any variable having the same four leading characters as the following keywords.

ABSERR,CONSTANT,DYNAMIC,END,ENDJOB,ENDPRO,FINISH,FIXED,
INCON,INITIAL,LIST,NOSORT,PARAMETER,PRTPLT,PROCEDURE,
RELERR,SORT,STOP,STORAGE,TABLE,TERMINAL,
TIMER,ASK100, ASK101, ...

(ii) Reserved function names:

The use of following symbolic names as user written function names is not allowed since these represent the set of ACMED.L function blocks:,

AND,COMPAR,COMXPL,EOR,EQUIV,FCNSW,INSW,INTGRL,
IOR,LEDLAG,LIMIT,MODINT,NAND,NLFGEN,NOR,NOT,
OUTSW,RAMP,REALPL,STEP

B.2 GASP Reserved Words [5]

(i) Reserved variable names:

To avoid confusing user variables with GASP variables the user should observe the following rules:

1- Never begin a variable name with the letters DT.

2- Never employ a user variable with the same first two letters.

3- Never use the following as user variables: ATRIB,ISEES, JEVNT,LFLAG,MFA,MFE,MLE,MSTOP,NCRDR,NFLAG,NPRNT,NSET, QSET, and TNOW.

(ii) Reserved function names:

The use of the following symbolic names as names for user-written functions is not allowed since functions with these names are included in GASP: BETA,DPROB ,ERLNG,EXPON,GAMA, NPSSN,RLOGN,RNORM,TRIAG,UNFRM,WEIBL.

B.3 Reserved Labels

The numbers from 9111 to 9111+n , where n is the number of runs, are reserved for the preprocessor generated code.

The unintentional use of reserved words or labels by the user may be detected during the translation/compilation phase. In such a case, the program execution will be terminated and an appropriate error message provided. However, it is possible that such usage will go undetected during translation/compilation and this can lead to execution errors that could be extremely different to locate.

APPENDIX C

Program Preparation

To run an ACMED.L job, the following two load modules are required:

1- GASPIV load module

2- ACMED.L load module

The creation of the latter is described below.

C.1 Creation of the ACMED.L Load Module

To produce an ACMED.L load module, the following steps are to be followed:

1- Storing the system file F0 on unit FT10F001, i.e. the data set AHMED.TABLE. Fig. C.1 shows the program for executing this step.

2- FORT - Compiling the ACMED.L processor package under FORTRAN compiler.

3- LKED - Linkage of the ACMED.L source with FORTLIB to form a load module.

The flowchart and program for the last two steps are shown in Fig. C.2 and C.3 respectivly. The system file F0 together with the ACMED.L load module, i.e. $SL.BIRTA.CSP.LOAD(CSP) form an executable module (See Appendix C.2 (i)).

C.2 Execution of an ACMED.L Job

To execute an ACMED .L job the following four steps are cerried out:

1- CREAT - Creation of a GASP program and the necessary data cards by executing the user ACMED.L source program under the ACMED.L load module.

2- FORT - Compiling the GASP program under FORTRAN compiler.

3- LKED - Linkage of the GASP program with both GASPIV load module and FORTLIB.

4- GO - Execution of the generated program

These four steps are catalogued in a procedure called GASPX. The flowchart for this procedure is shown in Fig. C.4 and the procedure itself is shown in Fig. C.5.

C.3 ACMED.L Table/Print-plot Files

Each table or print-plot required in the jth specification field is associated with a distinct file. These files must be properly defined by the user (See section 2.4). These files must carry the logical unit numbers starting at 11 up to 20 sequentially; e.g. in example 1 (See section 5.1) the maximum number of tables and print-plots in any specification field is four. Therefore the user has to define four files carrying the logical numbers 11, 12, 13, and 14. In addition the user has to define a file carrying the logical number 7. This file is used by GASP in processing the free format data cards. Fig. C.6 shows the JCL required for executing example 1.

