NASA
Technical
Paper
2422
c.2
June 1985

Real-Time Multiprocessor Programming Language (RTMPL)

Users Manual

Property of U. S. Air Force
AEDC LIBRARY
F40600-81-C-0004

Dale J. Arpasi

TECHNICAL REPORTS
FILE COPY

NASA

NASA Technical Paper 2422

1985

Real-Time Multiprocessor Programming Language (RTMPL)

Users Manual

Dale J. Arpasi

Lewis Research Center
Cleveland, Ohio

NASA
National Aeronautics and Space Administration

Scientific and Technical Information Branch

Contents

Summary

The NASA Lewis Research Center is developing and evaluating experimental hardware and software systems to help meet future needs for real-time, high-fidelity simulations of dynamic systems. Specifically, the Real-Time Multiprocessor Simulator (RTMPS) project focuses on the use of multiple microcomputers to achieve the required computing speed and accuracy at relatively low cost. A real-time multiprocessor programming language (RTMPL) has been developed to provide high-order language (HOL) programming of RTMPS systems. The RTMPL programming utility (translator) supports a variety of multiprocessor configurations and microcomputer types. The utility serves as an assembly language programmer. It translates the HOL source program for each RTMPS computer to a time-efficient assembler source program and sets up all data communication between the computers.

This manual describes the RTMPL from a user's viewpoint. A programming example is presented to illustrate the use of the RTMPL utility to program an RTMPS system consisting of six MC68000-based computers. The RTMPL source programs and translator listings are described, as well as the utility output files, including the assembler source code programs for each computer in the simulator, and a comprehensive data base that describes the simulation. The use of the data base in conjunction with a NASA-developed real-time multiprocessor operating system (RTMPOS) for interactive execution of the simulator is discussed. Finally the unique features of RTMPL are summarized.

Introduction

A real-time multiprocessor simulator (RTMPS) is being developed at the NASA Lewis Research Center (ref. 1). It is used to develop and evaluate experimental hardware and software systems that will allow real-time, interactive simulation of dynamic systems. The RTMPS project is focusing on the use of multiple microcomputers to achieve the required computing speed and accuracy at low cost (relative to mainframe digital and hybrid computers). A related goal is to devise a programming methodology that will permit engineering-level personnel to easily generate time-efficient code for the simulator and to conveniently operate the simulator.

A real-time multiprocessor programmers language (RTMPL) has been developed to provide high-order language (HOL) programming of RTMPS systems. The RTMPL programming utility (translator) supports a variety of multiprocessor configurations and microprocessors. The utility acts as an assembly language programmer. It translates the RTMPL source program for each RTMPS computer to a time-efficient assembler source program and sets up all data communication between the computers. The RTMPL utility is written in Pascal and is designed to run on a host computer under a disk operating system.

The NASA Lewis implementation of RTMPL runs on a Motorola EXORmacs[1] development system under the VERSAdos[1] operating system. The user generates RTMPL source files describing the simulation. The utility translates these source files into assembly language program files for the simulator computers. The utility generates an extensive listing file to aid the user in debugging the simulation and minimizing its execution time. The RTMPL language is macro based. Therefore only the macros have to be rewritten for different processors (typically done by system programmers). The current versions of the macros have been written for the Motorola MC68000 processor. The RTMPL utility also generates data-base files that describe the simulation to a real-time multiprocessor operating system (RTMPOS) (refs. 2 and 3). The RTMPOS and RTMPL complement each other, providing a unique environment for programming and engineering-level, interactive execution of the simulation.

Reference 4 describes the RTMPL concept, design philosophy, general features, and relationships to the simulator hardware and operating system. It also provides a general orientation and discussion of the language. The intent of this manual is to familiarize the RTMPL user with the language constructs and the functions, capabilities, and limitations of the RTMPL utility.

This users manual is organized so as to provide a top-down introduction to RTMPL programming. Since

[1]Motorola trademark.

the RTMPL was developed to program a general multiprocessor simulator configuration, that general configuration and the methods used to target the RTMPL utility to subsets of that configuration are described. That description is followed by a discussion of interprocessor information transfer. The RTMPL input/output file structure is then presented, and the function and interrelationships of these files are described in the context of interactive, real-time simulation. Each input file is then syntactically defined in terms of language constructs. Having established the language definition, a simple simulation example is developed in detail and used to illustrate the use of the RTMPL utility, the resultant listings, and the output files. Methods of using the listings to develop optimized simulations are also discussed.

Chapter 1: Application to Multiprocessor Configurations

Generally a multiprocessor system consists of a number of individual computers communicating with each other. To be solved on a multiprocessor system, a problem must be segmented. Each segment is assigned to a separate computer so that the problem may be solved in parallel, providing answers in less time than possible with a single computer. Generally this improvement in performance is gained at the cost of greater programming complexity since the transmission and reception of data on the individual computers must be handled by the programmer. To make the programming of a multiprocessor system attractive to engineering-level users, a high-order language (HOL) is needed that can automate these communications. Ideally the HOL will allow the user to program the system as a whole rather than on a computer-by-computer basis.

A multiprocessor system can be configured with a wide variety of communication paths (architecture) and computer hardware. To avoid obsolescence and to improve simulation transportability, the HOL must be conveniently targetable in terms of both simulator architecture and hardware. That is, the utility that translates the HOL should do so according to information describing the specifics of the target simulator so as to avoid the necessity of generating a new utility for each simulator. Further the HOL should allow the user to select the number of computers and communication paths (within the limits of the target simulator) to be used to solve a particular problem. Finally the HOL should automate information transfer and synchronization within the simulator on the basis of information contained in the problem statement.

