A PREPROCESSOR FOR A REAL-TIME DIGITAL COMPUTER

by

Richard Lee Bulle
B.S. Phys. University of Tulsa, 1963

A Thesis Submitted to the Faculty of
School of Engineering and Applied Science
of the George Washington University in Partial Satisfaction
of the Requirements for the Degree of Master of Science

February 1971

Thesis directed by

James Dean Harris, Ph.D.

Professional Lecturer in Engineering

ABSTRACT

One of the major functions of the National Aeronautics and Space Administration's Langley Research Center Computer Complex is to provide computational support for real-time flight investigations. For purposes of efficiency, several real-time application programs operate concurrently in a single Control Data Corporation 6000 series computer. To perform "man in the loop" digital simulation requires that the computer operates as part of a closed loop, time critical system where precise problem solution rates must be guaranteed in order to maintain the integrity of t'e solution.

For ease of operation and programing, a real-time digital simulation supervisor (supervisor) was written to interface between a Fortran simulation program and the real-time system. It provides all the real-time input/output control, timing and synchronization, communication, control and other system functions unique to real-time operations. To maintain the flexibility and computational speed needed in real-time simulation, supervisor was written as a series of interdependent subroutines that are loaded with each real-time program. This provides the programer the option of using only those features of the system that he needs. In addition to supervisor, there are two other subroutines that are used by most simulation programs. They provide integration and the capability of displaying, changing and recording on a typewriter the value of problem variables.

The objective of this project was to develop a real-time digital preprocessor (preprocessor) that would minimize the programing involved in writing a real-time simulation by providing a meta language to Fortran. The preprocessor would parse this meta language and develop a real-time Fortran program. This would allow the programer to communicate to the preprocessor his real-time requirements in a unified manner and place the burden on the preprocessor to write all the subroutine calls necessary for real time.

A brief description of the software and hardware necessary to support digital simulation is presented. The present method of writing a real-time program and a new method using the preprocessor are compared. The Syntax, Semantics and Pragmatics of the preprocessor's language are defined and discussed. The implementation of the preprocessor is discussed and in the appendixes the complete syntax, a flow chart of a major subroutine and sample input and output programs are included.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

The author wishes to thank Mr. Joseph W. Young of National Aeronautics and Space Administration, Langley Research Center for his advice and guidance in establishing the guidelines for a real-time preprocessor, and his advice concerning the research that followed. The author would also like to thank Dr. James O. Harris for his advice and guidance in preparing this thesis.

LIST OF FIGURES

LIST OF SYMBOLS

ADC	Analog to Digital Converter
ADCON	Analog to Digital Controller
ADDIS	Analog to Digital and Discrete Input System
BNF	Backus Naur Form
CDC	Control Data Corporation
CRT	Cathod Ray Tube
DAC	Digital to Analog Converter
DACON	Digital to Analog Controller
DADOS	Digital to Analog and Discrete Output System
IC	Initial Condition
LRC	Langley Research Center
RCT	Requested Computer Time, also called Maximum Computer Time
RTPL	Real-Time Preprocessor Language
RTSS	Real-Time Simulation System
SKED	Real-Time Scheduler

CHAPTER I

INTRODUCTION

One of the major functions of the National Aeronautics and Space Administration's Langley Research Center Computer Complex is to provide computational support for real-time flight investigations. For purposes of efficiency, several real-time application programs operate concurrently in a single Control Data Corporation 6000 series computer. To perform "man in the loop" digital simulation requires that the computer operates as part of a closed loop, time critical system where precise problem solution rates must be guaranteed in order to maintain the integrity of the solution.

For ease of operation and programing, a real-time digital simulation supervisor (supervisor) was written to interface between a Fortran simulation program and the real-time system. It provides all the real-time input/output control, timing synchronization, communication, control and other system functions unique to real-time operations. To maintain the flexibility and computational speed needed in real-time simulation, supervisor was written as a series of interdependent subroutines that are loaded with each real-time program. This provides the programer the option of using only those features of the system that he needs. In addition to supervisor, there are two other subroutines that are used by most simulation programs. They provide integration and the capability of displaying, changing, and recording on a typewriter the value of problem variables.

The result of this approach is that a program must contain many subroutine calls. The bad feature of this is that much of the information is repeated in several calls.

The objective of this project was to develop a real-time digital preprocessor (preprocessor) that would minimize the programing involved in writing a real-time simulation by providing a meta language to Fortran. The preprocessor would parse this meta language and develop a real-time Fortran program. This would allow the programer to communicate to the preprocessor his real-time requirements in a unified manner and place the burden on the preprocessor to write all the subroutine calls necessary for real time.

A general description of the hardware and software necessary for real-time simulation will be presented. The features of the preprocessor will be explained and the characteristics of the meta language described. A sample program will be used to convey the saving that the preprocessor can provide the programer.

CHAPTER II

REAL-TIME COMPUTING SYSTEM

In the winter of 1965, Langley Research Center was operating an analog computing facility. Most of the problems that were being solved on these computers consisted of man in the loop simulation problems. The vehicle that was being simulated was usually some aircraft that could be represented by a set of first and second order, nonlinear, differential equations. By means of a simulated cockpit connected to the analog computers, a pilot would "fly" the mathematical model. The simulated flights would be recorded on eight channel strip chart recorders or two axis plotters.

During the same winter, it was decided that Langley Research Center could use digital computers to handle the same type problems. The influence of the analog computers is evident in the hardware and software of the resulting digital computer complex.

System Hardware

The real-time digital complex consists of four Control Data Corporation 6000 series computers and associated subsystems as shown in figure 1. When the system is in operation, several of these subsystems may be actively connected to one computer. The exception to this is that only one simulation subsystem or data recording subsystem may be attached to a computer. This means that a real-time simulation may be processing with batch, interactive CRT system and remote

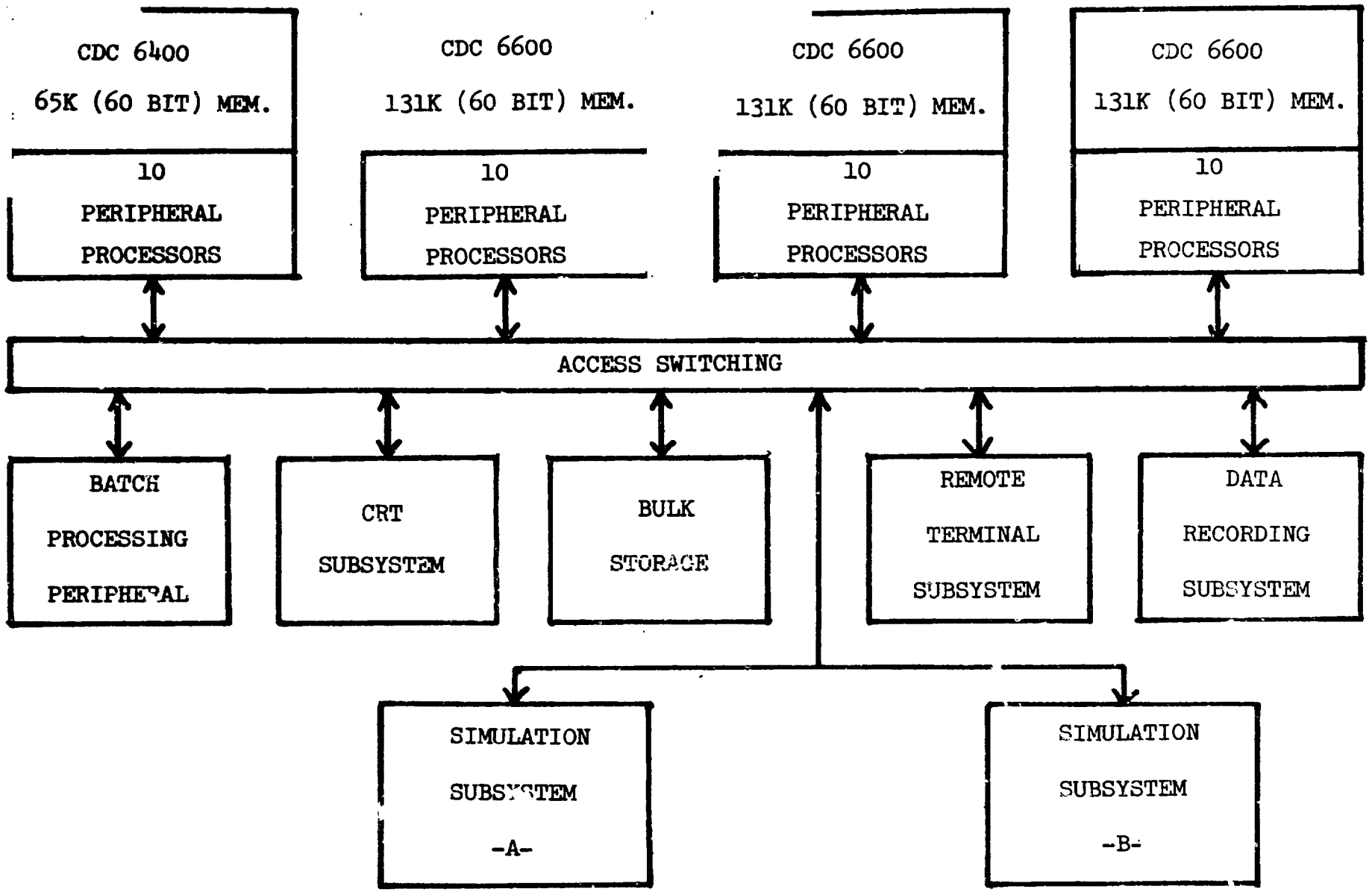


Figure 1.- LRC Digital Computer Complex.

terminal problems. It is in this environment that the computer system must guarantee the solutions of the simulations.

To perform digital simulation, two special Real-Time Simulation Subsystems (RTSS) are employed to perform inputting and outputting of analog and discrete signals and time synchronization. Figure 2 depicts one such system. Each RTSS consists of an Analog-to-Digital and Discrete Input System (ADDIS), a Digital-to-Analog and Discrete Output System (DADOS), a real-time clock and interval timer, and several control stations. The elements of DADOS are a set of digital-to-analog converters, a set of discrete output channels and a Digital-to-Analog Controller (DACON). ADDIS is the complement of DADOS with the addition of differential amplifiers, sample and hold amplifiers and analog multiplexers attached to the inputs of the analog-to-digital converters [1].

The program control station provides the person, who is operating a simulation program, the means to interact with the digital computer to obtain the results desired [2]. Each program control station as shown in figure 3 consists of a simulation console, a CRT control console, and a typewriter for displaying short messages. Both eight channel strip recorder and two axis recorders can be connected to the station for recording purposes.

Figure 4 illustrates the control panel of the program control console. Each panel contains a set of switches which are discrete inputs to the computer program. These discretes can be further subdivided into function sense switches, mode control switches and

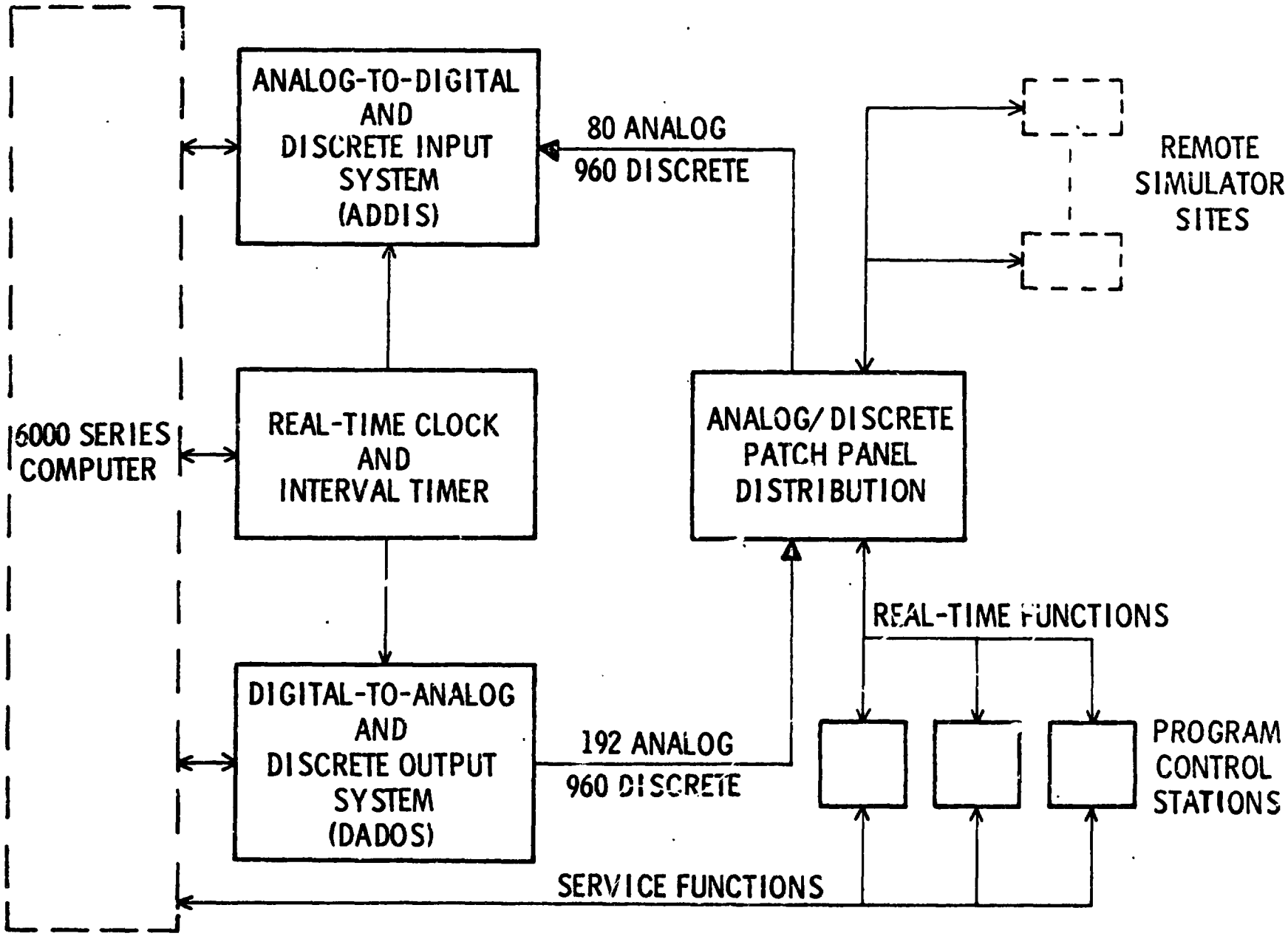


Figure 2.- Real-Time Simulation System.

Figure 3.- Program Control Station.

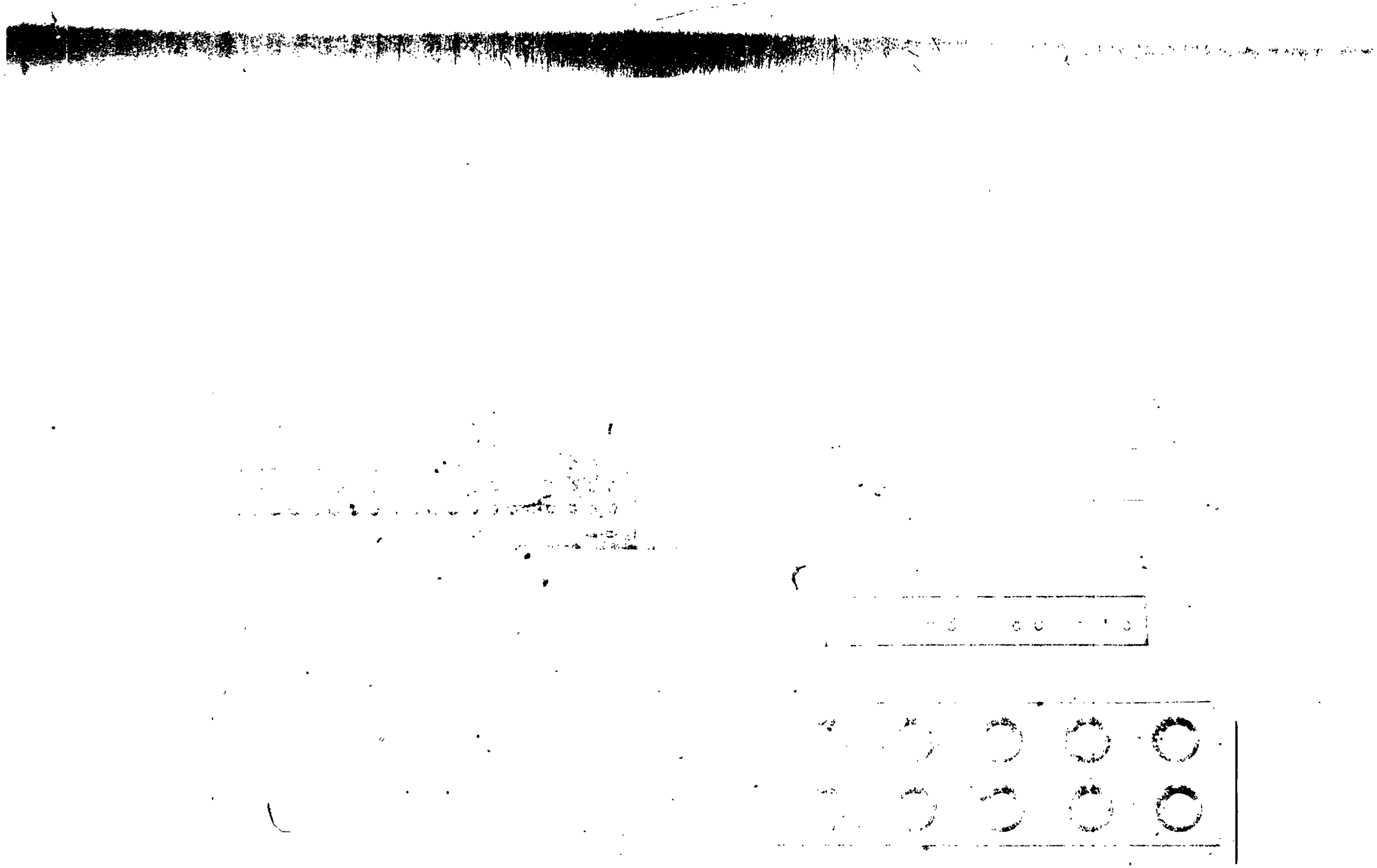


Figure 4.- Program Control Console.

data entry keys. The function sense switches can be used by the programer as logical variables to control and modify the real-time program. These discretes might be used to switch from one set of initial conditions to another or to switch the sign on one of the terms in the equations of motion. The data entry keys are used in conjunction with the decimal display unit. The keyboard is used to address problem variables, and their values will appear on the display unit. By continued manipulation of the keys and the use of one function on the mode switches, the value of the variable can be changed. The mode control switches include three modes, reset, hold, and operate and several functional operations that are unique to real-time simulation. Some of these functions are retrieval of real-time data and zeroing the DAC's and discrete outputs. The modes and functions will be explained in the next chapter. Also on the console, there are analog input devices in the lower right corner of the control panel. These handset potentiometers are connected to ADC input channels and can be used to continuously change the value of problem variables. This continuous changing of a value, sometimes referred to as twiddling, is most valuable in parameter studies where a value needs to be adjusted and the trend in the solution observed. On the left of control panel are lights that are driven by logical discrete output channels. The light can be programed to indicate the status of certain operations or that some event has occurred.