Note that the existance of DUMMY='' in the JCL's EXEC statement will cause the output of the CREAT step (See Appendix C.2) to be printed. On the other hand if the DUMMY='' doesnt exist, the CREAT step output is not printed.

C.4 Multiple Runs Output Identification

A run output is identified by a run number. If a CALL RERUN was encountered then the final output of each run is identified by the output headed by the first occurance of the run number (ignore run number 0, if appeared);e.g. if the run numbers are the following:

0, 1, 1, 2, 2, 2, 3

then the output identified by the undelined run numbers are the final output of each run.

```
//ASKSL006 JOB (
//        ),' AHMED K.          ',MSGLEVEL=(1,1),
// CLASS=J
// EXEC USERDEL,DSN='$SL.AHMED.TABLE'
// EXEC FORTGCLG
//FORT.SYSIN DD *
      INTEGER *2    BUFER(80),DIGIT(10)
      INTEGER       KIWORD(24,2),KIWRD2(20,2),KIWRD4( 4,2),INDX(5)
      INTEGER       IN/5/,OUT/6/,F0/10/
      INTEGER       N/116/,M/24/,M2/20/,M4/4/
      DEFINE FILE 10( 125,80,E,INF0)
      INF0    =1
5     FORMAT(80A1)
      DO 10 I=1,N
      READ (IN,5) BUFER
      WRITE(OUT,25)  BUFER
      WRITE(F0'INF0,5) BUFER
10    CONTINUE
      READ (IN,20) ((KIWORD(I,J),J=1,2),I=1,M),
     *              ((KIWRD2(I,J),J=1,2),I=1,M2) ,
     *              ((KIWRD4(I,J),J=1,2),I=1,M4)
      WRITE(F0'INF0,20)((KIWORD(I,J),J=1,2),I=1,M),
     *              ((KIWRD2(I,J),J=1,2),I=1,M2) ,
     *              ((KIWRD4(I,J),J=1,2),I=1,M4)
      WRITE(OUT,30)((KIWORD(I,J),J=1,2),I=1,M),
     *              ((KIWRD2(I,J),J=1,2),I=1,M2) ,
     *              ((KIWRD4(I,J),J=1,2),I=1,M4)
      READ (IN,35)INDX
      WRITE(F0'INF0,35)INDX
      WRITE(OUT,40)INDX
20    FORMAT(3(12(A4,I2)/),12(A4,I2))
25    FORMAT(5X ,80A1)
30    FORMAT((5X,12(A4,I2))/)
35    FORMAT(5 I5)
40    FORMAT(5X,5 I5)
      STOP
      END
//GO.FT10F001 DD UNIT=3330,VOL=SER=USER10,DISP=(,KEEP),
//            DSN=$SL.AHMED.TABLE,DCB=(RECFM=F,BLKSIZE=80),
//            SPACE=(TRK,3)
//GO.SYSIN DD *
```

Contents of File F0 (See section 3.2 (a))