This section describes the multiprocessor configurations that are programmable in RTMPL. It also presents an overview of the targeting methods used by the RTMPL utility to generate code for specific configurations and components. Also included is a discussion of the information transfer and past-value retention features of RTMPL, which allow the user to structure simulations for the shortest execution time.

General Simulator Configuration

RTMPL was developed to program the general multiprocessor simulator configuration shown in figure 1. Subsets of this configuration are also supported. The primary elements in the general configuration are the front-end processor (FEP), a real-time interface, and a number of simulation channels. Data are transferred between the simulation channels via the interactive information bus and the real-time information bus.

The FEP serves as the simulation controller and user interface. The FEP and its resident disk operating system provide for simulator run-time operations such as program loading, simulator mode control, data handling, and data display. The FEP also services the simulator peripherals (terminals, disks, printers, etc.). File-handling services are provided by the disk operating

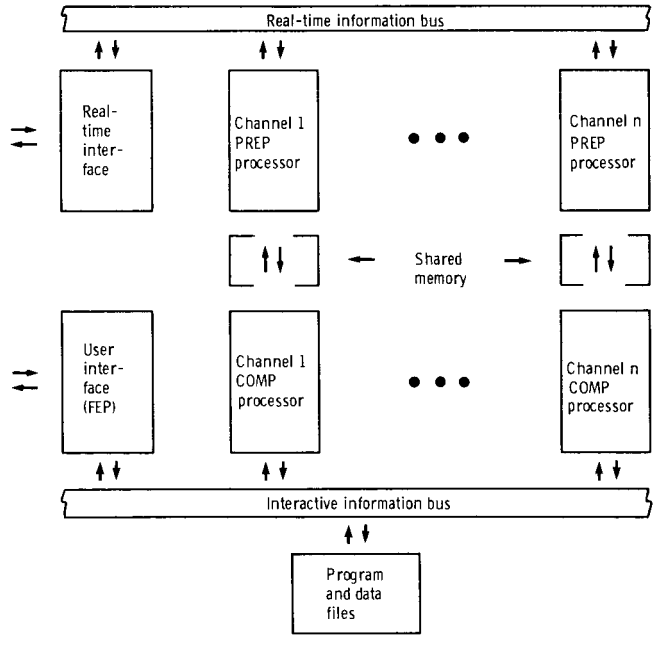


Figure 1. - General simulator configuration.

system. The FEP is the bus controller for the interactive information bus. All communications between the FEP and the simulation channels are via this bus. In the Lewis RTMPS a real-time multiprocessor operating system (RTMPOS) (refs. 2 and 3) works in conjunction with the FEP manufacturer's disk operating system to perform the required functions.

The real-time interface serves as the communication path between the simulator and the real-time world. Using digital-to-analog and analog-to-digital converters, it allows the simulation to be coupled to external analog components such as controllers, actuators, and display devices. Data to and from the real-time interface are transferred from and to the simulation channels via the real-time information bus.

The simulation channels are programmed to execute the user's simulation. For noninteractive simulations,-those that do not require communication to or from the FEP during execution,-both buses can be used for real-time, interchannel communications. However, for interactive simulation, the interactive information bus might be tied up servicing user requests and might not be available for real-time data transfer when required. RTMPL allows the user to assign data communication paths to meet specific simulation requirements.

Each simulation channel, in the general configuration, consists of two processors: a computation processor (COMP) tied directly to the interactive bus, and a preprocessor (PREP) tied directly to the real-time bus. These processors communicate through shared memory. The general configuration allows a simulation program to be segmented into 1 to 2N parts, where N is the number of available simulation channels. One of these channels can be assigned to perform real-time functions necessary to support the actual real-time simulation. This channel is designated as the "RTX" (real-time extension of the FEP). Depending on the specific implementation of the general configuration, the RTX may have to perform functions such as control of the real-time information bus, sequencing and control of simulation execution, and support of RTMPOS functions. The user should be aware that using the RTX functions available on the target simulator may require significant execution time overhead. This overhead may result in a limit on the time available for executing a user's program on the RTX. The other simulator channels are designated as "DSC" (digital simulation computers). The DSC channels should generally be used for the simulation computations since they require the least overhead. The RTX should be limited to performing nonessential functions (e.g., analytical) since they might have to be sacrificed to execute the simulation within a prescribed update interval.

The general configuration in figure 1 indicates the broad scope of multiprocessor simulators that may be programmed in RTMPL. Any subset of this general configuration may also be programmed in RTMPL. Subsets are obtained by eliminating elements from the general configuration. These subsets therefore include

(1) Single processor
(2) Single channel
(3) Multiprocessor, single bus
(4) Multichannel, single bus
(5) Multiprocessor, dual bus
(6) Multichannel, dual bus
(7) Multiprocessor, shared memory (no data transfer required)

Note that the general configuration assumes no specific hardware for any of its elements. RTMPL is hardware independent.

Targeting

The RTMPL utility contains features that allow the user to target a simulation to a particular simulator configuration, type of computational processor, and simulation purpose. The target configurations are specified in relation to the general configuration and include (1) the number of channels and processors to be used, (2) the location (COMP or PREP) and function (RTX or DSC) of each processor, and (3) the data transfer paths to be used. Processor type is specified by furnishing the utility with information that describes (1) the hardware characteristics of the processor, (2) the assembly language code for RTMPL operations, functions and commands, and (3) the format of the RTMPL utility's output (i.e., assembly language programs) as required for the further development of executable code for the processor. More than one set of assembly language code may exist for a processor. Simulation purpose is specified by selecting the set of codes that best meets this purpose. For example, to verify the execution of the simulation, the user might select a set of operation and function coding that contains calculation overflow tests. After verification, the user would reduce the computation time by selecting a set of operation and function coding without the overflow tests. The following paragraphs describe the methods to be used in supplying targeting information to the utility.