The CRT system is being used as an integral part of simulation. At present, it is being used to make on-line code modification of a

source program, display two axis plots of problem variables, display and modification of central memory, operating registers and the instruction counter and the advance of program's solution by one computer word at a time which may execute up to four computer instructions. The latter two uses of the CRT system are extremely useful in debugging a small portion of a program.

Real-Time Control

Figure 5 shows part of the internal computer organization which is employed to support digital simulations. The CDC 6000 series computers are multiprogramable. This feature is implemented by setting up control points. Each control point is assigned a starting address and a length. No memory fetches or stores will be executed outside this established area and if one is attempted, the program will be aborted. In figure 5, there are three real-time simulation programs running concurrently on control points two, three and four, while the rest of the computer is processing batch programs. Due to the increased demands on the computer's time while running real-time programs, the monitor in peripheral processor number zero was not sufficient. An additional monitor was written and resides in control point zero. Control point one contains scheduler (SKED) which is the software program that processes all the requests for timing and simulation resources. Two other resident programs are ADD and DAD. DAD resides in a dedicated peripheral processor that is connected to the Digital-to-Analog Controller (DACON) through a dedicated data channel. DAD is the software that implements the data flow. ADD, in addition to handling the

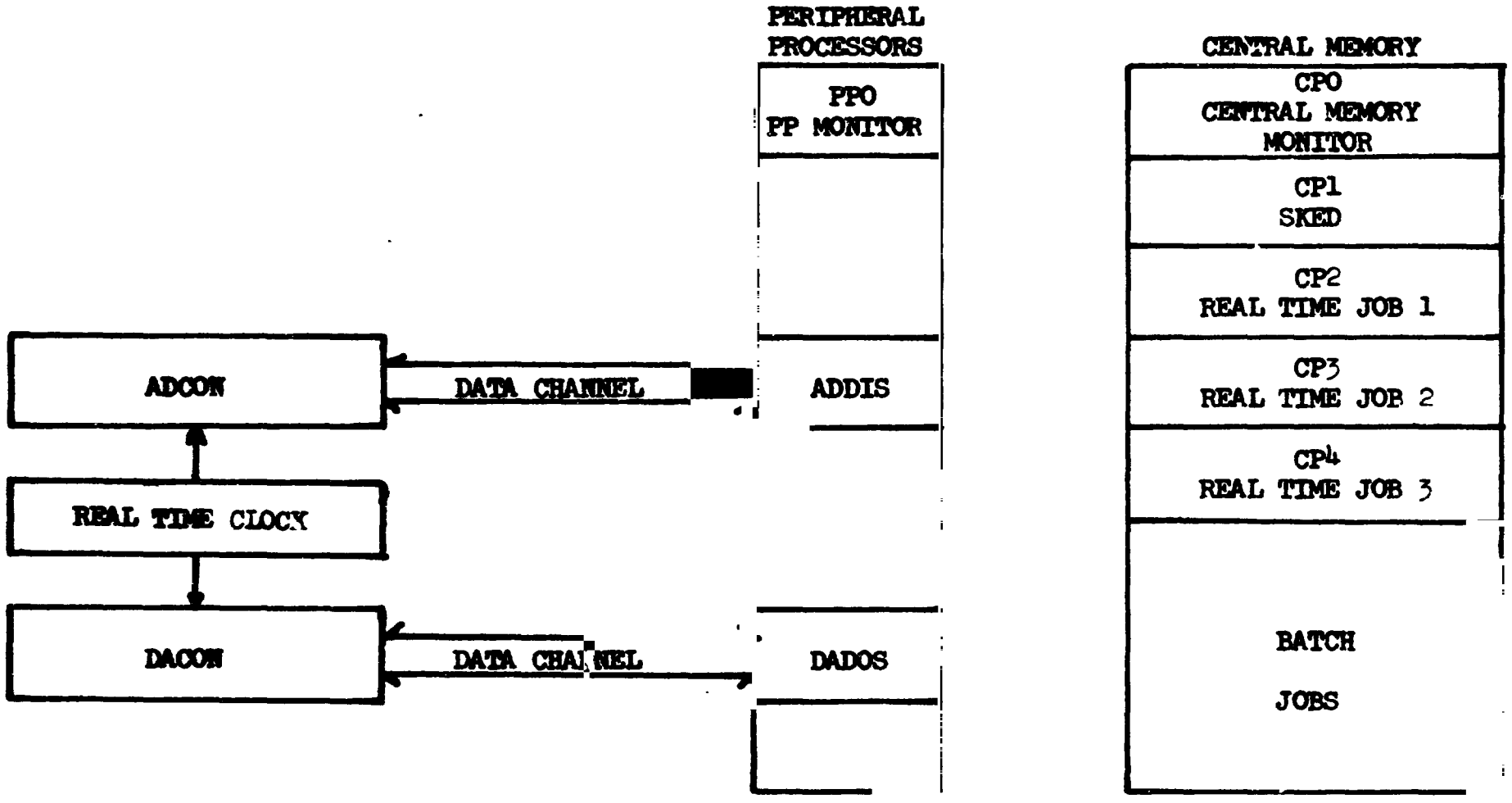


Figure 5.- Computer Organization with Real-Time Digital Simulation.

data flow for ADDIS, handles the timing information obtained by scheduler and implements the timing schedule.

The essence of real-time computation consists of a problem receiving inputs from the outside world at time t_1, integrating problem variables to t_2, and waiting until t_2. At t_2, the problem transmits outputs and receives inputs. This predicting ahead by integration and waiting, permits the system to synchronize inputs and outputs to the real world. The real-time clock and interval timer put out pulses that cause the input and output data to be transmitted at the precise time.

The Real-Time Scheduler

A real-time program contains control cards that reflect the program's needs for ADC's, DAC's, discretes, problem frame time, and maximum computer time per frame. The problem frame refers to the time between pulses of the interval timer. The maximum computer time refers to the amount of computer time required to complete one time step of the independent variable. After a real-time program has been compiled and is executing, a call to real-time digital simulation supervisor (supervisor) is made. After supervisor performs some necessary functions, the control is transferred to scheduler. The scheduler determines if the hardware requested, ADCO's, DAC's and discretes, is available. If it is available, the hardware is assigned. Scheduler develops a timing schedule based on the timing requirements of the programs currently in real time and the new program being processed. The scheduler develops this timing schedule by determining a common

frame time and simulating the central memory timing algorithm. If there is enough time to satisfy the maximum computer time requested by the new job, the new schedule will be implemented and the system will return control to supervisor and the new problem will be in real time.

The Real-Time Digital Simulation Supervisor

The real-time simulation supervisor, as stated in the introduction, is a set of subroutines that perform special functions for a real-time program. These functions are:

1. Real-time system initialization.
2. Real-time timing control.
3. Real-time central memory input/output control.
4. Control after time synchronization was lost.
5. Mode control.
6. Storage and retrieval of real-time data.
7. Print output control.
8. Error recovery and diagnostics.
9. Compatibility with batch jobs.

A further explanation of these features can be found in appendix A.

Integration Subroutine

Another aspect of real-time simulation that differs from the needs of batch processing is integration [4]. In batch processing, the integration algorithms contain additional features to drive the rounding error to the lower part of the computer word and usually provides some estimate of the truncation error. For the majority of simulations,

frame time and simulating the central memory timing algorithm. If there is enough time to satisfy the maximum computer time requested by the new job, the new schedule will be implemented and the system will return control to supervisor and the new problem will be in real time.

The Real-Time Digital Simulation Supervisor

The real-time simulation supervisor, as stated in the introduction, is a set of subroutines that perform special functions for a real-time program. These functions are:

1. Real-time system initialization.
2. Real-time timing control.
3. Real-time central memory input/output control.
4. Control after time synchronization was lost.
5. Mode control.
6. Storage and retrieval of real-time data.
7. Print output control.
8. Error recovery and diagnostics.
9. Compatibility with batch jobs.

A further explanation of these features can be found in appendix A.

Integration Subroutine

Another aspect of real-time simulation that differs from the needs of batch processing is integration [4]. In batch processing, the integration algorithms contain additional features to drive the rounding error to the lower part of the computer word and usually provides some estimate of the truncation error. For the majority of simulations,

the data is usually accurate to only a few places. Therefore, the four integration algorithm were written for real-time programs. These algorithms do not have an estimate of truncation error, only the basic integration formulas were programed. This approach was taken to minimize the computer time and it was left up to the programer to insure the accuracy of his solutions. The name of the subroutine is IGRATE1 and it will be discussed later.

CHAPTER III

CHARACTERISTICS OF THE PREPROCESSOR

Before the preprocessor can be discussed, the present method of writing a real-time program is presented. The new method of using the preprocessor is discussed followed by a description of the features of the preprocessor.

Present Method of Preparing a Real-Time Program

As Langley Research Center's real-time facility was becoming established, some variables received a Fortran name that became accepted and used by most people. An example of this is DAC's. DAC is the Fortran name of an array whose values are outputed to the real world through the digital-to-analog converters. As the real-time programs were written, the similarities among the jobs were noticed. To ease the programing burden, sets of Fortran cards were prepunched and made available to all real-time programers. These sets of cards contained many of the calls to supervisor that were necessary, but for some variables a Fortran name had to be assumed. The assumed Fortran variable name was usually a reasonable abbreviation of the complete name. This means that a programer can write a real-time program leaving out parts of the real-time structure and using the supplied cards to complete the program. Since these cards are in the programers deck, he can change any of the assumed Fortran names to a name more pleasing. For example, a programer may prefer "TIME" instead of "T" to represent time, the independent variable in the integration formula.

New Method

In lieu of obtaining a set of prepunched Fortran cards, a programer can write his real-time program without any real-time subroutine calls by using the real-time preprocessor's meta language, RTPL. The programer can specify his real-time requirements in a clear, uniform, and succinct manner. Some real-time programs may require the use of some additional real-time subroutine calls because of some special need.

Some of the real-time structure and subroutine calls that the preprocessor generates are absolutely necessary for a program to run in real time, and some of the other structures and calls are options of the programer. The philosophy of the preprocessor is to generate the necessary structures and calls everytime the preprocessor is used and to generate the other structures and calls only if certain constructs are present in the input program. The preprocessor makes use of default words and conditions. The preprocessor has a symbol table of variables that it needs to generate the Fortran code. Some of the constructs in RTPL determine the Fortran symbol to be used as certain variables. If these constructs are missing, then the preprocessor uses the default words. DAC is the Fortran default word for digital-to-analog converters. Certain conditions are defaulted if the corresponding constructs are missing. These default words and conditions should not be confused with items the preprocessor considers necessery. For example, integration is an option that the programer must define by a particular construct in RTPL. If an integration construct is present and if the independent variable and the magnitude of the step,

that the independent variable is incremented, are not defined by two o' er constructs, then the preprocessor will use the Fortran default variable name, "T", as the independent variable and "32/1024" as the step size.

Features of the Preprocessor

The preprocessor performs four tasks for the programer:

1. Parsing Fortran declaratives statements.
2. Interfacing with supervisor and other real-time subroutines.
3. Macro generation.
4. Error diagnostics.

Since RTPL is a meta language to Fortran and the preprocessor output is Fortran, the syntax of RTPL constructs is closely related to the syntax of Fortran. Therefore, the preprocessor needs a list of arrays and lists of variables that are typed real or integer before the preprocessor can properly parse the RTPL constructs. The preprocessor can obtain the information for the lists from parsing the declarative statements written in Fortran. The preprocessor does not need the programer to include any variable in a declarative statement that wouldn't be needed in a normal Fortran program.

Before the actual interface can be understood, it is important to have a clear understanding of what basic interface must be present to solve ordinary differential equations. To illustrate this, a problem is posed:

Problem – to solve the differential equation,

$$\ddot{y} + 2\xi w\dot{y} + w^2 y = F$$

where

$$\xi = 0.1, \quad w = 0.5, \quad F = 5 \sin 5t$$

The initial conditions are

$$t = 0., 0., 0; \quad y = -5., 0, 0; \quad \dot{y} = 0., -1., 0$$

Obtain time histories and printouts of F, y, and $\dot{y}$.

The elements of this problem can be categorized as initialize constants, set initial conditions (IC's), calculate derivatives, provide digital to analog output for recording, integration, save real-time data and modify IC's. In solving ordinary differential equations there are three phases or modes, setting the integrated variable equal to their IC's (Reset), integrating the variables (Operate), and maintaining current valves of integrated variables (Hold). These modes are present on current large analog computers and were implemented for LRC's digital simulations. In the reset mode, the integrated variables must be set equal to their IC's. Because the IC's must be changed for each run, a call to the display subroutine is useful. In the operate mode, the derivative equations must be executed, real-time data saved for later printout, integration and calculating values for DAC's. The hold mode must not include the setting of IC's, integration, or data storage. By adding an eight channel strip recorder to the DAC's, and initializing the constants, all the requirements of the problem are met.

In order to make the problems more controllable, LRC has included more code in each mode. There is one group of Fortran code that includes

derivative equations, DAC equations, and a call to display. Each mode passes through this code. For the above example, the reset mode needs to set IC's, and the operate mode needs to save real-time data and integrate in addition to the group of Fortran code. The hold mode can consist of only the group of code.

The interface with the real-time subroutines consists of the subroutine calls and any declarative or flow control that is necessary to sustain the call. The calling sequence to IGRATE1 contains no argument list. The input areas are established by a labeled common. Also, the integration algorithms are multipass. This means that the derivative equations must be executed with intermediate values of the dependent variables. The execution is managed by a flow control parameter. In discussing subroutines, a Fortran statement will be preceded and followed by the high set mark, '. Also, the actual parameter lists will be represented by 'LIST' regardless of the length of the list. This means that 'CALL ARCTAN (X, Y, Z)' would be represented by 'CALL ARCTAN (LIST)'.

Interface With Supervisor

The preprocessor is capable of generating 11 subroutine calls to supervisor and two common statements. These 13 statements can be classified into four groups: Block control, recorded data management, communication, and real-time initialization.

Block Control.- 'CALL RESET (LIST)'
'CALL HOLD (LIST)'
'CALL OPERATE (LIST)'
'CALL OPTION (LIST)'
'CALL LOSTIME (LIST)'

A real-time program can be subdivided into several blocks acccrding to different functions. Calculation of IC's could be in one block while evaluations of derivatives could be in another block. Each block is preceded by a CONTINJE card and followed by a RETURN, and both of these statements must have a statement number. Then by passing these statement numbers to the supervisor, the supervisor can write a jump to itself in the return location and jump to the location of the continue. The block is executed and a jump to the supervisor is made. This technique is used to thread the blocks according to the different modes and functions (the threading is similar to the way lists are threaded). A call to supervisor identifies a mode or functior and the pairs of statement numbers for each block. These statement numbers indicate which blocks must be threaded to form the complete code structure for the mode or functions. The modes have already been explained. Print establishes blocks that are used in printing out data that was recorded while the problem was in operate. The supervisor takes the stored data and returns it to central memory, a set of data at a time. After a set of data has been restored, supervisor transfers control to one of the print blocks and in this block the programer has a write statement with a format specified. After the write, the program transfers control to the supervisor. This restore and print is alternated until all the data is exhausted. Option and lostime are functions that have only a single block associated. The option block is executed when the option made control switch is depressed and the lostime block is executed when a program exceeds the maximum

compute time specified on the control cards. The contents of these two blocks are left up to the programer.