Fig. C.1 Program for Creating the System File F0

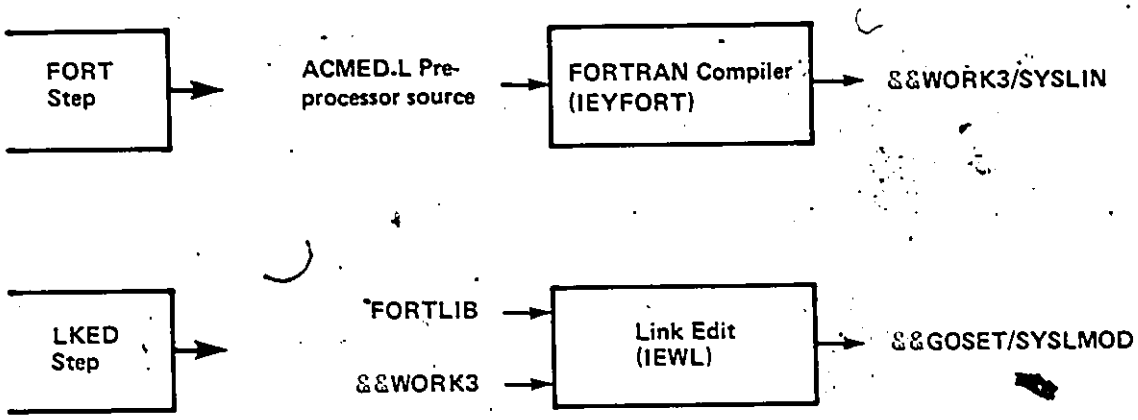


Fig C.2 Flowchart for Creating the ACMED.L Load Module

```
//ASKSL019 JOB (                .),AHMED,MSGLEVEL=(1,1),CLASS=B
***ROUTE PRINT LOCAL
***ROUTE PUNCH LOCAL
// EXEC FORTGCL,TIME=3,PARM.LKED='LIST,LET,MAP,DCBS'
//FORT.SYSIN DD *
```

The ACMED.L Processor FORTRAN Source code

```
//LKED.SYSLMOD DD DSN=$SL.BIRTA.CSP.LOAD(CSP),DISP=(OLD,KEEP),
// UNIT=3330,VOL=SER=USER10,DCB=BLKSIZE=13030,
// SPACE=(CYL,(01,1,2),RLSE)
//LKED.SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(40,10))
//
```