Configuration targeting is done within the RTMPL simulation programs. The utility requires the user to construct an RTMPL program for every processor to be used in the simulation. Each program must be assigned a unique identifier (i.e., RTX, RTXPREP, DSC, or DSCPREP) that indicates the function and location of the processor. (The specific formats for these identifiers are provided in Chapter 9.) These identifiers implicitly define the configuration of the target simulator to the utility. Additionally, an RTMPL construct is available (see the section Variables, Chapter 4) to allow the user to

specify the data path for transferring variables from one processor to another. This implicit definition of the data paths in the target simulator requires that the RTMPL user be familiar with the available hardware and with these data paths.

The user targets the utility to processor type and simulation purpose by specifying a set of target definition files (Control Segment and File, Chapter 3) that govern the translation function of the utility. The RTMPL utility translates RTMPL source programs into assembly language macro statements according to information contained in this set of files. These files describe the target processors, the target assembler, and the assembly language macros to the utility. They are normally developed by systems programmers during installation of the RTMPL utility. More than one set may be available for a particular simulator to allow for optimum code generation for different applications or simulation objectives. The use of the target definition files makes for easy transportation of RTMPL simulations between simulators containing different processors. It also allows a single source program to be translated differently for different purposes. For example, different target definitions may be used to produce both efficient noninteractive code and code that permits maximum interaction of the user with the simulation at run time. Other information contained in the target definition files will be covered in the discussion of RTMPL constructs and utility functions.

Information Transfer

The RTMPL utility translates RTMPL programs into assembly language programs by breaking them down into a sequence of macro operations and arguments. The target assembler then substitutes assembly language code for each macro in the sequence and assembles this code into machine language for execution on the target processors. The utility selects the macro operations from a standard set on the basis of targeting information and program requirements. Arguments are specified to meet program requirements. The assembly language code used for macro substitution is obtained from the target definition files. This code is usually formulated by systems programmers during installation of the RTMPL utility to provide time-efficient execution of the RTMPL macro set on the target processors.

The RTMPL user need not be concerned with most aspects of this translation process. However, knowledge of how the utility mechanizes information transfer may be important since the information transfer may significantly affect the execution time of the simulation.

Three types of information transfer can be mechanized by the utility: control, analytical, and data. Control information regulates the computational sequencing of the simulation computations. Analytical information conveys simulation results to the user. Data information is passed between the various processors as required to compute the simulation. Control and analytical information transfer requirements are specified explicitly by the user in the RTMPL source programs (Commands, Chapter 5). Data information transfer is specified implicitly (Argument Specification, Chapter 4).

At this point a description of the data transfer mechanism employed by the RTMPL utility is appropriate. Figure 2 shows the data paths in a single channel of the general configuration. One segment of shared memory is shown and it contains the channel's transfer and external variables. A transfer variable is one whose value is computed in one processor in a channel and referenced either by the other processor in the channel or by a processor in another channel. An external variable is one whose value is used in a processor but is computed in another processor. Data information transfer between processors is implemented on the PREP and the COMP by using the macros shown above and below the segment of shared memory in figure 2. Their functions are defined as follows:

STX$	stores value of a transfer variable into shared memory and sends value to other channels via bus external to processor in which value is computed
STV$	as above, but sends value via bus local to processor in which value is computed
TSTXVX$	tests currency of value of an external variable sent via bus external to receiving processor; repeated until value becomes current

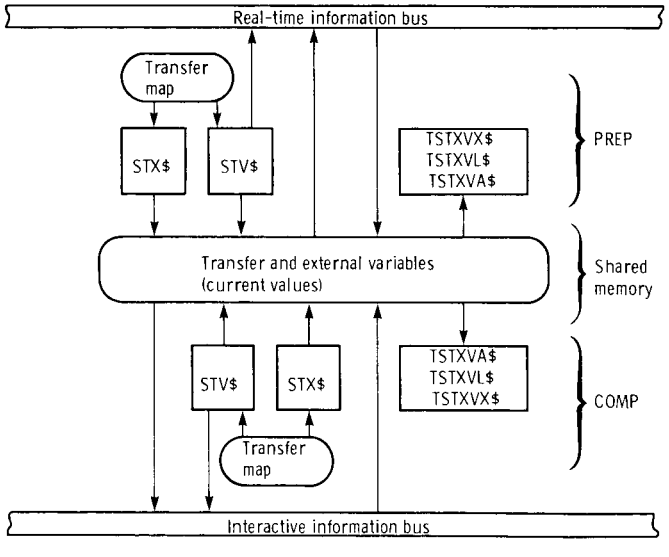


Figure 2. - Information transfer.

TSTXVL$ as above, but used for values sent via bus local to receiving processor

TSTXVA$ as above, but used for values sent via shared memory from alternative processor in channel

The local bus for a PREP is the real-time information bus. The interactive information bus is local to a COMP.