Recorded Data Management.- 'CALL RTROUTE (LIST)'
'CALL READOUT (LIST)'
'CALL RECORD (LIST)'

Fortran write statements with a format specified use up an excessive amount of computer time when compared to other Fortran statements. For real-time recording an alternate method was implemented that saves the data during real-time computation and writes the saved values with a format specification in a nonreal-time computational mode. While in the operate mode, the valves of the variables to be recorded are copied from their central memory locations into a buffer. The contents of the buffer is then copied to a special disk file. In the print block and with the aid of supervisor, the contents of the disk file is restored to central memory and the program performs writes to a real-time file. The call to RTROUTE informs the supervisor of the symbolic Fortran name for the real-time file. The argument list of READOUT contains the names of all the variables that shall be recorded. The position of Record is used to indicate where in the operate mode the saving of data is to be implemented and argument of the call indicates the frequency at which the data is to be saved. If the argument were 32 and if the computer were cycling through the operate mode 32 times per second, then the first, 33rd, 65th, etc., passes would be saved and these points would occur every second.

Communication.- 'CALL INOUT (LIST)'
'COMMON/INOUT/LIST'
'COMMON/MASKS/LIST'

These three Fortran statements provide communication between the real-time program and supervisor. Inout informs supervisor of the addresses of the symbolic names and numbers of ADC's and DAC's that the real-time program needs to be converted. COMMON INOUT is used by supervisor to locate the central memory locations for discrete inputs and outputs. COMMON MASKS establishes the central memory locations for some masks. Supervisor creates the mask for the real-time program because it uses the discrete words in a packed form. This packed form means that each bit represents the status of a discrete channel, and one discrete word represents 60 channels. The programer can obtain the status of a discrete input or set the status of a discrete output by using the logical operators, "OR" and "AND" with a mask on a discrete word.

Real-Time Initialization.- 'CALL READY'. READY is an entry point to a subroutine that initializes the real-time system. This call causes the supervisor to activate the scheduler. When control is returned to the real-time program the control point is operating in real-time.

Interface With Display

'CALL DATABLX (LIST)'
'CALL XDSPLAY (LIST)'
'CALL DSPLAY'
'CALL TYPVAR'

Display is a subroutine with several different entry points that permits the value of problem variables to be displayed on the display unit, changed by the ch nge switch (one of the mode control switches)

and recorded by the typewriter. Each entry point of the subroutine is called by the name of the entry point like the entry point was a separate subroutine. The DATABLX call indicates to display the symbolic name and length of arrays that will be displayed. The XDSPLAY call indicates where the status of the data entry keys can be located. DSPLAY is the name of the entry point that does the actual displaying and changing of variables. The call to the TYPVAR entry point causes messages about the present and past values of the displayed variable to be typed.

Interface with IGRATE1

```
'COMMON/INTCOMM/LIST'
'COMMON/INTINTR/LIST'
'CALL IGRATE1'
```

IGRATE1 is a subroutine that contains four multipass integration algorithms. The labeled common block INTCOMM, contains the names of all the variables that are necessary for integration. The necessary variables are independent variable, dependent variables, derivatives, time step, integration flow parameter and the number of variables to be integrated. The labeled common block, INTINTR, provides storage for all intermediate valves of the dependent variables. For some integration algorithms, the derivatives are calculated in a separate subroutine that the integration algorithm can call as often as necessary. For real-time simulation the derivative equations are embedded in the main program. The flow control parameter whose default Fortran name is 'INT' is used to iterate through the derivative equations and a call to IGRATE1 until the algorithm is complete. A flow control flag whose default name is

'FLAG', is used to recalculate the value of the derivatives and DAC's prior to the DAC's value being output to the real world. The reason that the derivatives and DAC's must be recalculated is that the integration is predicting to the time when the DAC's are output. Therefore, these equations must be recalculated with the integrated values corresponding to the upcoming output time.

Macro Generator

Since Fortran is not a very convenient language for macros, the effectiveness of the macro generator in the preprocessor is curtailed. The macros are not recursive. No statement numbers are permitted and all formal parameters must be preceded and followed by two "$"'s. More information is contained in the next chapter.

Error Diagnostics

When a program is being debugged, one of the most important aspects is the machine to man communication of errors. Some of the typical problems with debugging facilities are error messages that aren't clear or precise in identifying the problem. Most everybody has encountered a syntax error in a Fortran compiler. Some of these errors are very difficult to detect. Another problem in some compilers is that there appears to be different levels of compiling, and the later levels are not examined until earlier levels are correct. This means that several recompilations must be made to debug a program. The structure of do loops are not examined by the Fortran compiler at LRC until after all syntax errors are removed.

If a system is hard to debug, then a lot of man-hours will be wasted due to poor design. The preprocessor contains many error diagnostics, and every effort has been made to make them precise and clear. Some of these diagnostics are in appendix B.

CHAPTER IV

REAL-TIME PREPROCESSOR LANGUAGE

RTPL is a meta language to Fortran; that is, the preprocessor translates a RTPL statement into tables and files and converts this information into Fortran statements. To describe the language, three aspects of it will be presented: syntax, semantics and usage. Since these three aspects are dependent on each other, the order in which they are presented is arbitrary. It would be well for the reader to review the first aspect after all three aspects have been covered.

SYNTAX

With the growth of so many programing languages, the uses and advantages of a well defined syntax is generally accepted by people in the computing field. Since Fortran was designed and implemented before the importance of syntax was recognized, it and all its later versions do not have a well defined syntax. The most difficult feature of Fortran that makes it hard to specify a syntax is the quantization that is everywhere present in Fortran. This quantization is usually due to implementation and not to any specific concept in the Fortran specifications. On the IBM 7094 a Fortran variable has a maximum length of six characters which filled the word. On the CDC 6000 series computers a Fortran variable can have up to seven characters and the rest of the computer word is used for systems information. In both of these examples, the quantization was due to implementation.

There has been some notable work in defining the quantization of Fortran. "Report on the Algorithmic Language Fortran II" by Rabinowitz expressed the syntax of Fortran II in a modified Bacus Naur Form [5]. Rabinowitz added the meta operator $F_1[m,n]$ which meant that the syntactical unit to the left of F_1 must appear at least m times but not more than n times. Meta Language and Syntax Specification" by Walter H. Burkhardt uses a quantization approach and adds level numbers [6]. Burkhardt uses the two meta operators $ and ↑ to denote the minimum and maximum number of times the syntactical unit on the right can be repeated respectively. The level numbers are more useful in defining the syntax for syntax directed compilers than syntactical names. Another report, "A Syntax-Directed Fortran Statement Checker" by Susan S. Hoffberg and Max Goldstein, uses only the meta operator ↑ [7]. The meta operator indicates the maximum number of times that the syntactical unit to the left of the operator can be repeated.

This report on RTPL uses the BNF meta linguistic symbols with the additional operator ↑n. Also, level numbers have been assigned to each syntactical unit in the left portion of the definition. The level numbers make the syntax more usable by people because the definitions of syntactical units are placed in ascending order for easy reference by level numbers. The meta-linguistic symbols and meanings are:

Symbol	Meaning
:: =	is defined to be
|	or

(no symbol)	concatenation
↑n.	zero to n repetitions of (the syntactical unit on the right)
< >	defines a syntactical unit
< m,o >	level number m and syntactical name 0
b	null syntactical unit

The definition of a defaulted integer variable is presented below:

< 1, nonzero digit > :: = 1|2|3|4|5|6|7|8|9

< 2, digit > :: = 0|< 1, nonzero digit >

< 3, real letter > :: = A|B|C|D|E|F|G|H|P|Q|R|S|T|U|V|W|X|Y|Z

< 4, integer letter > :: = I|J|K|L|M|N

< 5, non 0 letter > :: = < 3,real letter >|< 4, integer letter >

< 6, letter > :: = < 5, non 0 letter >|0

< 7, alphanumeric character > :: = < 6, letter >|< 2, digit >

< 8, default integer variable > :: = < 4, integer letter >↑6.

< 7, alphanumeric character >

The first statement would read, "A nonzero digit with a level number 1 is defined to be a 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9." The syntax of Fortran declarative statements is presented in appendix D. If BNF is not familiar, reference 8 contains a very good presentation of BNF to define ALGOL 60. On pages 4 to 6 in reference 7 there is a very good discussion of syntactical definitions. The syntactical defintion of a default variable would be, "A default integer variable with a level number 8 is defined to be an integer letter, level

number 4, with zero to six repetition of an alphanumeric character, level number 7."

Semantics

There are three types of RTPL statements that the preprocessor will parse and store the information in tables. They are block, nonblock, and macro statements. The nonblock and macro statements will be represented by an operator followed by one or more operands: OPERATOR (OPERAND1,OPERAND2). The block statements will be represented by BNF.

Block Statements

'BEGIN < block name 2 >'
'DISCONTINUE < block name 1 >'
'CONTINUE < block name 1 >'
'END < block name 2 >'
where < block name 1 > :: = RESET|HOLD|OPERATE|PRINT
< block name 2 > :: = < block name 1 >|DECLARATIVE|INITIALIZE|OPTION

LOSTIME

A RTPL program consists of a Program name card, a series of blocks and an End card. The Program name and End cards are valid Fortran cards. Each block is composed of a beginning block indicator, a string of Fortran code or RTPL code or both, and an ending block indicator. Each block can be classified into one of three classes. Class 1 contains uniblock types, class 2 contains multiblock types and class 3 contains a single, two block type. Figure 6 depicts one type of block from each class. Each block indicating statement in RTPL consists of a beginning or ending word followed by a word that names the type of block. For example, 'BEGIN OPTION' indicates that the block type is OPTION and it

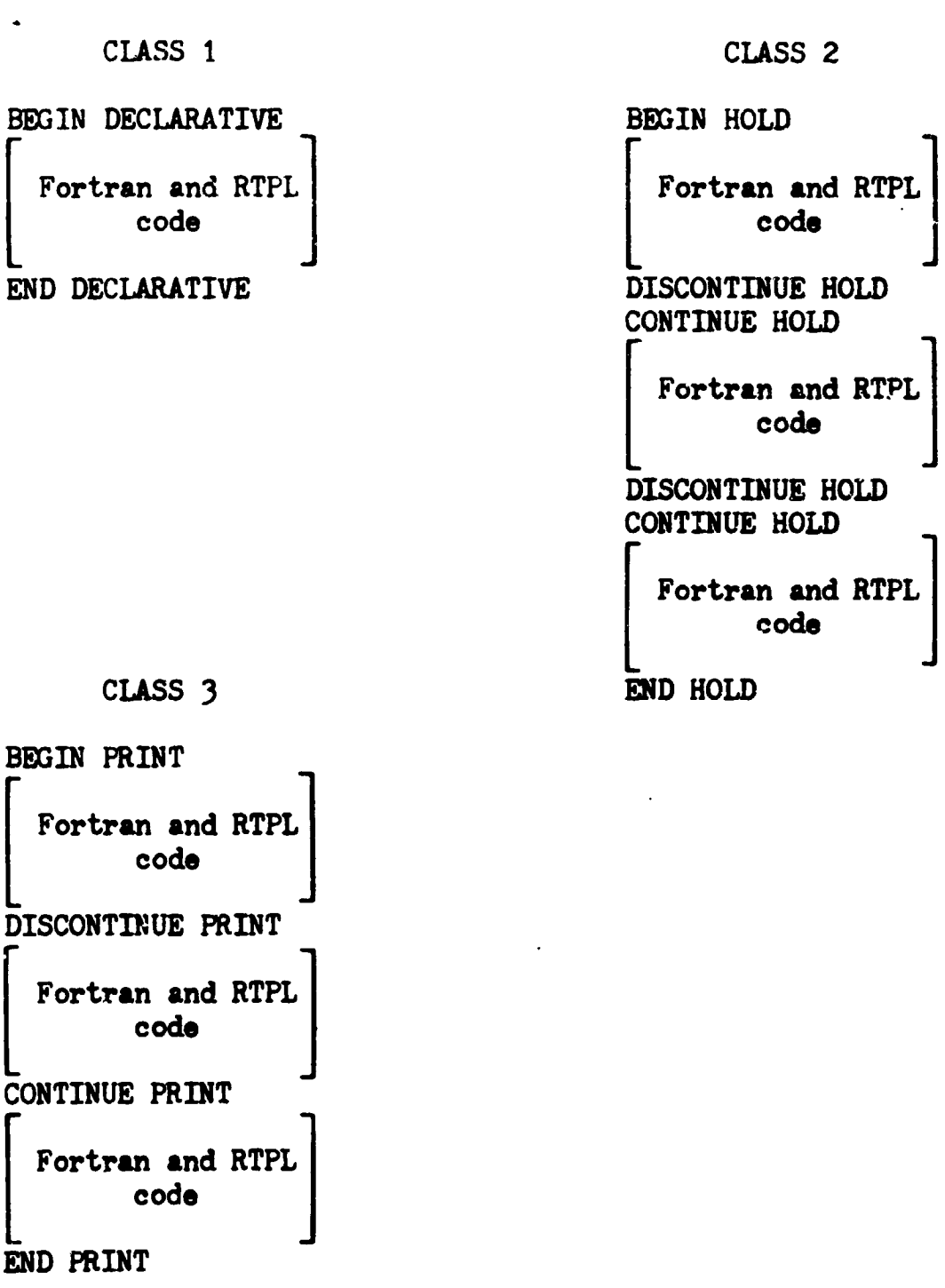


Figure 6.- Block Structure

is a beginning block indicator. For class 1 blocks, only the words BEGIN and END can be used as beginning and ending words. For class 2 blocks, the first block must use BEGIN and DISCONTINUE, the last block must use CONTINUE, and the intervening blocks, if any, must use CONTINUE and DISCONTINUE as beginning and ending block indicators. If there is only one block of class 2, it will use the block indicators for class 1. Class 3 blocks, which there are only two, are like the first and last block of class 2. Another difference is that the two class 3 blocks can have Fortran code between them. Class 1 consists of the following types: DECLARATIVE, INTIALIZE, OPTION, LOSTIME. RESET, HOLD, OPERATE comprise the class 2 blocks. Print is the only member of class 3. The meaning and usage of these blocks will be explained later.

Nonblock Statements

There are five types of nonblock statements. They are: integration, display, recording, conversion equipment and communication. Each one of these types will be discussed in detail. In the representation of the RTPL statement the default words are inserted for the sake of clarity.

Integration

'TIME INTERVAL (32)'

The symbol, '32', is an integer constant that defines the number of 1/1024th of a second that the independent variable will be stepped

during each integration step. To run in real time this integer must agree with the problem frame time specified on the control cards. A problem can run two to one fast by setting this operand equal to twice the number on the control cards.

'SET INTEGRATION (T,H,INT,NEQ,ISCHEME,DERINT)'

The above Fortran symbols are the names of the variables in the list of the common labeled 'INTCOMM'. The resulting Fortran statement would be: 'COMMON/INTCOMM/T,H,INT,NEQ,ISCHEME,DERINT(2,10)', if there were 10 variables to be integrated. The symbol, 'T', is the name for the independent variable, time. The symbol, 'H', is the name of the step size of the integration. 'INT' is the symbolic name of the flow control parameter used to iterate the derivative equation for multi-pass integration. The symbol 'NEQ' is the name of the variable that contains the number of variables to be integrated. 'ISCHEME' is the symbolic name of the variable that indicates which integration algorithm will be used. The symbol 'DERINT' is the name of a two by NEQ array that contains the values of the integrated variables and their derivatives.

'INTEGRATION BUFFER (INTERN)'

This statement establishes a symbolic name for a five by NEQ array which is used to store intermediate values of the independent variables during a time step by the integration algorithm. If there were 10 variables to be integrated the resulting Fortran statement would be: 'COMMON/INTINTR/INTERN(5,10)'. The storage area is passed to the integration algorithm by means of the labeled common.

'SCHEME (1)'

The symbol, '1', is an integer constant or variable that indicates which integration algorithm will be used.

'INTEGRATE (OP1,OP2,OP3)'

This statement defines the symbol name for the dependent variable, its derivative and its initial condition. OPERAND1 is the variable name, OPERAND2 is the derivative name and OPERAND3 is the name or value of the initial condition.

Display

'CHANGE NAME (TABLE)'
'CHANGE INTEGER NAME (INTEG)'
'CHANGE LOGIC NAME (LOGIC)'

The symbols, TABLE, INTEG, LOGIC are the names of three arrays whose value can be displayed and changed by the subroutine DSPLAY. Values in TABLE are displayed in a floating point form. Values in INTEG are displayed in a fixed point form. A special floating point type code is used for displaying LOGIC variable.