Fig. C.3 Program for Creating ACMED.L Load Module

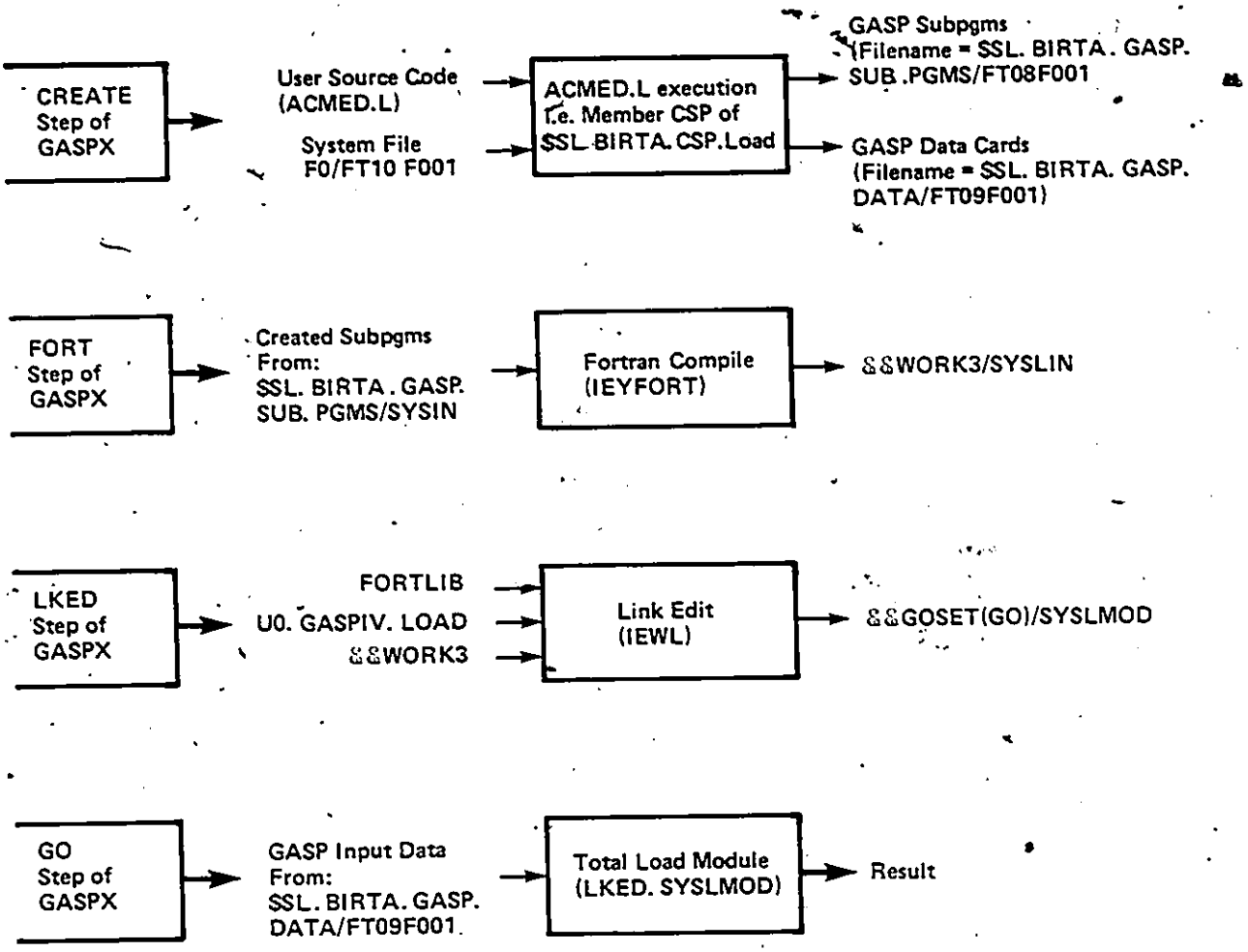


Fig C.4 Flowchart for the Catalogued Procedure GASPX

```
//ASKSL011 JOB           ,AHMED,CLASS=J
***MESSAGE          PLEASE REPLY U
***ROUTE PRINT LOCAL
***ROUTE PUNCH LOCAL
//    EXEC PGM=IEBUPDTE,REGION=100K
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=USER.PROCLIB,DISP=SHR
//SYSUT2 DD DSN=USER.PROCLIB,DISP=SHR
//SYSIN DD DATA
./   REPL NAME=GASPX
//GASPX PROC    PROG=IEYFORT,DUMMY='DUMMY '
//CREATE    EXEC PGM=CSP,REGION=160K
//STEPLIB   DD DSN=$SL.BIRTA.CSP.LOAD,UNIT=3330,VOL=SER=USER10,
//             DISP=SHR
//FT01F001  DD DDNAME=SYSIN
//FT06F001  DD SYSOUT=A
//FT08F001  DD DSN=$SL.BIRTA.GASP.SLB.PGMS,DISP=(,PASS),
//             SPACE=(3120,(200,50)),UNIT=3330,VOL=SER=USER10,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//FT09F001  DD DSN=$SL.BIRTA.GASP.DATA,DISP=(,PASS),
//             SPACE=(3120,(40,50)),UNIT=3330,VOL=SER=USER10,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//FT10F001  DD DSN=$SL.AHMED.TABLE,UNIT=3330,
//             VOL=SER=USER10,DISP=SHR
//FT11F001  DD DSN=&&TEMP,UNIT=SYSDA,SPACE=(3120,(50,10)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//FT12F001  DD DSN=&&TEMP,UNIT=SYSDA,SPACE=(3120,(50,10)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//FORT    EXEC PGM=&PROG,COND=(4,LT,CREATE),REGION=100K,ACCT=GASPIV
//SYSIN     DD DSN=$SL.BIRTA.GASP.SUE-PGMS,DISP=(OLD,DELETE)
//SYSLIN    DD DSN=&&WORK3,UNIT=SYSDA,SPACE=(1600,400),
//             DISP=(,PASS),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSPRINT  DD &DUMMY,SYSOUT=A
//LKED EXEC PGM=IEWL,PARM='LIST,MAP,LET,SIZE=(120K,24K)',REGION=120K,
//             COND=((4,LT,CREATE),(4,LT,FORT))
//SYSLIB    DD DSN=SYS1.FORTLIB,DISP=SHR
//          DD DSN=UO.GASPIV.LOAD.VOL00178,UNIT=3330,
//             VOL=SER=SYSVOL,DISP=SHR
//SYSLIN    DD DSN=&&WORK3,DISP=(OLD,DELETE)
//SYSLMOD   DD DSN=&&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),
//             SPACE=(TRK,(60,,1))
//SYSPRINT  DD &DUMMY,SYSOUT=A
//SYSUT1    DD DSN=&&WORK1,UNIT=SYSDA,SPACE=(1600,400)
//GO      EXEC PGM=*.LKED.SYSLMOD,REGION=120K,
//             COND=((4,LT,CREATE),(4,LT,FORT),(4,LT,LKED))
//FT06F001  DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=399)
//FT07F001  DD DSN=&&WORK2,UNIT=SYSDA,SPACE=(1600,400),
//             DCB=(RECFM=VBS,LRECL=20008,BLKSIZE=7294)
//FT09F001  DD DSN=$SL.BIRTA.GASP.DATA,DISP=(OLD,DELETE)
```