When the RTMPL utility encounters a reference to the current value of an external variable in one of the user's programs (Argument Specification, Chapter 4), it inserts either a STX$ or STV$ macro directly after the macro sequence used to compute the value in the source program. It also inserts either a TSTXVX$, TSTXVL$, or TSTXVA$ macro before the reference in the receiving program. The exact selection of these macros is made according to the location of the source and receiving processors in the general configuration and the data path specified for the variable by the user (Variables, Chapter 4). For example, if a variable were to be transferred from a COMP to another channel via the real-time information bus, STX$ would be inserted after the variable was computed on the COMP. If it were to be received by a PREP in the other channel, TSTXVL$ would be used to test currency. If it were to be received only by a COMP, TSTXVX$ would be used. If it were to be received only by the PREP in the same channel as the source processor, TSTXVA$ would be used in the PREP.

To accommodate transfers of variables to multiple destinations, the RTMPL utility generates a transfer map for each transfer variable. When a variable is selected for transfer, it is assigned a location in channel-shared transfer memory. This becomes a global assignment for all channels used in the simulation. Therefore this is the destination location (or external variable location) for that variable in all channels receiving the transferred value. The transfer map for a variable consists of a list of channels that reference the variable externally. The STX$/STV$ macros consult this map to implement transfers from a specified location in the local-shared transfer memory to identical locations in the memory of the mapped channels. If a variable is externally referenced only on the alternate processor in the local channel, the transfer map for this variable contains no entries. Similarly in a multiprocessor configuration communicating only by shared memory the STX$/STV$ macros would not be required to consult the transfer map at all.

The exact functions of the data information transfer macros and their use of the transfer maps depend on the specific configuration of the target simulation. Their general functions, as described, permit data transfer to be implemented for any subset of the general configuration. This is done during generation of the target definition files. Again, the RTMPL user need not be concerned with the specifics as to how data are transferred. It is important, however, that the user realize that the time required to transmit and receive these data depends on the data path and specific configuration of the simulator. Proper structuring of the simulation to minimize these times may make the difference in realizing real-time execution.

Past-Value Control

Dynamics are incorporated into simulations by manipulating the past values of variables. Integration algorithms, for example, require the retention of one or more past values of a variable. The RTMPL utility automates the retention of past values.

Figure 3 illustrates the memory configuration and macros used to control past values in a single channel of the general configuration. Local memory is shown on each processor in the channel. All variables whose values are calculated on a processor have local memory assigned to them to store their current value and all required past values (Variables, Chapter 4). After the macros that calculate a variable's current value the RTMPL utility inserts SVL$ macros, which roll the past values down one calculation interval. That is,

$$\text{VALUE}\ (T-i) \rightarrow \text{VALUE}\ (T-i-1)$$

where T represents the current calculation and i goes from 1 to the number of past values to be retained. The oldest past value is discarded. The SLV$ macro is then inserted to store the current value in VALUE(T). It is at this point that the RTMPL utility would insert the data transfer macros STX$/STV$ if external transfer of the variable's value were necessary.

The RTMPL utility also automates the retention of a single past value of an external variable. As indicated in the section INFORMATION TRANSFER, referencing the current value of an external variable causes the TSTXVX$/TSTXVL$/TSTXVA$ macros to be used to test the currency of the value on the receiving processor. If only the past value of an external variable is referenced, these macros are not used since currency is not a consideration. Certain actions are necessary, however, to accommodate the retention of the past value.

Two segments of shared memory are shown in figure 3. One segment is used to receive the current values of external variables as described in figure 2. The second segment is identical to the first but is used to retain the past values of the external variables. The XFERSV$ macro is used to test the currency of values of external variables that are referenced only in terms of past values and to transfer these current values into the past-value segment of shared memory for use in the next calculation interval. The XFERSV$ macros are inserted after all

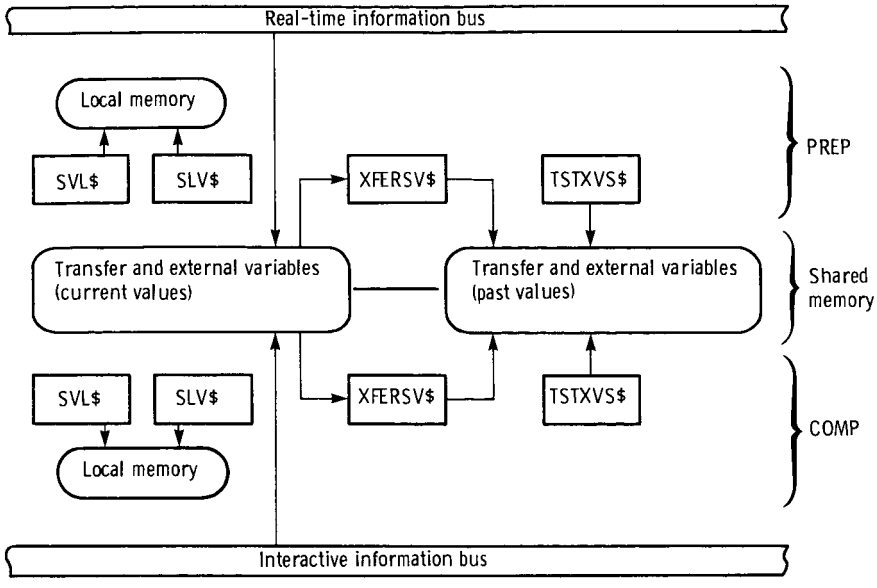


Figure 3. - RTMPS past-value control.

other calculations are completed. On a PREP this macro operates only on values transferred on the real-time information bus, and on a COMP it operates only on values transferred on the interactive information bus.

The TSTXVS$ macro is inserted into the calculation sequence prior to any use of the referenced external past value. Its function is to implement the access of the external past value to local processor computation. Unlike the TSTXVX$/TSTXVL$/TSTXVA$ macros currency testing is not required before implementing this access.