'CHANGE (OP1,OP2,...,OP199)'
'CHANGE INTEGER (OP1,OP2,...,OP99)'
'CHANGE LOGIC (OP1,OP2,...,OP99)'

Variables that are to be displayed can be equivalenced to an element in TABLE, INTEG or LOGIC depending on the type of variable. If the variables are equivalenced, then the Fortran symbol name can be used in the real-time program and the position in the arrays can be used to display and change the Fortran variable. The operands in the

above statement are the Fortran symbol names that are to be equivalenced to the elements in the three arrays.

'SCANNER (OP)'

The symbol, 'OP', is an integer constant or variable name which determines how often the decimal display unit is incremented when the Scan switch on the control panel is depressed. The operand indicates how many problem frames must pass before the element in the display is incremented. If the computer was running at 32 problem frames per second and 'OP' was equal to 32, then every second the next element in the display arrays would be displayed.

'DISPLAY VARIABLES (VARCHNG,ITYPE,IVARBUF,FS14,14,FS15,15,FS16,16,ENABLE)'

All of the operands except the fifth, seventh and ninth are Fortran variable names. The fifth, seventh and ninth operands are small integer constants. 'VARCHNG' is a symbolic representation of a logical flag that is set true by DSPLAY whenever the value of a variable is changed. This flag is used to call TYPVAR which is a subroutine that types out the past and present value of the variable that was changed. ITYPE is the name of a variable internal to DSPLAY. IVARBUF is the symbolic representation of the array. This five element array contains information about the present and past values of the variable being changed. 'FS14', 'FS15', and 'FS17' are the symbolic representations of three functions associated with the display unit. '14', '15', and '16' are small integer constants that indicate which function switch will activate the three functions respectively. There are two automatic type functions other than the automatic type out of variables as they are

changed. The first function causes printout of every variable as it is displayed. The other function works in conjunction with the scan switch. Every time a new variable is displayed in the scan mode, its value will be recorded by the typewriter. This feature permits a quick means of recording a group of variables that ordinarily would not need to be recorded. The two features are associated with logical variables FS14 and FS15. FS16 is associated with two modes of displaying variables. If FS16 is true, then the display unit displays variables stored in arrays. If FS16 is false, then any variable in the real-time program can be displayed. This form of addressing takes several steps to set up the address of the variable to be displayed or changed.

Recording

'RECORD (OP1,OP2,...,OP64)'

This statement contains the Fortran symbols of the variables and array elements that will be recorded while the real-time program is in the Operate mode.

'REAL TIME FILE (MF)'

'MF' is the Fortran symbol name of the REAL TIME FILE onto which the program will write data in the Print blocks.

'RECORDING FREQUENCY (32)'

'32' is an integer constant that determines how often the real-time variables will be stored. The RECORD entry point of supervisor is called every problem frame but data will be saved on every '32'nd problem frame, that is, the first, 33rd, 65th passes.

Conversion Equipment

'ADC NAME (ADC)'
'DAC NAME (DAC)'

'ADC' and 'DAC' are the Fortran symbol names for analog-to-digital and digital-to-analog converters.

'ADC (OP1,OP2,OP3)'
'DAC (OP1,OP2,OP3)'

These statements are used to scale ADC's and DAC's. The first operand is the variable name, For 'ADC', OPERAND2 is a signed bias and OPERAND3 is a scale factor. For 'DAC', the meaning of the second and third operands are reversed. The following two examples of a RTPL statement and the resulting Fortran statement are:

```
RTPL ADC (ALPHA,-100.,.01)
FORTRAN ALPHA = (ADC(1)-100.)*.01
RTPL DAC (BETA,SCALE,+BIAS)
FORTRAN DAC(1) = (BETA*SCALE)+BIAS.
```

'ADC SKIP (OP)'
'DAC SKIP (OP)'

The operand is the integer number of ADC's or DAC's that are to be skipped. This is necessary because the ADC and DAC statements do not have an index associated with the statements and the preprocessor assigns a number each time an ADC or DAC statement is encountered. The skip statement provides a means to not use certain ADC's and DAC's.

Communication

'DISCRETES AND MODES (REMOTE,IDIS,ODIS,TYPIO,TYPTPE,FLAG)'

This statement defines six Fortran variable names for use in the real-time program. The first three operands pertain to the discretes and mode control. 'IDIS' and 'ODIS' are the discrete input and output arrays. The supervisor normally obtains the status of the mode control switches by decoding IDIS(1). By calling subroutine MODEREM, supervisor will decode REMOTE instead of IDIS(1). This feature provides mode control from a source other than the mode control switches. By calling MODENOR, the supervisor will return to using IDIS(1). 'TYPIO' is a four word buffer that supervisor uses to store messages for the typewriter. 'TYPTPE' is the typewriter's unit number that was requested to be assigned to the real-time program. 'FLAG' is the logical flow parameter that is used to recompute the derivative equations and DAC equations one time after the integration step is complete.

Macro Statements

'BEGIN MACRO < Fortran variable name > (OP1,OP2,...,OP100)'
'END MACRO < Fortran variable name >'
'CALL MACRO < Fortran variable name > (OP1,OP2,...,OP100)'

The BEGIN statement defines the name of a macro and its formal parameters and commences the definition of the named macro. The END statements terminate the definition of the macro. The CALL MACRO statement causes the macro named to be fetched from the macro definition file and to be expanded with actual parameters replacing formal

parameters. A macro can have up to 100 arguments or as little as none. The macro body will be discussed in the next section.

Usage

This section will cover the pragmatic aspects of the language.

Card Format

The card format consists of an R in column 1, columns 7 to 72 are free field and columns 73 to 80 can be used for sequence tags. The preprocessor never tries to decode columns 2 to 5 and 73 to 80. The free field from columns 7 to 72 means that an RTPL statement can commence anywhere to the left of column 6 and to the right of column 73 with as many blanks as desired by the programer. If one card is not sufficient for the RTPL statement, the statement can use up to 19 continuation cards. A continuation card consists of an R in column 1 and a nonblank character in column 6. The same free field applies to continuation cards.

Program Structure

An RTPL program must contain a Program name card, a declarative block, an initialize block, at least one block for each of the modes and an End card. In addition to the above required structure, a program can have up to seven blocks of each mode, two Print blocks, an OPTION block and a LOSTIME block. The declarative block must contain any Fortran declaratives that are necessary followed by nonblock or macro definition statements. Any statements other than a comment will be detected by the

preprocessor as an error. A set of macro definition statements consists of a BEGIN MACRO, a Fortran macro body and an END MACRO. All other blocks consist of valid Fortran statements, and macro calls, and other block indicators. Figure 7 contains a symbolic real-time program. It should be noted that one block of Fortran code is in blocks of all three modes. This is a valid structure.

As a general rule there is no particular order for nonblock RTPL statements. There are some statements that establish arrays and their position with respect to all other statements of the same type is important. CHANGE, CHANGE INTEGER, CHANGE LOGIC and RECORD statements are examples of this. Also ADC, DAC, ADC SKIP and DAC SKIP statements determine their position in the ADC array and DAC array. Integrate also establishes a specific order for the DERINT array but the order usually is of no consequence to a programer.

The Print blocks are unique because Fortran code is permitted between the two blocks. For convenience, the blocks will be referred to as sections. The first section is the code in the first block. This section is executed once every time the Print switch is turned on. The second section is the code between the two blocks. This section is iterated until all the stored data is exhausted. The third section is the code in the last block and it is also executed once. The first section should contain a write to the REAL TIME FILE which identifies the data in the form of a header. Section 2 should contain a write to the REAL TIME FILE of all the data saved. Section 3 can be used to perform any postprint processing. If the programer prefers to

```
PROGRAM THESIS(INPUT,OUTPUT)
BEGIN DECLARATIVE
[ Fortran and RTPL code ]
END DECLARATIVE
BEGIN INITIALIZE
[ Fortran and RTPL code ]
END INITIALIZE
BEGIN RESET
[ Fortran and RTPL code ]
BEGIN HOLD
[ Fortran and RTPL code ]
BEGIN OPERATE
[ Fortran and RTPL code ]
END RESET
END HOLD
DISCONTINUE OPERATE
CONTINUE OPERATE
[ Fortran and RTPL code ]
END OPERATE
END
```

Figure 7.- Symbolic Program

have his header appearing with the data, then section 3 may be left empty and the writes in section 2 can contain the header and data information.

As stated before, the macro definition statements consist of a BEGIN MACRO name, Fortran statements, and an END MACRO name. The Fortran statements have all the formal parameters surrounded by a pair of $'s. The code cannot contain any statement numbers.

Appendix E contains a sample input program that was preprocessed. The output program is the contents that was produced in Fine1. It should be noted that the preprocessor used statement numbers in the 90,000 range for the block statement numbers. This range of statement numbers are prohibited from use by the programer.

All errors that the preprocessor detected will be flagged and a diagnostic will appear in the error directory. If there are programing errors, the input file will be printed with all statements with errors flagged. Also, the error directory will be printed. The error directory will contain a diagnostic for each error encountered.

CHAPTER V

IMPLEMENTATION OF THE PREPROCESSOR

The preprocessor has four functional units; they are: a Fortran parser, a RTPL parser, a Fortran generator, and a macro generator. These functional units are not separate algorithms but are intertwined throughout the preprocessor's code. The Fortran parser decodes Fortran declaratives and stores the symbols of arrays and real and integer variables that were typed for use by the RTPL parser. The RTPL parser decodes the RTPL statements and stores the information in tables for use by the Fortran generator. The Fortran generator writes Fortran code on Temp, a temporary file, as a result of the information stored by the RTPL parser. The macro generator has two phases of operation, macro definition and macro expansion. The macro definition phase consists of writing the macro body and a macro text consisting of the macro name and a list of formal parameters on the macro definition file. The macro expansion consists of locating the macro in the definition file, replacing formal parameters with actual parameters, adjusting the Fortran code, and writing the expanded macro on Temp.

The preprocessor passes through the input file only one time. The preprocessor reads a card and inspects it one character at a time. Once the card image is parsed it will be discarded or written on Temp. After the Fortran End card is detected, the preprocessor copies the contents of the Temp file to Final, inserting the necessary supervisor calls to establish blocks for the modes and certain functionals. Any remaining

subroutine will be copied to Final. After the preprocessor is finished, Final can be compiled and executed as a real-time program. Because continuation of RTPL statements is permitted, the preprocessor, if the syntax is not complete on one input card, will read the next card. The preprocessor tests the ne card to determine if it is a valid continuation. If it is a continuation card, the preprocessor will continue to parse the syntax. If it isn't a continuation or a comment card, then the previous card has an invalid syntax. The previous card will be flagged as invalid and a detailed error message will be entered in the Error Directory. If the card was a comment, the next card will be read and checked as before.

The first card of an RTPL program must be a Fortran Program name card. If it isn't, the preprocessor will write a diagnostic and try to decode the first card as any other card. The input card is tested to find out if it is an RTPL statement, a Fortran comment or another Fortran statement. Comments are copied from input to Temp. There are several types of Fortran statements. If the macro definition switch is true, the input information is written in the macro definition file. If the declarative block switch is true (indicating a declarative block is being processed) the string of code will be parsed. If neither switch is true, the card is tested for an End card. If not, the card image would be written in Temp. If it is, the end processing mentioned earlier would be executed. If the card was a RTPL statement, the preprocessor tries to parse the string as a block statement. If it isn't, it is parsed as a nonblock statement. If the nonblock parsing fails, the

statement is flagged as an error and a diagnostic is written. If the nonblock parsing succeeded, a subroutine is called to parse the operands. If the block parsing succeeds, then statement numbers are stored and the proper Fortran CONTINUE or RETURN statement is written. The macro statements are handled as a set of the block statements. If any MACRO BEGINS or ENDS are detected, the macro definition switch is switched true or false respectively. If a CALL MACRO is detected, a subroutine is used to handle the expansion.

Appendix F contains a list of all the subroutines that are used by the main program of the preprocessor.

CHAPTER VI

CONCLUSION

The objective of this research was to design and implement a computer program that would accept a Fortran program with a few non-fortran statements as input, decode the nonfortran statements, and develop a real-time program. Through the nonfortran statements, the programer would explain his real-time requirements in as simple form as possible. There was to be no infringement on the programer's freedom in programing his problem, yet he should be freed as much as possible from routine coding and bookkeeping information.

These objectives were adequately met by the preprocessor. The operator form of statements is easy to understand and to use. The supervisor required a block structure for the modes and associated functions. This block structure was easily extended for declaratives and initialization processes and provided good definition of where to insert the Fortran code generated by the preprocessor. The macro capability, defaulted names of variables and diagnostics will save many hours of programing, coding and debugging of a problem. It also means that many system changes can be implemented through the preprocessor and the programer's deck will not be disturbed.

Since this preprocessor will be used by many people, from time to time changes will be made to the program to enhance its features and to meet new needs. At the present, plans are being made to extend the macro capability and to provide a print package similar to the ones

found in continuous system simulation languages. For the print package, the variable name and some Hollerith string would be passed tc the preprocessor. The Hollerith string would be used as a header and the variable name would be used in a write statement with a fixed format. This continual development will keep the preprocessor current of the needs of the computing facility.

APPENDIX A

DESCRIPTION OF THE REAL-TIME DIGITAL SIMULATION SUPERVISOR

The following material is taken directly from reference 3, pages 12 to 17.

The supervisor is a set of subroutines integral to each simulation job. The supervisor performs all real-time input/output control, timing synchronization, communication and control, and other related functions that are system dependent. This allows the simulation program to be coded in Fortran with little regard to the computer interface with the real-time world.

The real-time digital simulation supervisor must perform the following functions:

A. Real-Time System Initialization

B. Real-Time Timing Control

C. Real-Time Central Memory Input/Output Control

D. Control After Lost Time Synchronization Interrupt

E. Mode Control

F. Real-Time Data Storage and Retrieval

G. Print Output Control

H. Error Recovery and Diagnostics

I. Batch Job Compatibility

A. Real-Tim System Initialization

When a real-time job enters the computer system, the only special characteristic that it has is the priority. Once the job begins to execute, it runs like a high priority batch job. Through a series of initializing calls, the simulation applications job communicates certain real-time data that is required for real-time operations. At this point, the supervisor must communicate to the operating system information for execution of the real-time portions of the job.

The supervisor must communicate to SKED the addresses where the ADC, DAC, discrete, real-time clock, and other real-time information for this job reside. The supervisor must construct an interrupt table to the real-time monitor. In addition, the supervisor must set up internal flow control, data areas, and perform other functions necsary to prepare for real-time operation.

B. Real-Time Timing Control

A real-time simulation job may execute in one of two states. It may execute in real time, where strict time synchronization is held and real-time responses are calculated. It may also execute in non-real time where time synchronization is not maintained and the job executes like any high priority batch job. A real-time simulation job may change readily from real time to nonreal time or vice versa. The supervisor must perform the necessary monitor functions to perform the transition described. The supervisor must also perform the necessary system functions to guarantee time synchronization while the job is

operating in real time. The supervisor also computes the maximum CPU time per frame for programer information.

C. Real-Time Central Memory Input/Output

The supervisor controls the transmission and distribution of input/output from the RTSS. ADC's and DAC's are packed four channels per word and the supervisor provides the pack/unpack capabilities so that these quantities appear in normal floating point numbers in the Fortran program. Discretes are packed 60 per word and may be unpacked into normal Fortran logical variables if that mode of operation is selected.

D. Control After Lost Time Synchronization Interrupt

A simulation job requests of the system two time increments that are pertinent to real-time execution. The first increment requested is frame time--this is the time between samples and defines the iteration rate. The second is requested compute time. Since more than one simulation can use a computer, each simulation must have an allotted time slice in which to compute a response. This time slice is the requested compute time (RCT).

In order to preserve time synchronization of all real-time jobs, the system guarantees that no job will be allowed to compute more than its allotted RCT per frame for that job. When a job does attempt to exceed the RCT, a lost time synchronization interrupt is issued by the real-time monitor and the central processor is given to another

job. It is the task of the supervisor to control and coordinate activity of a simulation after lost time synchronization interrupt occurs. A more detailed discussion of lost time execution is given in a later chapter.

E. Mode Control

The process of real-time digitial simulation requires an interactive man-machine control capability. By using the mode control keyboard, a simulation programer is able to control the flow and function of his program. This manual control is called mode control and is interpreted and coordinated by the supervisor. A detailed description of mode controls and implementation follows in a later section.

F. Real-Time Data Storage and Retrieval

During the course of a simulation, it is necessary to store information about the simulation such as values of state variables, external disturbances, and event status for later analysis. Because of real-time simulation timing constraints, Fortran input/output cannot be accomplished during real-time operation. It is also infeasible in a multiprograming system to have extensive storage of data in central memory. Therefore, it is the task of the supervisor to control and coordinate the storage on disk of data generated during real-time operation, without interfering with the timing and synchronization of the simulation.

G. Print Output Control

With the standard batch operating system, information to be printed is routed to the printer only after the job has completed all processing and has left the system. The supervisor by special communication with the operating system, can route information directly to the line printer upon command without relinquishing the central processor. This allows the programer to supplement the analog data on recording equipment with printed data at his request.