Fig C.5 The Catalogued Procedure GASPX

```
//ASKSL026 JOB (          .....1821,2),AHMED,MSGLEVEL=1,CLASS=B
//    EXEC GASPX,REGION.CREATE=200K,REGION.GO=200K,DUMMY=***
//SYSIN     DD *
                :
                :
                :
                :
```

Source ACMED.L Program (Example 1)

```
                :
                :
                :
//GO.FT07F001 DD DCB=(RECFM=VBS,LRECL=20008,BLKSIZE=400)
//GO.FT11F001 DD UNIT=SYSDA,SPACE=(1600,400),DSN=&AHMEDA,
//            DCB=(RECFM=VBS,LRECL=20008,BLKSIZE=400)
//
//GO.FT12F001 DD UNIT=SYSDA,SPACE=(1600,400),DSN=&AHMEDB,
//            DCB=(RECFM=VBS,LRECL=20008,BLKSIZE=400)
//
//GO.FT13F001 DD UNIT=SYSDA,SPACE=(1600,400),DSN=&AHMEDC,
//            DCB=(RECFM=VBS,LRECL=20008,BLKSIZE=400)
//
//GO.FT14F001 DD UNIT=SYSDA,SPACE=(1600,400),DSN=&AHMEDD,
//            DCB=(RECFM=VBS,LRECL=20008,BLKSIZE=400)
//
//GO.FT15F001 DD UNIT=SYSDA,SPACE=(1600,400),DSN=&AHMEDE,
//            DCB=(RECFM=VBS,LRECL=20008,BLKSIZE=400)
//
//
```

Fig. C.6 The JCL for Executing Example 1

APPENDIX D

Debugging Features

The ACMED.L system has been extensively tested during the course of this work. This was done by carrying out many experiments as each new feature was incorporated.

In order to facilitate the location of (as yet) undetected "bugs" within the ACMED.L preprocessor, a trace feature has been incorporated. This feature provides a comprehensive summary of the processing that takes place during Phase II.

The trace is activated by including in the source code, a card with the keyword TRACE. Such a card causes the preprocessor to trace the translation activities for all the source code statements that follow the TRACE statement. (Note that the trace feature cannot be reset and it is not "retroactive".)

The trace provides a very large amount of information. For example, if the translation of a statement causes the invocation of subprograms X1, X2, X3, ... Xn of the preprocessor, then the trace feature provides the input and output of each of these subprograms. It also provides information about the interval pointers of file F2 and the stages in which the newly generated statements are created.