Note that the precise functions of the data-transfer and past-value control macros depend on the target simulator configuration. The preceding descriptions apply to their functions in the general multiprocessor configuration. Although these functions are required in any configuration, they may be performed in whole or in part by the target simulator's hardware. By allowing these macros to be structured to suit the target hardware, RTMPL can be applied to the various subsets of the general configuration.

Chapter 2: RTMPL Environment

The RTMPL utility functions under a disk operating system (DOS). It was initially implemented under Motorola's VERSAdos on an EXORmacs development system. The utility is specific to a particular DOS only in the file identification format. In this manual, files are identified by using the following VERSAdos format:

VOLUME:USERNUMBER.CATALOG.
FILENAME.EXTENSION

where the field widths (i.e., maximum number of characters) are

(4):(4).(8).(8).(2)

The user should become familiar with the file identification format required by the DOS used in the specific installation of RTMPL.

The RTMPL utility processes and produces files of information. It operates in conjunction with other DOS utilities to develop user simulations. This section places the RTMPL utility in context with the overall simulation effort and familiarizes the user with its major functions.

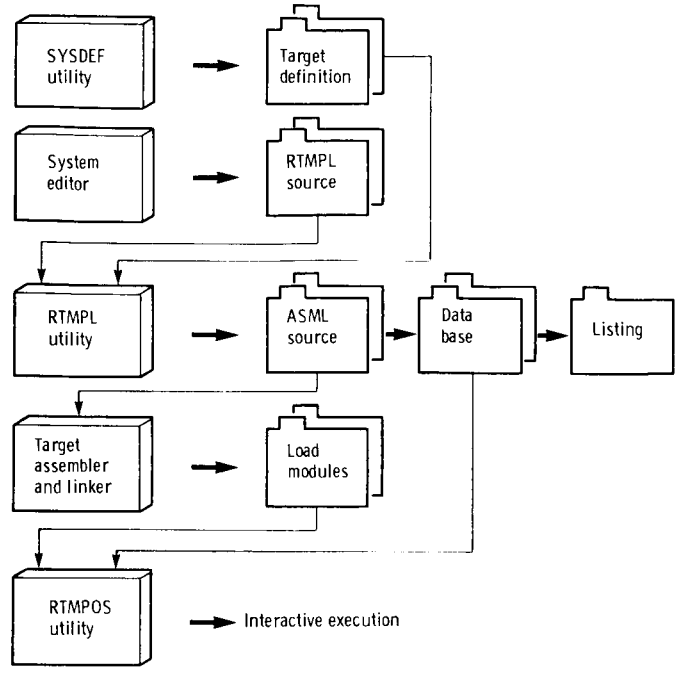


Figure 4. - RTMPS software utilities.

Simulation Development System

Figure 4 shows the RTMPS software utilities used in the development and execution of real-time multiprocessor simulations. The SYSDEF utility is used to generate target definition files. This is normally done by qualified systems programmers and need not concern the general user. Simulation development begins with the DOS editor, which is used to develop RTMPL source files. These files define the simulation problem and contain programs for each simulator processor to be used in its execution. The RTMPL utility translates the RTMPL source into assembly language source files according to information contained in the target definition files. The target assembler and linker utilities are used to create load modules for execution on the target processors. The RTMPL utility also produces simulation-descriptive data-base files. The RTMPOS utility accepts the load modules and loads them into the simulator at run time. Also at run time RTMPOS reads the data-base files to establish a simulation data base that will allow engineering-level interactive execution of the simulation. In addition to the object files (assembler source and data base) the RTMPL utility produces an extensive listing file that provides messages and source interpretations to aid the user in developing error-free, time-efficient simulations. The RTMPL source, object, and listing files are discussed more completely in later sections of this manual.

Source Translation

The RTMPL utility is, in effect, an assembly language programmer. From the simulation description supplied in

the source files the utility develops assembly language programs according to information supplied in the specified target definition files. Although the translation process is essentially transparent to the user, the major functions of the utility in performing this translation are described here to provide a background for the source and object file descriptions that follow.

The utility parses each executable source statement into a list of operations and associated arguments. While doing this, it tests each statement syntactically for correctness. (Parsing is the breaking down of complex statements into an ordered sequence of primative operations.) It also tests each statement semantically against source argument definitions. Arguments are defined in terms of data type and precision. RTMPL supports the Boolean data type and three arithmetic data types (integer, scaled fraction, and floating point), as well as three arithmetic precisions (single, double, and triple). The required data type of the operation is determined from the data type of the statement result. As part of the semantics test the utility compares the required data type with the data type associated with the arguments of the operation.

If the statement is found to be syntactically and semantically correct, the operation/argument list is translated into a list of assembly language macros. The utility supports three argument sources (register, memory, and immediate). An arithmetic addition operation, for example, could be supported by up to 81 addition macros in the target definition files (considering all possible combinations of data type, precision, and argument sources). Having already determined the required data type, the required precision is determined based on look-aheads and look-backs at the other operations in the list. The minimum precision necessary to provide the required accuracy of the statement result is selected as the desired precision of the macro. The target definition files are consulted to see if this precision is supported. If it is not, the next best macro is determined. The user is advised if accuracy will be impaired because the proper macro is not available. The required precision of the arguments is obtained from the target information once the precision of the macro has been determined. Precision conversion macros are inserted automatically by the utility to provide the proper precision of the arguments.