H. Error Recovery and Diagnostics

During the execution of a program, many different errors can occur. The supervisor must provide the error recovery and diagnostics necessary to maintain the integrity and effectiveness of a real-time simulation job. In a batch environment, when an error occurs, the job aborts. In real time, because of the large quantity of resources (i.e., computer, A-D conversion equipment, cockpits, etc.) and personnel required, it is best to capture the error and allow the programer the chance to fix the program, if possible, and to continue operation. It is desired that the programer not be required to provide for all contingencies, e.g., if the solution goes unstable, the supervisor will trap the error, allowing the programer to access his stored data and to reset and to begin anew.

I. Batch Job Compatability

Real-time digital simulation is expensive in terms of machine resources and execution time. Therefore, it is undesirable to do computations in real time when it is not necessary, such as during early coding checkout and purely analytic studies where real-time input and control is not needed. An additional requirement of the supervisor is the capability of operating the simulation job as a real-time job or as a normal batch job with minimum change necessary for the program.

APPENDIX B

TYPICAL ERROR DIAGNOSTICS

The following error diagnostics are a few of the typical diagnostics that may appear in the Error Directory due to programing errors.

CARD NO. 1 IS THE FIRST CARD AND IT IS NOT A PROGRAM NAME CARD

CARD NO. 2 IS AN INVALID BEGIN CARD

CARD NO. 2 IS NOT THE FIRST BEGIN PRINT

CARD NO. 2 IS A CONTINUE HOLD THAT IS NOT PRECEDED BY A DISCONTINUE

CARD NO. 2 IS GREATER THAN THE SIXTH CONTINUE PRINT

CARD NO. 2 IS GREATER THAN THE 19TH CONTINUATION CARD

CARD NO. 2 HAS AN INVALID MACRO NAME

CARD NO. 2 HAS AN ARRAY WITH FOUR SUBSCRIPTS

CARD NO. 2 CONTAINS 'a variable name' FOR TABLE WHICH EXCEEDS ITS SIZE

THE FIFTH OPERAND ON CARD NO. 2 IS INVALID

CARD NO. 2 IS AN INVALID COMMON STATEMENT

THE PRINT BLOCKS ARE NOT COMPLETE

APPENDIX C

RTPL SYNTAX

< 1, nonzero digit > :: = 1|2|3|4|5|6|7|8|9

< 2, digit > :: = 0 | < 1, nonzero digit >

< 3, real letter > :: = A|B|C|D|E|F|G|H|P|Q|R|S|T|U|V|W|X|Y|Z

< 4, integer letter > :: = I|J|K|L|M|N

< 5, non-0 letter > :: = < 3, real letter >|< 4, integer letter >

< 6, letter > :: = < 5, non-0 letter >|0

< 7, alphanumeric character > :: = < 6, letter >|< 2, digit >

< 8, default integer variable > :: = < 4, integer letter >↑6.
< 7, alphanum. c. >

< 9, octal digit > :: = 0|1|2|3|4|5|6|7

< 10, octal constant > :: = 0 < 9, 0. digit >< 9, 0. digit >
< 9,0. digit >< 9, 0. digit >< 9, 0. digit >< 9, 0. digit >↑14.
< 9, 0. digit >

< 11 > :: = < 6, letter >< 7, alphanum. c. >< 7, alphanum. c. >
< 7, alphanum. c. >< 7, alphanum. c. >< 7, alphanum. c. >

< 12 > :: = < 7, alphanum. c. >< 6, letter >< 7, alphanum. c. >
< 7, alphanum. c. >< 7, alphanum. c. >< 7, alphanum. c. >

< 13 > :: = < 7, alphanum. c. >< 7, alphanum. c. >< 6, letter >
< 7, alphanum. c. >< 7, alphanum. c. >< 7, alphanum. c. >

< 14 > :: = < 7, alphanum. c. >< 7, alphanum. c. >< 7, alphanum. c. >
< 6, letter >< 7, alphanum. c. >< 7, alphanum. c. >

< 15 > :: = < 7, alphanum. c. > < 7, alphanum. c. > < 7, alphanum. c. >
< 7, alphanum. c. > < 6, letter > < 7, alphanum. c. >

< 16 > :: = < 7, alphanum. c. > < 7, alphanum. c. > < 7, alphanum. c. >
< 7, alphanum. c. > < 7, alphanum. c. > < 6, letter >

< 17 > :: = < 11 > | < 12 > | < 13 > | < 14 > | < 15 > | < 16 >

< 18 > :: = 0 < 17 >

< 19 > :: = 0↑5. < 7, alphanum. c. >

< 20 > :: = < 3, real letter >↑6. < 7, alphanum. c. >

< 21, default real variable > :: = < 18 > | < 19 > | < 20 >

< 22, integer variable > :: = < 21, d. real var. > | < 8, d. int. var. >

< 23, real variable > :: = < 8, d. int. var. > | < 21, d. real. var. >

< 24, variable > :: = < 22, int. var. > | < 23, real var. >

< 25 > :: = 0|1|2|3|4|5|6

< 26 > :: = 1 < 25 >

< 27, small integer constant > :: = < 1, nonzero digit > | < 26 >

< 28 > :: = 1

< 29 > :: = 0|1|2|3

< 30 > :: = 0|1

< 31 > :: = 0

< 32 > :: = 0|1|2|3|4|5|6|7

< 33 > :: = 0

< 34 > :: = < 28 > < 29 > < 30 > < 31 > < 32 > < 33 >

< 35 > :: = < 2, digit >↑4. < 2, digit >

< 36, subscript constant > :: = < 34 > | < 35 >

< 37 > :: = 5

< 38 > :: = 0|1|2|3|4|5|6|7

< 39 > :: = 0|1|2|3|4|5|6

< 40 > :: = 0|1|2|3|4

< 41 > :: = 0|1|2|3|4|5|6

< 42 > :: = 0

< 43 > :: = 0|1|2|3|4|5|6|7

< 44 > :: = 0|1|2|3|4|5

< 45 > :: = 0|1|2

< 46 > :: = 0|1|2|3

< 47 > :: = 0

< 48 > :: = 0|1|2|3

< 49 > :: = 0|1|2|3|4

< 50 > :: = 0|1|2

< 51 > :: = 0|1|2|3

< 52 > :: = 0|1|2|3|4

< 53 > :: = 0|1|2|3|4|5|6|7|8

< 54 > :: = 0|1|2|3|4|5|6|7

< 55 > :: = < 37 > < 38 > < 39 > < 40 > < 41 > < 42 > < 43 > < 44 >
< 45 > < 46 > < 47 > < 48 > < 49 > < 50 > < 51 > < 52 > < 53 >
< 54 >

< 56 > :: = < 2, digit >↑16. < 2, digit >

< 57, integer constant > :: = < 55 >|< 56 >

< 58 > :: = b↑15. < 2, digit >

< 59 > :: = .↑15. < 2, digit >

< 60 > :: = b↑m < 2, digit > .↑n. < 2, digit > where m + n = 15

< 61 > :: = < 58 >|< 59 >|< 60 >

< 62, sign > :: = + | -

< 63 > :: = < 2, digit > ↑2. < 2, digit >

< 64 > :: = < 62, sign >|b

< 65 > :: = < 64 > < 63 >

< 66, exponent part > :: = < 65 >|E < 65 >

< 67, real constant > :: = < 61 > < 66, exp. part >

< 68 > :: = , < 36, subscript con. >

< 69 > :: = < 36, subscript con. > ↑2 < 36, subscript con. >

< 70 > :: = (< 69 >)

< 71, real array > :: = < 23, real var. >|< 23, real var. > < 70 >

< 72, integer array > :: = < 22, int. var. >|< 22, int. var. > < 70 >

< 73, array > :: = < 71, real arr. >|< 72, int. arr. >

< 74, signed real variable > :: = < 62, sign > < 23, real var. >

< 75, signed real constant > :: = < 62, sign > < 67, real con. >

< 76, integer variable or constant > :: = < 22, int. var. > |
< 57, int. con. >

< 77, real variable or constant > :: = < 23, real var. >|< 67, real con.>

< 78, variable or array > :: = < 23, real var. >|< 73, array >

< 79 > :: = BEGIN|END

< 80 > :: = < 79 >|CONTINUE|DISCONTINUE

< 81 > :: = DECLARATIVE|INITIALIZE|OPTION|LOSTIME

< 82 > :: = < 79 > < 81 >

< 83 > :: = < 80 > PRINT

< 84 > :: = < 80 > RESET ↑ 1. HOLD ↑ 1. OPERATE

< 85 > :: = < 80 > HOLD ↑ 1. OPERATE

< 86 > :: = < 80 > OPERATE

< 87, block statements > :: = < 82 > | < 83 > | < 84 > | < 85 > | < 86 >

< 88 > :: = < 24, var. > |,

< 89 > :: = < 24, var. > ↑ 99. < 88 > | b ↑ 99. < 88 >

< 90 > :: = CHANGE LOGIC (< 89 >)

< 91 > :: = , < 24, var. >

< 92 > :: = < 24, var. > ↑ 100. < 91 >

< 93 > :: = BEGIN MACRO < 24, var. > (< 91 >) | BEGIN MACRO < 24, var. >

< 94 > :: CALL MACRO < 24, var. > (< 91 >) | CALL MACRO < 24, var. >

< 95 > :: = END MACRO < 24, var. >

< 96, macro statements > :: = < 93 > | < 94 > | < 95 >

< 97 > :: = , < 23, real var. > |,

< 98 > :: = < 23, real var. > ↑ 199. < 97 > | b ↑ 199. < 97 >

< 99 > :: = CHANGE (< 98 >)

< 100 > :: = , < 22, int. var. > |,

< 101 > :: = < 22, int. var. > ↑ 99. < 100 > | b ↑ 99. < 100 >

< 102 > :: = CHANGE INTEGER (< 101 >)

< 103 > :: = , < 78, var. or arr. >

< 104 > :: = < 78, var. or arr. > ↑ 64. < 103 >

< 105 > :: = RECORD (< 104 >)

< 106 > :: = ADC NAME (< 23, real var. >)

< 107 > :: = DAC NAME (< 23, real var. >)

< 110 > :: = CHANGE NAME (< 23, real var. >)

< 111 > :: = REAL TIME FILE (< 24, var. >)

< 112 > :: = INTEGRATION BUFFER (< 24, var. >)

< 113 > :: = SCANNER (< 76, int. var. or con. >)

< 114 > :: = TIME INTERVAL (< 76, int. var. or con. >)

< 115 > :: = RECORDING FREQUENCY (< 76, int. var. or con. >)

< 116 > :: = < 23, real var. >,< 23, real var. >,< 77, real var. or con. >

< 117 > :: = , < 116 >

< 118 > :: = < 116 > ↑220. < 117 >

< 119 > :: = INTEGRATE (< 118 >)

< 120 > :: = ADC SKIP (< 57, int. con. >)

< 121 > :: = DAC SKIP (< 57, int. con. >)

< 122 > :: = < 74, s. real var. >|< 75, s. real con. >

< 123 > :: = ADC (< 23, real var. >,< 122 >, < 77, real var. or con. >)

< 124 > :: = DAC (< 23, real var. >,< 77, real var. or con. >,< 122 >)

< 125 > :: = < 24, var. > |b

< 126 > :: = < 27, small int. con. >|b

< 127 > :: = DISPLAY VARIABLES (< 125 >,< 125 >,< 125 >,< 125 >,< 126 >,
< 125 >,< 126 >,< 125 >,< 126 >,< 125 >)

< 128 > :: = < 23, real var. >|b

< 129 > :: = < 22, int. var. >|b

< 130 > :: = SET INTEGRATION (< 128 >,< 128 >,< 129 >,< 129 >,< 129 >,
< 125 >)

< 131 > :: = MODES AND DISCRETES (< 125 >,< 125 >,< 125 >,< 125 >,
< 125 >,< 125 >)

< 132. nonblock statements > :: = < 90 >|< 99 >|< 102 >|< 105 >|< 106 >|
< 107 >|< 108 >|< 109 >|< 110 >|< 111 >|< 112 >|< 113 >|< 114 >|
< 115 >|< 119 >|<120>|<121>|< 123 >|< 124 >|< 127 >|< 130 >|< 131 >
< 133, a formal parameter in the body of a macro > :: = $$ < 24, var. > $$

The preprocessor does not parse Fortran statements except Fortran declaratives and formal parameters in the body of a macro definition. The syntactical definition of formal parameters is stated above and appendix D contains the definitions of Fortran declarative

APPENDIX D

FORTRAN DECLARATIVE SYNTAX

The following syntactical definitions are taken from the RTPL syntax:

< 2, digit >,< 24, var. >, and < 70 >

< 201 > :: = COMPLEX|DOUBLE|PRECISION|DOUBLE|REAL|INTEGER|LOGICAL

< 202, TYPE NAME > :: = < 201 >|TYPE < 201 >

< 203, array > :: = < 24, var. > < 70 >

< 204 > :: = < 24, var. > < 203, array >

< 205 > :: = , < 204 >

< 206, list T > :: = < 204 >↑ . < 205 > *

< 207, type statement > :: = < 203, array > < 206, list T >

< 208, common identifier > :: = < 2, digit >↑ 6. < 2, digit > |
< 24, var. >

< 209, common label A > :: = b|//

< 210, common unit A > :: = < 209, common label A > < 206, list T >

< 211, blank common statement > :: = COMMON < 210, common unit A >
↑ . < 210, common unit A >

< 212, common label B > :: = / < 208, common identifier > /

< 213, common unit B > :: = < 212, com. lable B > < 206, list T >

< 214, labeled common statement > :: = COMMON < 213, common unit B >
60. < 213, common unit B >

*Number of times the syntactical unit can be repeated is unspecified.

< 215 > :: = , < 203, arr. >

< 216, list D > :: = < 203, array >↑ . < 215 >

< 217, dimension statement > :: = DIMENSION < 216, list D >

APPENDIX E

SAMPLE INPUT AND OUTPUT PROGRAMS

A sample program in RTPL and the resulting Fortran real-time program has been included to illustrate the use of RTPL.

INPUT PROGRAM

```
      PROGRAM YDD (INPUT=201,OUTPUT=201)
R     BEGIN DECLARATIVE
R     INTEGRATE(PSI,PSIDT,PSI0,PSID,PSIDD,PSID0)
R     CHANGE(PSI0,PSID0,A,B,,)
R     CHANDE(ANSWER1,ANSWER2)
R     CHANGE INTEGER(N)
R     CHANGE LOGIC ( FLAG,FLAG2)
R     ADC(FORCE1,-.1,.06)
R     ADC(FORCE2,+1.333,.001)
R     DAC(PSI,.1,-.3)
R     DAC SKIP (5)
R     DAC(PSID,.7,+D)
R     RECORD(T,PSI,PSID,PSIDD,A,B)
R     BEGIN MACRO ROOTS(D,F,F,U,V)
      POS = .F.
      NEG = .F.
      $$V$$=$$E$$*$$E$$-4.*$$D$$*$$F$$
      $$U$$=1.
      IF(SIGN($$U$$,$$V$$).GT.0) POS=.T.
      IF(SIGN($$U$$,$$V$$).LT.0) NEG=.T.
      IF(POS) $$U$$=(-$$E$$+SQRT($$V$$))/(2.*$$D$$)
      IF(POS) $$V$$=(-$$E$$-SQRT($$V$$))/(2.*$$D$$)
      IF(NEQ) $$U$$=0
      IF(NEQ) $$V      $$=0
R     END MACRO ROOTS
R     END DECLARATIVE
R     BEGIN INITIALIZE

C**** SECTION D.  CONSTANTS AND INITIAL PARAMETERS

    1 FORMAT (1H15X4HTIME,17X3HPSI,18X7HPSI DOT,14X9HPSI D
     1 DOT,12X,1HA,20X1HB)
    2 FORMAT (6E21.8)
    3 FORMAT (F10.2)
      N = 2
      A = 5.201875
      B = 2.57397
      C = 8.3333
      D = .12
      PSI0 = 0.2
      PSID0 = 0.0
      T0 = 0.0
R     END INITIALIZE
R     BEGIN RESET

C**** SECTION E.  INITIALIZATION OF INTEGRALS
```

```
      T = T0
R     CALL  MACRO ROOTS (A,B,C,ANSWER1,ANSWER2)
R     BEGIN HOLD

C**** SECTION F.  HOLD CONTROL

R     BEGIN OPERATE

C**** SECTION G.  OPERATE LOOP

      PSIDT = PSID
      PSIDD = -A*SIGN(1.0,PSID)*PSID**N - B*SIN(PSI) + FOR
     1CE1 - FORCE2*SIN(PSI)
      TABLE(5) = T
      TABLE(6) = PSI
R     END RESET HOLD
R     DISCONTINUE OPERATE
R     CONTINUE OPERATE
R     END OPERATE
R     BEGIN PRINT

C**** SECTION H.  PRINT CONTROL

      WRITE (MF,1)
R     DISCONTINUE PRINT
      WRITE(MF,2) T,PSI,PSID,PSIDD,A,B
R     CONTINUE PRINT
R     END PRINT
R     BEGIN OPTION

C**** SECTION I.  READ CONTROL

      READ 3,A
R     END OPTION
      END
```