The trace feature can be useful in debugging and in addition, can be useful to those who are concerned with altering or extending the ACMED.L preprocessor. Its use can produce a very large output file and it can significantly slow down the processing. This feature should therefore only be used in special circumstances. Fig. D.1 shows a sample output generated from a trace file.

```
==> PTIFGO:                                        Subroutine PTIFGO in
 RUNNUM=     1                                       Arguments and values
==> NONBLK:
 SPTR=     1 INDI=     0 IEND=    72
INCON         EEEE=34567            .FFFF=89012     Input ACMED.L statement
==> NONBLK:
 SPTR=     1 INDI=     0 IEND=    72
INCON         EEEE=34567            .FFFF=89012
==> IPACK:
 WORD=CONS   STOR=INCO
==> IPACK:
 WORD=INCO   STOR=
==> CHEK:
 WORD= INCO IFLAG=     0
 KIWORD=INIT     1
 KIWORD=PARA     2
 KIWORD=CONS     2
 KIWORD=INCO     2
 KIWORD=FIXE     3
 KIWORD=STOR     4
 KIWORD=DIME     4
 KIWORD=FUNO     5
 KIWORD=TABL     6                                 Search Table
 KIWORD=DYNA     7
 KIWORD=TERM     8
 KIWORD=PRIN     9
 KIWORD=PRTE    10
 KIWORD=TIME    11
 KIWORD=ABSE    11
 KIWORD=RELE    11
 KIWORD=FINI    12
 KIWORD=END     13
 KIWORD=STOP    14
==> CHEK:
 WORD= INCO IFLAG=     2
==> NOTOBF:
 INBUF=    41 IBASE= 9113 LEN=     4
       IF(RIRUN.NE.0  )              GO TO          Output statement 1
                                                      generation step 1
==> NOTOBF:
 INBUF=    41 IBASE= 9114 LEN=     4
       IF(RIRUN.NE.0  )              GO TO 9113     generation step 2
==> NOTOBF:
 INBUF=    19 IBASE=    11 LEN=     3
       IF(NNRUN.NE.   )              GO TO 9113     Output statement 2
==> NOTOBF:
 INBUF=    19 IBASE=    21 LEN=     3
       IF(NNRUN.NE.   1)             GO TO 9113     Subroutine PTIFGO out
==> PTIFGO:
 RUNNUM=     1
==> NONBLK:
 SPTR=     6 INDI=     0 IEND=    72
INCON         EEEE=34567            .FFFF=89012
==> NONBLK:
 SPTR=    13 INDI=     0 IEND=    72
INCON         EEEE=34567            .FFFF=89012
==> COPY:
 LENTH=    1 OSPTR=    13 LOCAT=     7
INCON         EEEE=34567            .FFFF=89012


==> COPY:
 LENTH=    1 CSPTR=    13 LOCAT=     7
INCON         EEEE=34567            .FFFF=89012
      EEEE=34567                                   Output statement 3
File F2 pointers     24        130        220        322        420        433
```

Fig. D.1 Sample Output from a TRACE File

REFERENCES

[1] HURST, N. R. and PRITSKER, A. A. B., "GASP IV:a Combined Continuous/ Discrete, FORTRAN Based Simulation Lnaguage", Proceedings, 1973 Winter Simulation Conference, San Francisco, Calif., Jan. 17-19 1973A pp 795-803

[2] PRITSKER, A. A. B. AND HURST, N. R., "A Manual for GASP IV", Purdue University, West Lafayette, Indiana, Mar 1973A

[3] HURST, N. R., "GASP IV:a Combined Continuous/Discrete, FORTRAN Based Simulation Language", Ph. D. Dissertation, Purdue University, West Lafayette, Indiana

[4] PRITSKER, A. A. B. and HURST, N. R., "GASP IV:a Combined Continuous-Discrete FORTRAN Based Simulation Language", Simulation, vol 21, No. 3, Sep. 1973B, pp 65-70

[5] PRITSKER, A. A. B., "The GASP IV Users Manual", Pritsker & Associates Inc., 1710 South Street, Lafayette, Indiana, 1973

[6] PRITSKER, A. A. B., "GASP IV Can Broaden Your Modeling Prespectives", Simuletter, vol. 5, No. 4, Jul 1974

[7] WORTMAN, D. B., "An Introduction to GASP IV:a Combined Continuous-Discrete Simulation Language", Simuletter, vol. 6, No. 1, Oct. 1974 pp 60-64