At this point the macro options have been reduced to those that support the required operation, data type, and precision. It now remains for the utility to select the macro, from this set, that best supports the available argument sources. The values of the arguments may reside in memory or in register (if they are the results of previous calculations). If the argument is a constant, it may be expedient to use an immediate data source (where the value exists only within the code of the operation). Desired sources are determined so as to minimize the loading and storing of data from and to memory. The target macro set is consulted to see if the desired sources are supported. If they are not, the best available source is selected and appropriate load and store macros are inserted to conform the arguments to the required sources. The use of scratch pad memory (temporary storage) is fully automated by the utility.

When the macro set for the source statement has been formed, it is edited by the utility (i.e., scaling macros are inserted) if the scaled-fraction data type has been specified by the user. Past-value control and data transfer macros are also inserted as required.

The user is advised of all precision and scale factor adjustments by means of warnings in the listing file. If adjustments to constant arguments are necessary, a new constant with the proper attributes is created by the utility. Definitions of variables are not modified. Using the warnings, however, the user may opt to incorporate the redefinitions in the source programs to eliminate superfluous adjustment macros and thereby reduce simulation execution time.

The final steps in the translation process are the assignment of registers and the generation of assembly language source files.

Chapter 3: Simulation Structure

RTMPL is a structured, high-order language designed to facilitate the development of error-free, time-efficient simulations. The user constructs an RTMPL simulation by creating RTMPL source files as shown in figure 5. The source files define the four segments: control, global data, local data, and execution of an RTMPL simulation. There is one control segment and, at most, one global data segment for each simulation. There is one local data segment and one execution segment for each processor to be used in the simulation. The combination of local data and execution segments for a processor is referred to as a program.

Separate source files are used to contain the control and global data segments. A separate source file is needed for each program. These files are text files, but they must conform to RTMPL constructs. In the following sections the format of RTMPL constructs and the structure of the RTMPL source files are described.

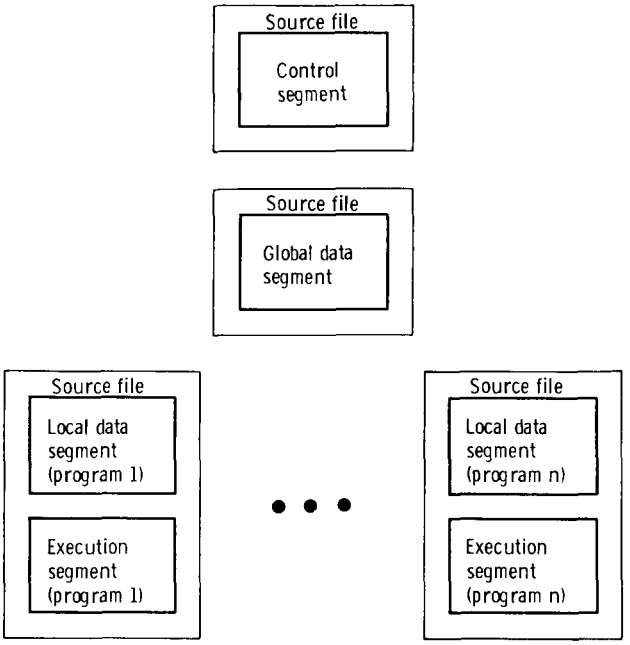


Figure 5. - RTMPL source files.

Syntax Diagram and Basic Constructs

RTMPL constructs can be best illustrated by using syntax diagrams. Syntax diagrams for the basic RTMPL constructs, shown in figure 6, define these constructs and show how to use them in writing source code.

In general a syntax diagram contains one or more programming elements linked together by curved paths that are terminated with arrowheads to show the direction of travel. A rectangular element is used to indicate where a named construct is to be inserted. Parenthetical remarks are used within some syntax diagram rectangles for clarification and generally to denote a special application of the referenced construct. They do not modify the construct in any way. A circular or oval element contains a symbol or string of symbols that must be exactly replicated. The basic construct, NAME (fig. 6(a)), consists of the LETTER construct followed by a series of LETTER or DIGIT constructs (figs. 6(b) and (c), respectively). Note the use of the "..." sequence in these figures to show the inclusions of all symbols from "A" to "Z" and from "0" to "9."

A basic problem with syntax diagrams is in the definition of limits. In the NAME construct the LETTER/DIGIT choice is contained in an infinite loop. In reality this construct is limited to eight symbols. To get around this problem, notes are used in the diagrams to indicate the limitations imposed.

The INTEGER and SIGNED-INTEGER constructs are shown in figures 6(d) and (e). The number of digits allowed in an integer depends on the Pascal compiler used to generate the RTMPL utility. Sufficient digits will generally be available for all applications. There is no need to burden the user with precise specifications on these limits since violations will be rare and flagged as errors by the utility.

The VALUE construct (fig. 6(f)) is used to specify real numbers in RTMPL. A versatile E-format is used. An additional syntax diagram element, the hexagon, is shown in the construct. The hexagon denotes that only one path may follow from it, depending on the value of a