OUTPUT PROGRAM

```
 PROGRAM YDD (INPUT=201,OUTPUT=201)
 COMMON /INTCOMM /T,H,INT,NEQ,ISCHEME,DERINT(2,2)
 COMMON /INTINTR/ INTERN ( 5, 2)
 EQUIVALENCE (DERINT(1,1),PSI),(DERINT(2,1),PSID),(DF
1RINT(1,2),PSID),(DERINT(2,2),PSIDD)
 COMMON /MASKS/TMASK(60),FMASK(60)
 LOGICAL LOGIC
 DIMENSION TABLE(8),INTEG(1),LOGIC(2)
 LOGICAL VARCHNG,FS14,FS15,FS16,ENABLE,MSI
 EQUIVALENCE (TABLE(1),PSI0),(TABLE(2),PSID0),(TABLE(
13),A),(TABLE(4),B),(TABLE(7),ANSWER1),(TABLE(8),ANSW
2ER2)
 EQUIVALENCE (INTEG(1),N)
 EQUIVALENCE (LOGIC(1),FLAG),(LOGIC(2),FLAG2)
 DIMENSION ADC(80),DAC(180)
 COMMON /INOUT/ REMOTF,IDIS(16),ODIS(16),TYIO(4)
 DIMENSION IVARBUF(5)
 CALL RESET   (90001S,90007S)
 CALL HOLD    (90002S,90007S)
 CALL OPERATE(90003S,90008S,90009S,90010S)
 CALL PRINT(90011S,90012S,90013S,90014S)
 CALL OPTION(90015S,90016S)
 CALL READOUT(6,T,PSI,PSID,PSIDD,A,B)
 CALL RTOUTF(MF)
 CALL INOUT(ADC,2,DAC,7)
 CALL DATABLX(TABLE,8,INTEG,1,LOGIC,2,ADC,2,DAC,7,IDI
1S,16,ODIS,16)
 CALL XDSPLAY(IDIS,ODIS,VARCHNG,ITYPE,IVARBUF,FS16)
 CALL NM218(6LTYPTPE)
 ISCHEME=1
 NEQ=2
 H=32./1024.
 FLAG=.F.

C**** SECTION D.  CONSTANTS AND INITIAL PARAMETERS

     1 FORMAT (1H15X4HTIME,17X3HPSI,18X7HPSI DOT,14X9HPSI D
      1 DOT,12X,1HA,20X1HB)
     2 FORMAT (6E21.8)
     3 FORMAT (F10.2)
       N = 2
       A = 5.201875
       B = 2.57392
       C = 8.3333
       D = .12
       PSI0 = 0.2
       PSID0 = 0.0
```

```
      T0 = 0.0
      CALL READY
90001 CONTINUE
      INT=0
      PSI=PSI0
      PSID=PSID0

C**** SECTION F.  INITIALIZATION OF INTEGRALS

      T = T0
      POS = .F.
      NEG = .F.
      ANSWER2=B*B-4.A*C
      ANSWER1=1.
      IF(SIGN(ANSWER1,ANSWER2).GT.0.) POS=.T.
      IF(SIGN(ANSWER1,ANSWER2).LT.0.) NEG=.T.
      IF(POS) ANSWER1=(-B+SQRT(ANSWER2))/(2.*A)
      IF(POS) ANSWER2=(-B-SQRT(ANSWER2))/(2.*A)
      IF(NEG) ANSWER1=0.
      IF(NEG) ANSWER2=0.
90002 CONTINUE

C**** SECTION F.  HOLD CONTROL

98003 CONTINUE
      FORCE1 = (ADC( 1)-.1)*.06
      FORCE2 = (ADC( 2)+.333)*.001
      FS14=IDIS.AND.FMASK(46)
      FS15=IDIS.AND.FMASK(47)
      FS16=IDIS.AND.FMASK(48)
      MS1=IDIS.AND.FMASK(17)
90004 CONTINUE

C**** SECTION G.  OPERATE LOOP

      PSIDT = PSID
     1CE1 - FORCE2*SIN(PSI)
      PSIDD = -A*SIGN(1.0,PSID)*PSID**N - B*SIN(PSI) + FOR
      TABLE(5) = T
      TABLE(6) = PSI
      IF(INT.GT.1) GO TO 90006
      DAC( 1) = (PSI*.1)-.3
      DAC( 7) = (PSID*.7)+0
      IF(FLAG) GO TO 90005
      IF(IDIS.AND.TMASK(22)) CALL SCANNER(32)
      CALL DSPLAY
      IF(MS1) GO TO 90007
      IF(VARCHNG) CALL TYPVAR
      IF(ENABLE.AND.FS15) CALL TYPEVAR
      ENABLE=.NOT.FS15
      IF(FS14.AND.(IDIS.AND.TMASK(14))) CALL TYPEVAR
```

```
90007 RETURN
90008 RETURN
90009 CONTINUE
      CALL RECORD(32)
90006 CONTINUE
      CALL IGRATE1
      IF(INT.GT.1) GO TO 90004
      FLAG=.T.
      GO TO 90004
98005 FLAG=.F.
90010 RETURN
90011 CONTINUE

C**** SECTION H.  PRINT CONTROL

      WRITE (MF,1)
90012 CONTINUE
      WRITE(MF,2) T,PSI,PSID,PSIDD,A,B
90013 RETURN
90014 RETURN
90015 CONTINUE

C**** SECTION I.  READ CONTROL

      READ 3,A
90016 RETURN
      END
```

APPENDIX F

PREPROCESSOR SUBROUTINES

The following subroutines are used by the preprocessor in developing a real-time Fortran program. Their name and function are listed.

Name	Function
BLA	to remove blanks from a string of Fortran code in Hollerith form and write the modified code in Temp, the temporary output file.
CCON	to parse a string of input code to determine if the string contains an integer constant, floating point constant.
CEND	to parse a string of input code to determine if the string is an End card.
CMACRO	to fetch a macro from the macro definition file, parse the operands of the macro call, replace formal parameters with actual parameters, clean up the Fortran code, and write the expanded macro on Temp.
COMPARE	to compare two strings, one with one character per word and the other with 10 characters per word.
CPRONAM	to parse a string of input code to determine if the string is a program name card.
CRECORD	to write the subroutine calls to RECORD.
CSMINT	to parse a string of input code to determine if the string contains a small integer constant between one and 16 exclusively.

Name	Function
CVAR	to parse a string of input code to determine if the string contains a variable, a real variable, an integer variable, an array, a real array or an integer array in Fortran.
DEREQU	to write equivalence statements to equate integrated variables to the array in common INTCOMM (DERINT array).
INTEQU	to write equivalence statements to equate integer variables that will be displayed to the display integer array (INTEG array).
LOGEQU	to write equivalence statements to equate logical variables that will be displayed to the display logical array (LOGIC array).
PACK	to remove blanks from a string of Fortran code.
PARSED	to parse Fortran declarative statements.

The following subroutines are used to parse operands of RTPL statements. For these subroutines the name and corresponding RTPL statements will be listed.

Subroutine	RTPL Statements
SUB1	DAC
SUB2	ADC
SUB4	CHANGE
SUB6	RECORD
SUB7	SCHEME, ADC SKIP, DAC SKIP, TIME INTERVAL

Subroutine	RTPL Statements
SUB8	ADC NAME, DAC NAME, CHANGE NAME, REAL TIME FILE, CHANGE LOGIC NAME, INTEGRATION BUFFER, CHANGE INTEGER NAME.
SUB12	SCANNER, RECORDING FREQUENCY
SUB13	INTEGRATE
SUB15	CHANGE LOGIC
SUB19	CHANGE INTEGER
SUB20	SET INTEGRATION
SUB22	DISPLAY VARIABLES
SUB25	MODES AND DISCRETES

APPENDIX G

FLOW DIAGRAM OF THE CVAR SUBROUTINE

The CVAR subroutine is used to detect all variables. Flags and parameters are passed to the subroutine to indicate if the routine is to try to detect:

a variable,

a real variable,

an integer variable,

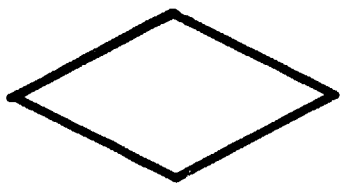
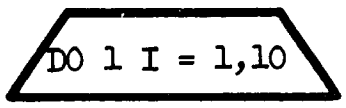
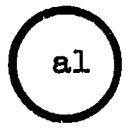
an array,