[8] PRITSKER, A. A. B., "The GASP IV Simulation Language", John Willy & Sons, Inc., New York, 1974

[9] FOX, M. A. and PRITSKER, A. A. B., "Interactive Simulation with GASP IV on a Minicomputer", Simuletter, vol. 6, No. 2/3, Jan.-Apr. 1975

[10] WORTMAN, D. B., "Simulation with GASP IV:a Combined Discrete-Continuous Simulation Language", Soc. Computer Simulation, 9th Annual Simulation Symposium, Tampa, Fla., Mar. 1976, pp 47-59

[11] MOORE, L. J.;LEE, S. M. and TAYLOR, B. W.,"Combined Continuous-Discrete System Simulation with GASP IV", Computer and Operations Research, vol. 4, No. 2, 1977 pp 129-137, Coden:CMORAP

[12] OREN, T.I., "Software Simulation of Combined Continuous and Discrete Systems: a State-of-the-Art Review", Simulation, Vol. 28, No. 2, Feb 1977, pp 33-45

[13] CELLIER, F.E. and BONGULIELMI, A.P., "The COSY Simulation Language", Proceedings of IMACS Congress 1979, Sorrento, Sept 24-28, 1979, pp 271-281

[14] KIVIAT, P.J., MARKOWITZ, H.M. and VILLANUEVE, R., "SIMSCRIPT II.5 Programming Language", (Edited by RUSSEL, E.C.), CACI, 1973

[15] System/360 Continuous System Modelling Program, User's Manual, IBM #GH20-0367-4 (Fifth Edition), Jan 1972

[16] SPECKHART, F.H. and GREEN, W.G., "A Guide to Using CSMP", Prentice-Hall, 1976

[17] PUGH, A.L., "DYNAMO Users' Manual", The M.I.T. Press. Mass. Instit. of Tech., Cambridge, Mass., 1963

[18] AGNEW, D., "SCAMPER, Circuit Design for the 1980s", Telesis, 1980-Three, Publication of Bell-Northern Research Ltd., 1980

[19] QUEDNAU, H.D., "Extension of LIBAFORM for Generating Programs in High Level Programming Language", ANGEW. INF., Vol. 22, No. 5, May 1980, pp 194-196

[20] "CPS, Circuit Pack System", Bell-Northern Research Report, Nov 1978.

[21] SHEW, E. and WILCOX, P., "F/LOGIC Simulation, Designing Better Circuits", Telesis, 1980 - Three, Publication of Bell-Northern Research Ltd. 1980

[22] "The CSI Continuous System Simulation Language (CSSL)", Simulation, Dec 1967, pp 281-303

[23] BIRTA, L., KHADR, A.S.Z. and HARIA, R., "A CSSL Interface to GASP IV", Proceedings, 1979 Summer Computer Simulation Conference, Toronto, Ontario, July 16-18, 1979, pp 20-31

[24] SHAH, R.R., "CATS, A Software Aid for Analog/Hybrid Programming", M.A.Sc. Thesis, University of Ottawa, Ottawa, Ontario, Aug 1976

[25] STROUD, A.H., "Numerical Quadrature and Solution of Ordinary Differential Equations", Springer-Verlag, 1974

[26] HARRIS, G.R., "A Model of the Body Water Regulatory System", Coed Transactions, Vol. 5, No. 2, Feb 1973, pp 13-26

[27] KORN, G.A. and WAIT, J.V., "Digital Continuous System Simulation", Prentice-Hall, 1978

[28] "Advanced Continuous Simulation Language (ACL), User Guide/Reference Manual", Mitchell and Gauthier Associates, Concord, Mass., 1975

VITA

Name: Ahmed Saied Zaki Khadr

Born: Egypt
March 1, 1948

Education:

Bachelor of Science (Honours) 1970
Department of Electrical Engineering
University of Ain Shams
Cairo
Egypt

Enrolled in M.A.Sc. program in Electrical Engineering at University of Ottawa from Sept 1977 to June 1981