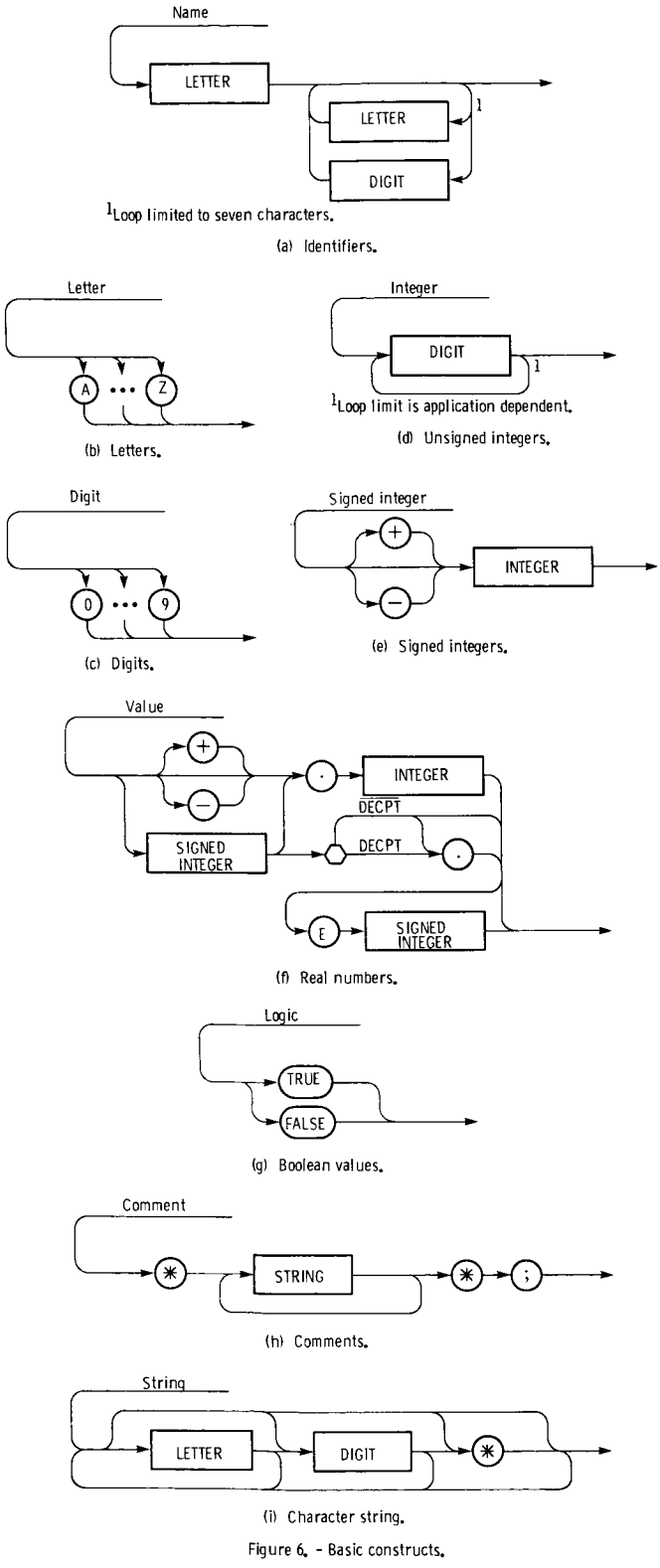


Figure 6. - Basic constructs.

previously specified condition. In this case the condition "DECPT" or "not DECPT" (#DECPT) depends on an option selected during generation of the control file. It is used to specify whether a decimal point is required in the representation of whole numbers (following section). Some examples of real numbers using the VALUE construct are

3479., .3479E+4, 1.7048

or, if #DECPT is set in the control segment,

7048,7048E−4

Boolean values are defined by the LOGIC construct in figure 6(g).

RTMPL allows the user to program in a free form. That is, although the syntax diagrams must be followed exactly, the user is free to insert spaces and line feeds anywhere. This feature allows the user to structure source programs that are personally readable. The semicolon is used in RTMPL to denote the end of a statement or entry. To further enhance readability, the user may insert comments anywhere after a semicolon or at the start of a source file. Comments are structured by using the COMMENT and STRING constructs (figs. 6(h) and (i)). An example of a comment is

*THE*COMPRESSOR*HAS*STALLED*;

Note that asterisks are used in lieu of spaces in strings. All statements, entries, and comments are limited to 3200 nonspace characters (i.e., semicolons may not be separated by more than 3200 nonspace characters).

Control Segment and File

The control segment describes the nature of the simulation to the RTMPL utility and regulates its actions in processing the other segments of the simulation. The file containing this segment has no identification restrictions (as do other RTMPL source files). It is referenced as an argument in the DOS command line that invokes the utility (see Chapter 7). This single-record segment (file) is generated by using the CONTROL construct (fig. 7). The file (fig. 7(a)) consists of up to 11+N entries, where N is the number of simulator channels to be used for the simulation. Each entry must be terminated with a semicolon. An example of a control file for a simulation called T700SIM is shown here.

DECPT,FLOPPY;	(entry 1)
T700SIM;	(entry 2)
TRANSIENT*TEST*CASE;	(entry 3)
FLO*T*BLADE;	(entry 4)
FLOP;	(entry 5)
FLOP;	(entry 6)
FLOP;	(entry 7)
1;	(entry 8)

```
4;                          (entry 9)
RTX.INPRC;                  (entry 9 + 1)
DSC.CMPSIM;                 (entry 9 + 2)
DSC.BNRSIM;                 (entry 9 + 3)
DSC.TRBSIM;                 (entry 9 + 4)
GLOBAL.INT700;              (entry 10 + 4)
I8086.MACHCHAR;             (entry 11 + 4)
```

The first entry contains user-specified options that govern the operation of the utility. One or more options must be specified from the defined set. Multiple options are separated by commas. These options are defined in figure 7(b) and summarized in table I for easy reference. When the utility encounters the NONE option, any previously specified options are ignored, but any options following NONE are enforced.