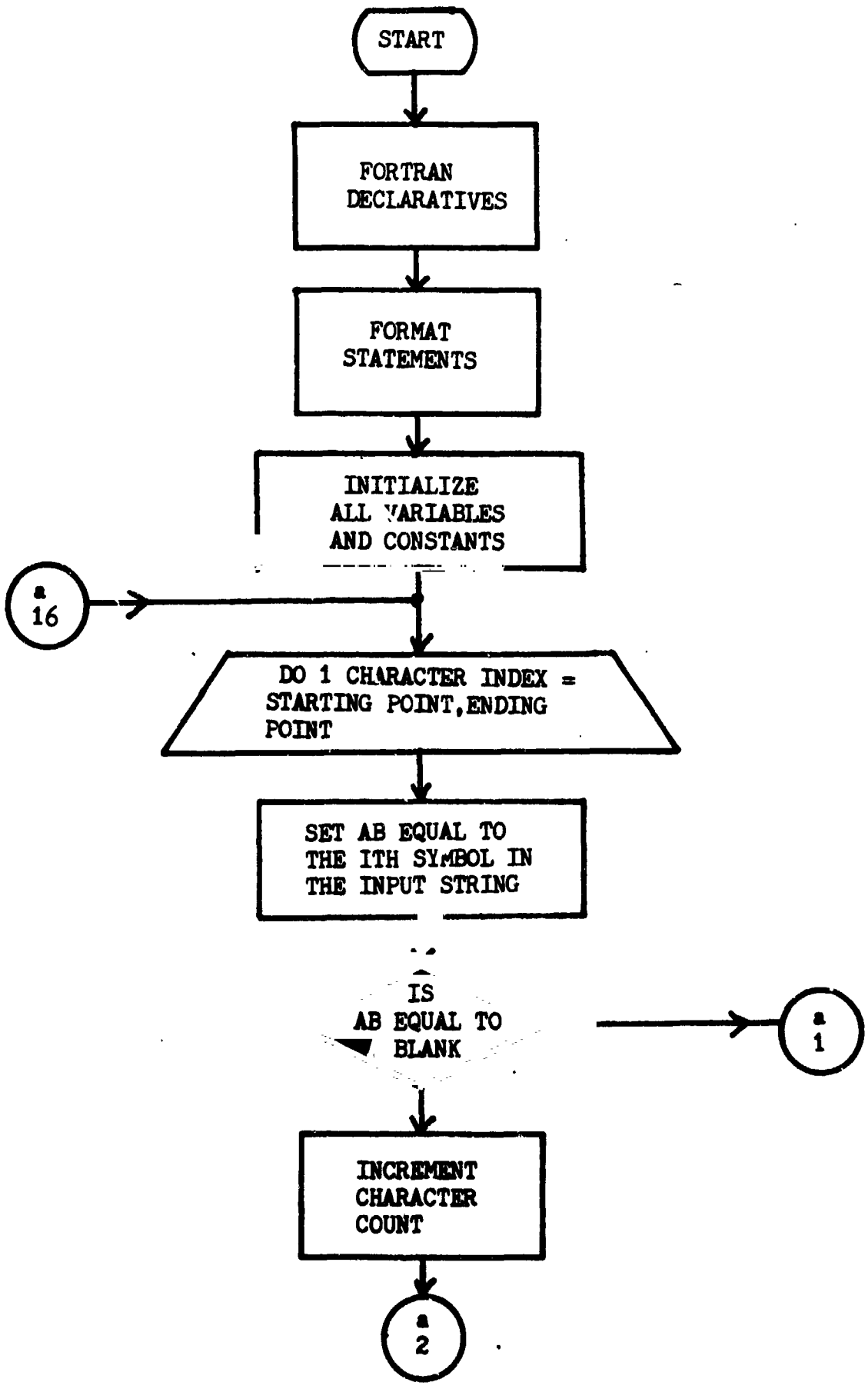
a real array, or

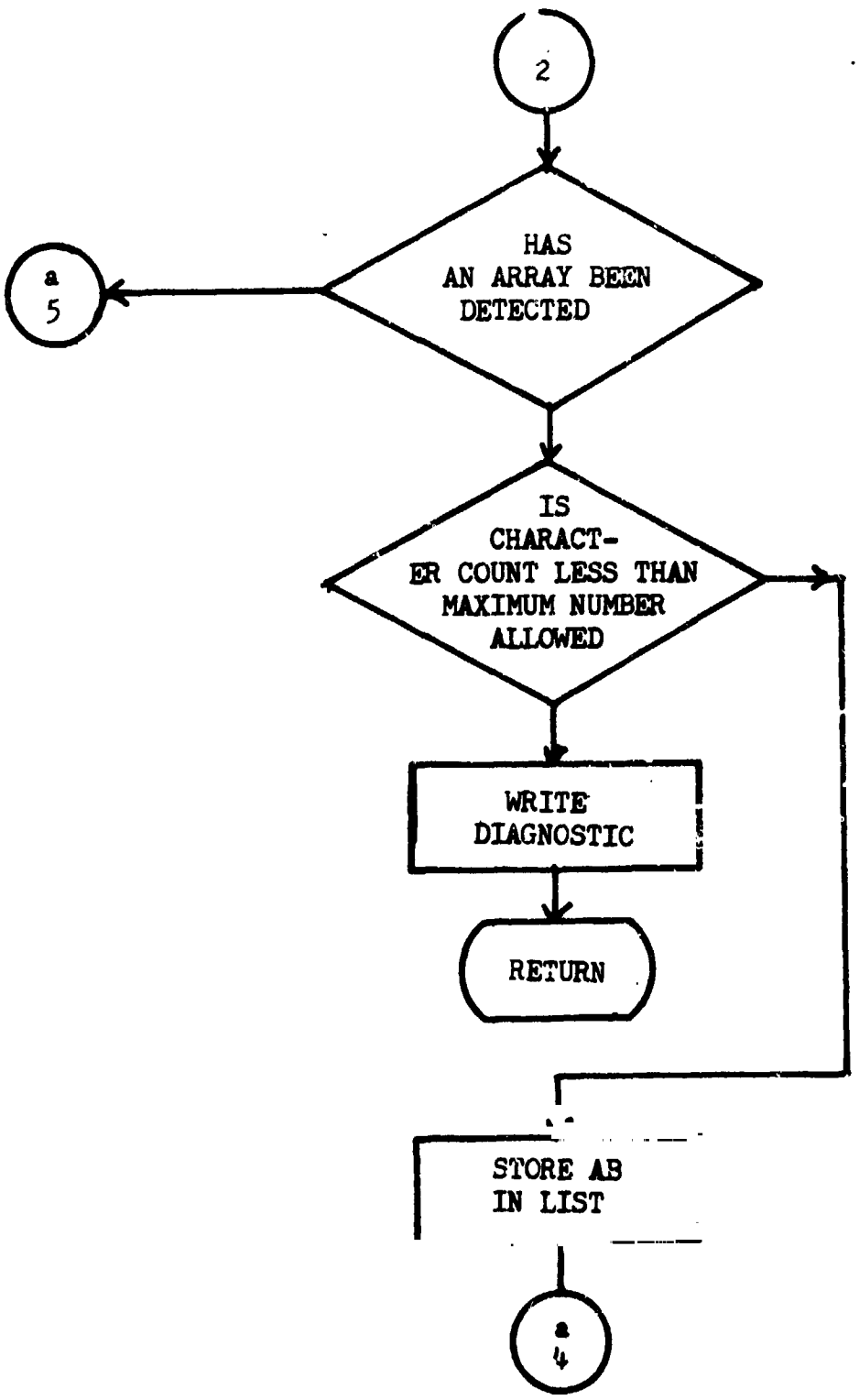
an integer array

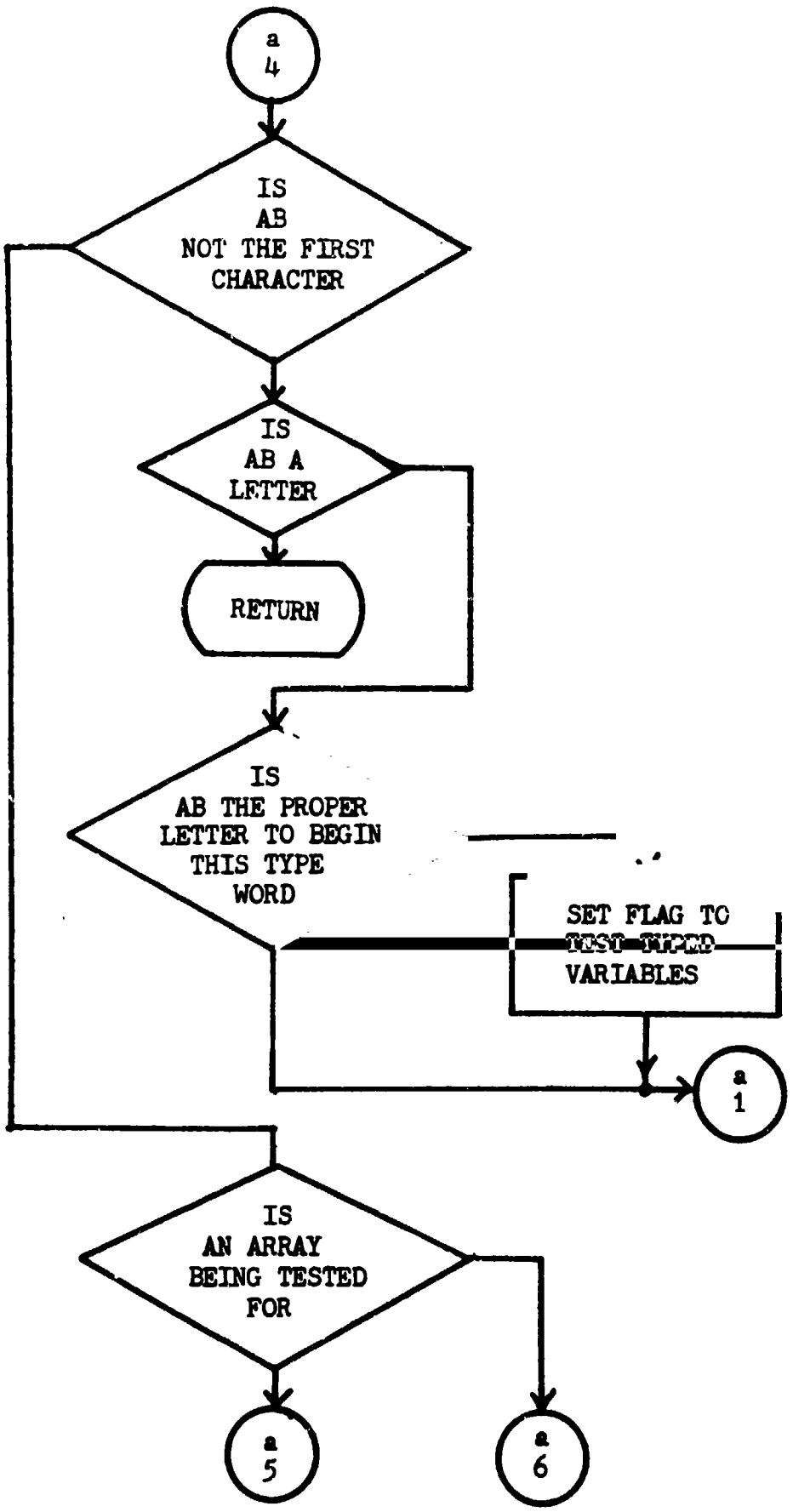
using the syntax of RTPL or Fortran depending whether the preprocessor is parsing a RTPL or Fortran statement. This subroutine is called with a pointer indicating where to start operating on the input string. The subroutine takes the first character and performs a series of tests on it to determine what it is. The routine operates on each consecutive characters until a valid name is detected. Once a variable or an array name is located, the subroutine checks the declarative tables that the Fortran parser generated. If a valid variable or array is detected, the answer is set true and the Fortran symbol is passed to the program that called CVAR. 'ENCODE' and 'DECODE' are Fortran statements that perform a transfer and modification of the contents of memory to another location in memory. They are used form words out of a string of characters (ENCODE) and to convert a symbol into a string of characters.

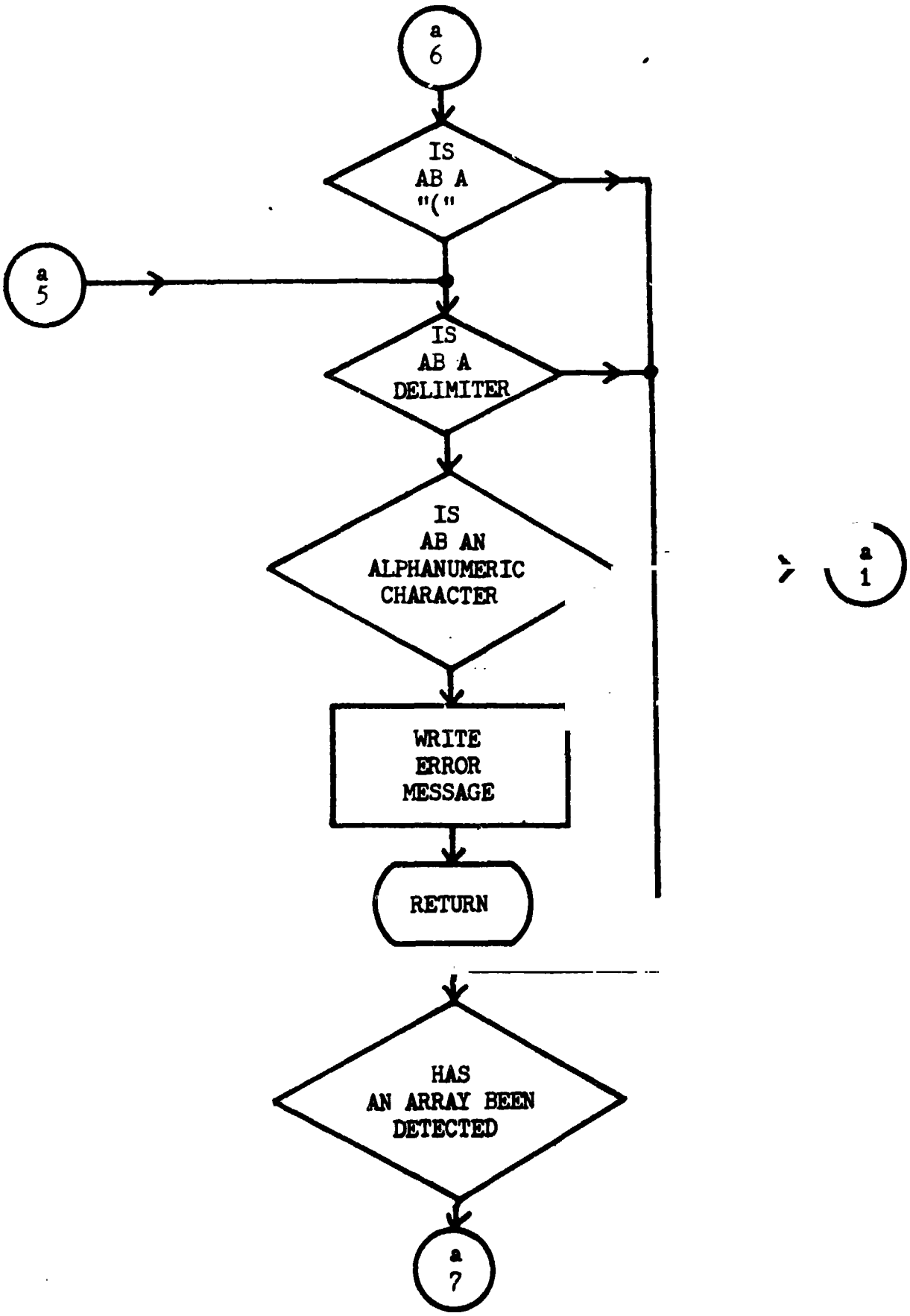
The following flow chart uses the following symbols:

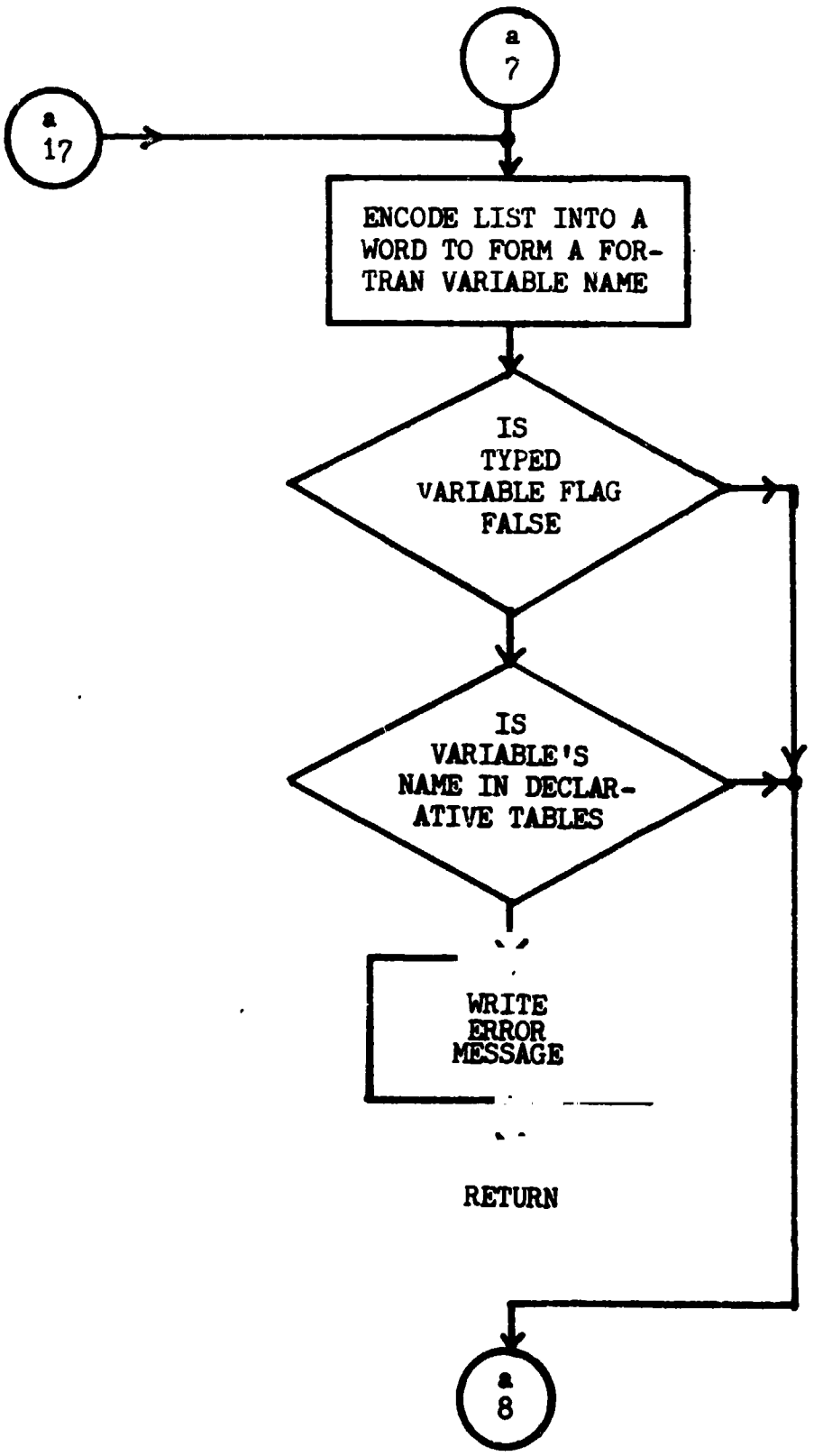
Symbol	Meaning
	Fortran code
	Test, usually an if statement or a larger group of code, bottom point is a false transfer, and either side points are true transfers.
	DO loop
	
	Starting or ending point of a subroutine
	Continuation point in flow chart. The symbol, a1, indicates how to connect a point on one page to a point on another page.

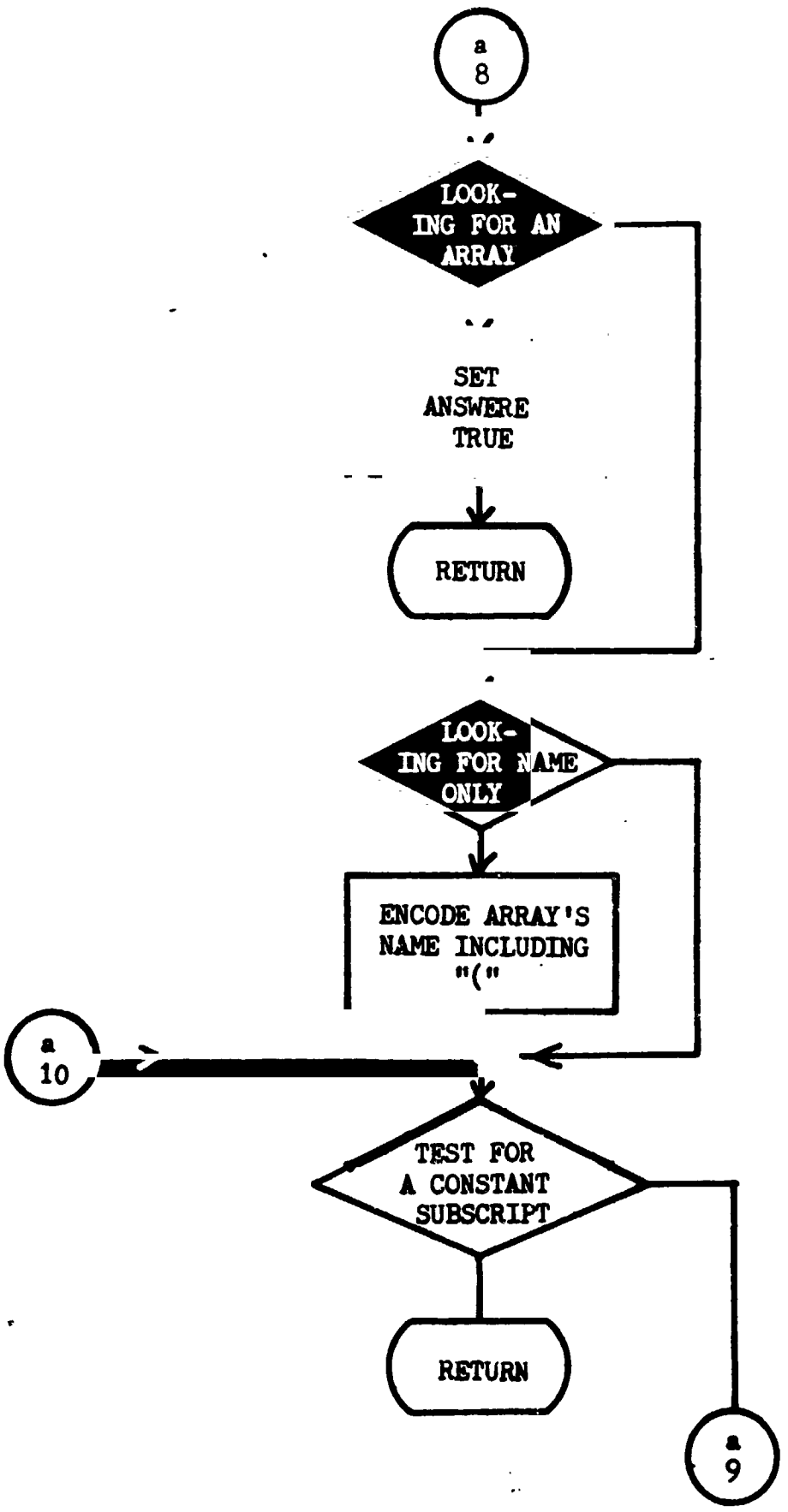

START
FORTRAN DECLARATIVES
FORMAT STATEMENTS
INITIALIZE ALL VARIABLES AND CONSTANTS
a 16
DO 1 CHARACTER INDEX = STARTING POINT,ENDING POINT
SET AB EQUAL TO THE ITH SYMBOL IN THE INPUT STRING
IS AB EQUAL TO BLANK
a 1
INCREMENT CHARACTER COUNT
a 2

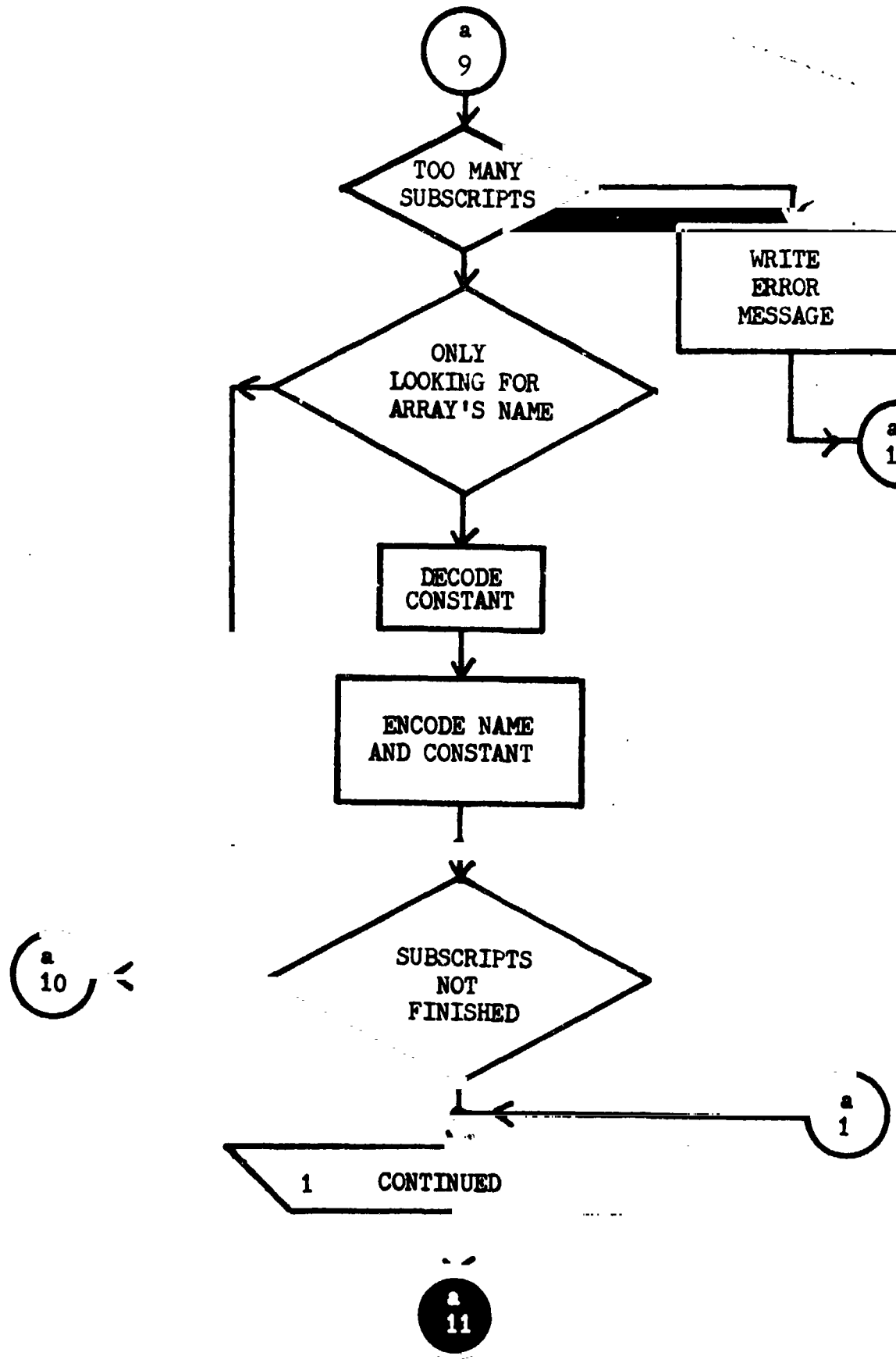

2
HAS AN ARRAY BEEN DETECTED
a 5
IS CHARACT-ER COUNT LESS THAN MAXIMUM NUMBER ALLOWED
WRITE DIAGNOSTIC
RETURN
STORE AB IN LIST
a 4

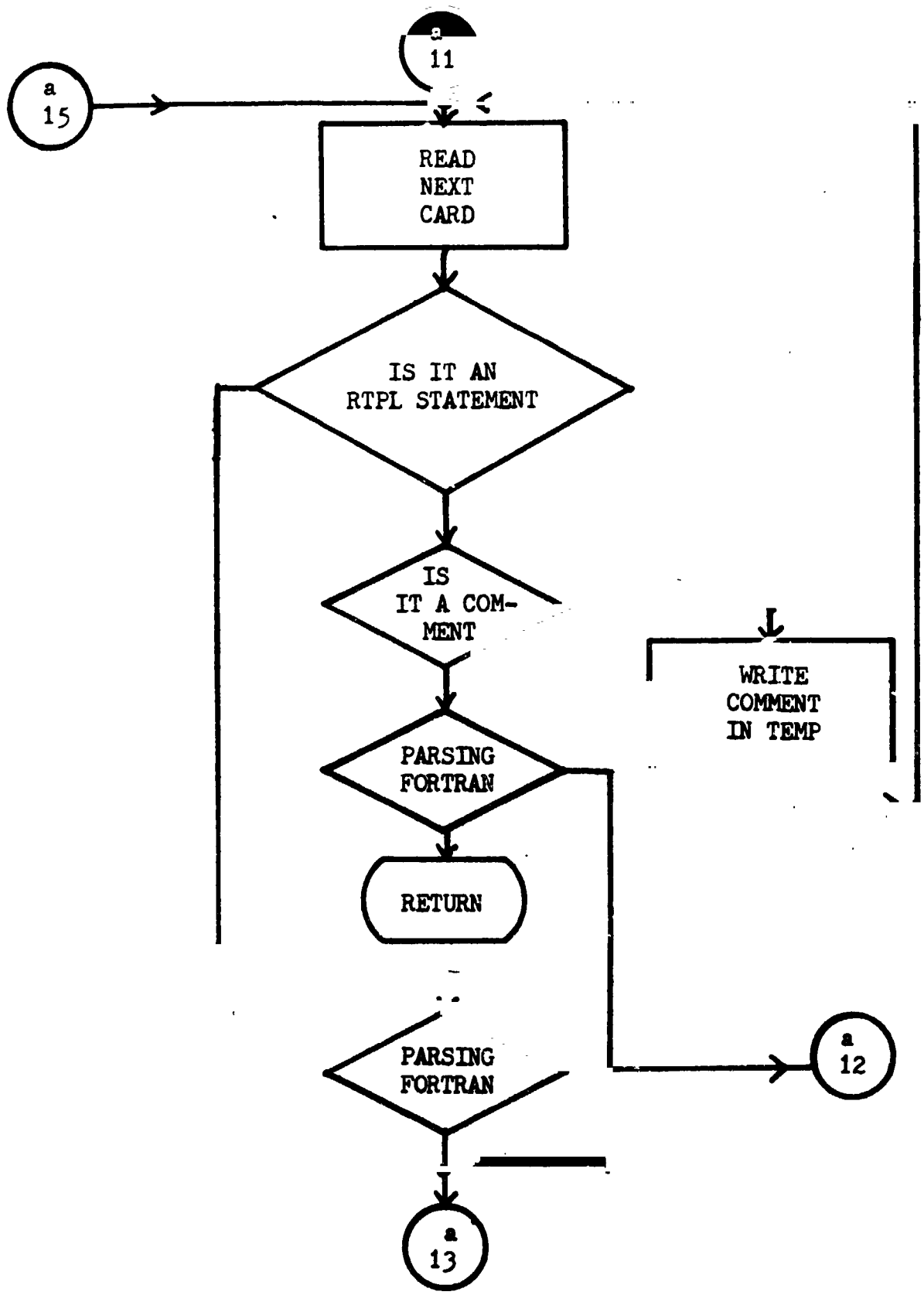

a
4
IS
AB
NOT THE FIRST
CHARACTER
IS
AB A
LETTER
RETURN
IS
AB THE PROPER
LETTER TO BEGIN
THIS TYPE
WORD
SET FLAG TO
VARIABLES
a
1
IS
AN ARRAY
BEING TESTED
FOR
a
5
a
6

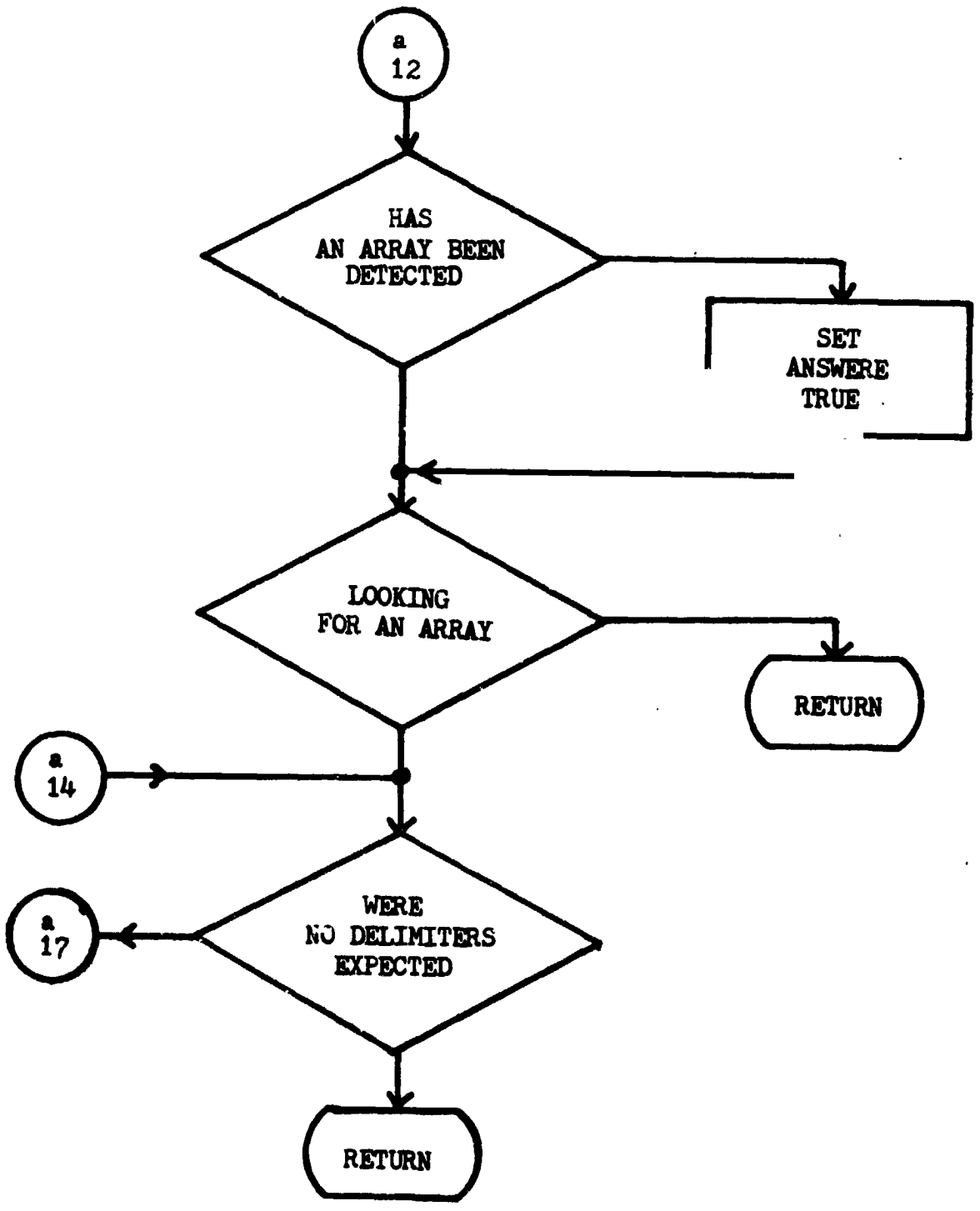

a
6
IS
AB A
"("
a
5
IS
AB A
DELIMITER
IS
AB AN
ALPHANUMERIC
CHARACTER
a
1
WRITE
ERROR
MESSAGE
RETURN
HAS
AN ARRAY BEEN
DETECTED
a
7

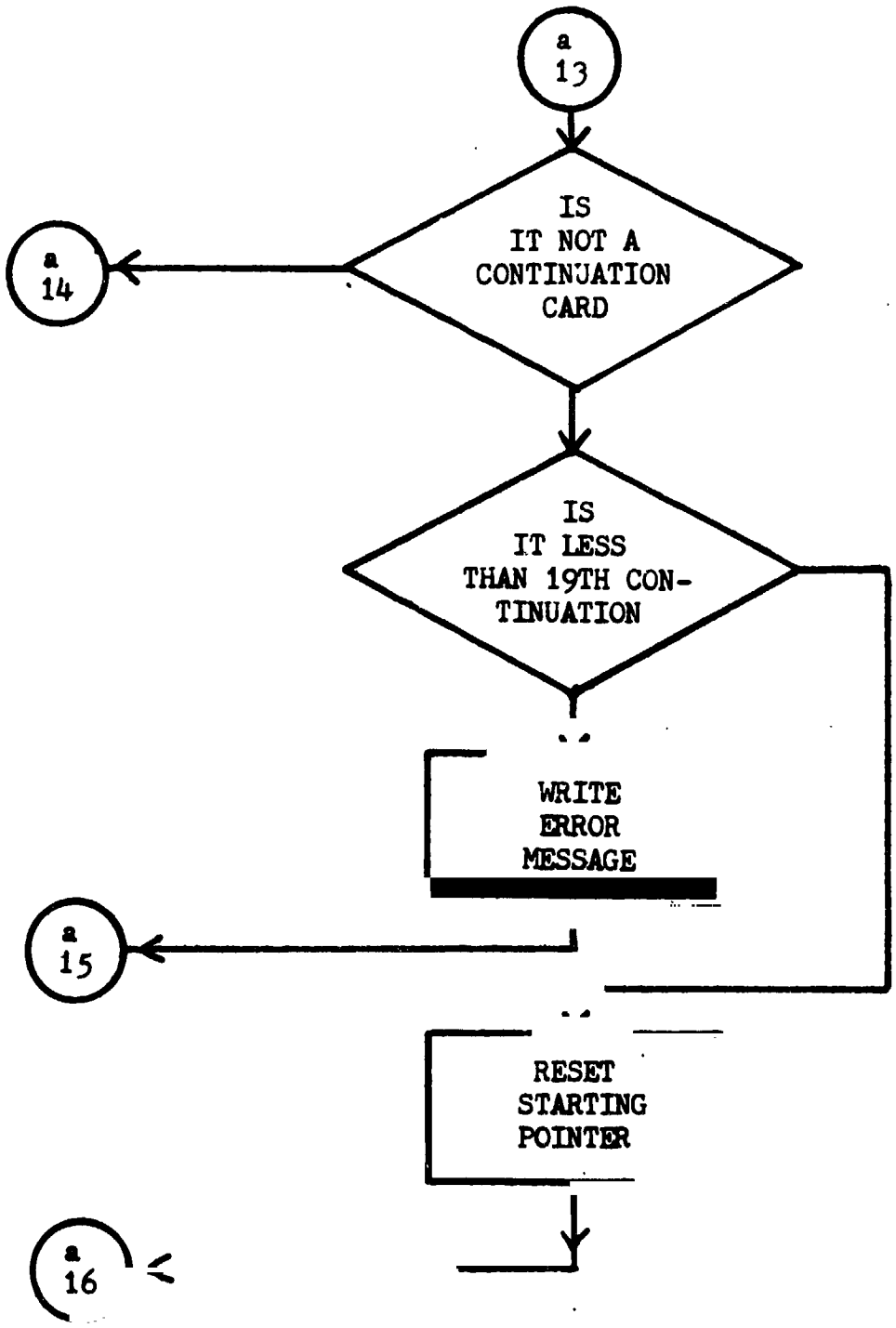

a
7
a
17
ENCODE LIST INTO A WORD TO FORM A FOR-TRAN VARIABLE NAME
IS TYPED VARIABLE FLAG FALSE
IS VARIABLE'S NAME IN DECLAR-ATIVE TABLES
WRITE ERROR MESSAGE
RETURN
a
8


a
8
LOOK-
ING FOR AN
ARRAY
SET
ANSWERE
TRUE
RETURN
LOOK-
ING FOR NAME
ONLY
ENCODE ARRAY'S
NAME INCLUDING
"("
a
10
TEST FOR
A CONSTANT
SUBSCRIPT
RETURN
a
9


a
9
TOO MANY
SUBSCRIPTS
WRITE
ERROR
MESSAGE
a
10
ONLY
LOOKING FOR
ARRAY'S NAME
DECODE
CONSTANT
ENCODE NAME
AND CONSTANT
a
10
SUBSCRIPTS
NOT
FINISHED
a
1
1 CONTINUED
a
11


a
11
a
15
READ
NEXT
CARD
IS IT AN
RTPL STATEMENT
IS
IT A COM-
MENT
WRITE
COMMENT
IN TEMP
PARSING
FORTRAN
RETURN
PARSING
FORTRAN
a
12
a
13


a
12
HAS
AN ARRAY BEEN
DETECTED
SET
ANSWERE
TRUE
LOOKING
FOR AN ARRAY
RETURN
a
14
WERE
NO DELIMITERS
EXPECTED
a
17
RETURN


a
13
IS
IT NOT A
CONTINUATION
CARD
a
14
IS
IT LESS
THAN 19TH CON-
TINUATION
WRITE
ERROR
MESSAGE
a
15
RESET
STARTING
POINTER
a
16

BIBLIOGRAPHY

1. Eckhardt, Dave E., Jr.: Description of Langley Research Center Computer Complex and Special Features for Real-Time Simulation Applications. Paper presented at the Eastern Simulation Council Meeting, Hampton, Virginia, September 26, 1968.

2. Cleveland, Jeff I., II: Description of Software Features for Program Control. Paper presented at the Eastern Simulation Council Meeting, Hampton, Virginia, September 26, 1968.

3. Cleveland, Jeff I., II: A Real-Time Digital Simulation Supervisor. Thesis for George Washington University, April 1970.

4. Crawford, Daniel J.; and Cleveland, Jeff I., II: Real-Time Digital Simulation Cooperative Programing Guide. Internal manual for National Aeronautics and Space Administration, Langley Research Center Real-Time Simulation Facility, February 1969.

5. Rabinowitz, I.: Report Algorithmic Language Fortran II. Communications of the ACM, vol. 5, no.6 , June 1962, pp. 327-337.

6. Burkhardt, W. H.: Metalanguage and Syntax Specification. Communications of the ACM, vol. 8, no.5 , May 1965, pp. 304-305.

7. Hoffberg, Susan S.; and Goldstein, Max: A Syntax-Directed Fortran Statement Checker. Courant Institute of Mathematical Sciences, January 1968.

8. Brever, Hans: Dictionary for Computer Languages. London Academic Press Inc., LTD., 1966.