The FLOPPY option results in the utility pausing prior to accessing an RTMPL source file. The message

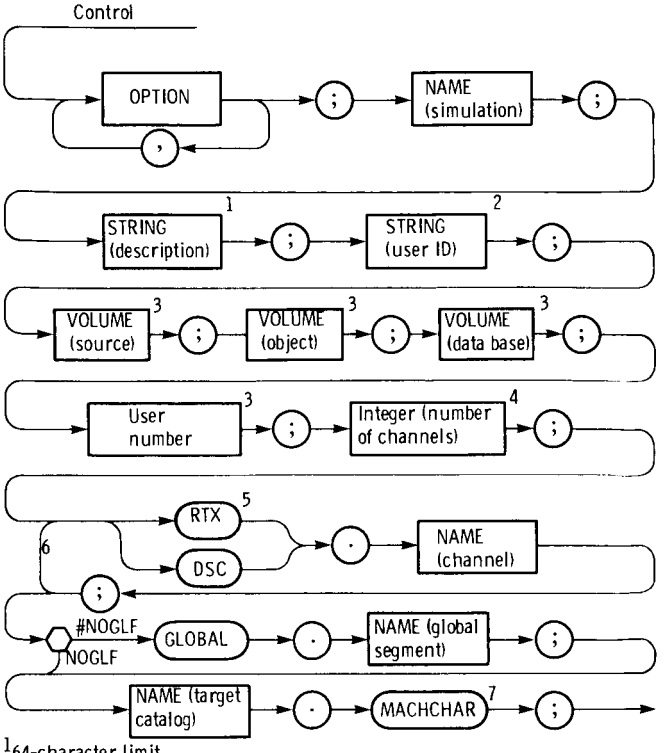
⟨FILE ID⟩ READY? (Y/N)

is displayed and the utility waits for a "Y" response. This option is useful if all RTMPL sources files cannot be contained on a single physical volume such as a floppy disk. Note the use of angular brackets. The convention will be used in this manual to denote user-supplied or, in this case, utility-supplied information of the type specified within the brackets.

The DECPT option requires the utility to insert warnings in the listing file if a decimal point is not contained within an engineering unit value. This option should be used by those wishing to differentiate between real and integer values or those worried about decimal point omissions. The DEBUG option will not normally be selected by the user. It requires the utility to provide functional information in the listing file to verify the validity of the utility's operation. The NOGLF option, if used, advises the utility that the simulation contains no global data file. The global data segment is optional in RTMPL.

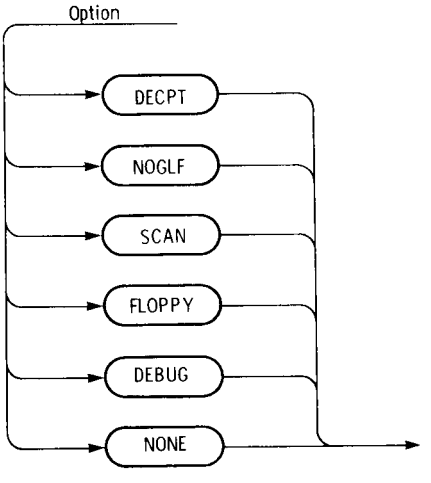
Entry 2 contains a user-assigned simulation name. It is used by the utility whenever reference to the entire simulation is required. It is used in developing listing headers, in certain diagnostic messages, and in file identification for non-program-specific data-base file assignments. Entries 3 and 4 allow further user descriptions of the simulation for use in listing headers. They are limited to 64 and 32 characters, respectively.

Entries 5, 6, and 7 specify logical volume names (designating the disk or medium containing the files) for the RTMPL source, object (assembler source), and data-base files ownership. Entry 8 is the user number for file identification. The constructs for these entries and entry 2 are defined not in RTMPL but by the resident DOS and are installation dependent. The volume names



[1]64-character limit.
[2]32-character limit.
[3]Installation dependent and not defined in RTMPL.
[4]Magnitude limited to number of target channels available.
[5]One RTX channel allowed per simulation.
[6]Loop limited to number of channels specified.
[7]Used to designate machine characteristics target file.

(a) File structure.



(b) Utility operational options.

Figure 7. - Control segment.

and user number are used by the utility in accessing source files and in generating object files.

Entry 9 defines the number of simulator channels (N) to be used in the simulation. Entries 10 through 9 + N

TABLE I.—UTILITY CONTROL OPTIONS

Option	Description	Default
DECPT	Causes a listing file warning if a decimal point is not encountered in a real number	NOT DECPT
NOGLF	Advises that the simulation does not have a global segment	NOT NOGLF
SCAN	Causes the object files not to be generated and the listing file and special macro files to be generated	NOT SCAN
FLOPPY	Causes a pause for disk insertion before files are accessed	NOT FLOPPY
DEBUG	Not for general use; causes expanded listing containing RTMPL diagnostic information	NOT DEBUG
NONE	Causes all previously specified options to be set to default value	----------

contain the channel identifiers in terms of type and logical name. Entry 10+N must be included if the NOGLF option was not selected. Entry 11+N specifies the target simulator characteristics and forms the basis for the utility's referencing of all target definition files.

Program Files

A program file contains the local data and execution segments for each processor to be used in the simulation. The program file for a VERSAdos installation must be named as follows:

VOLUME ID — must be consistent with that specified in control file (entry 5)

USER NUM — must be consistent with that specified in control file (entry 8)

CATALOG ID — must be "RTX" or "DSC" to identify program function. (If program is for a PREP, "PREP" must be appended (e.g., DSCPREP).)

FILE NAME — must be logical name assigned to a channel in control file (entries 9 through 8+N)

EXTENSION — "SA," indicating a text file

Examples of program file names based on the previous control file examples are

FLOP:1.RTXPREP.INPRC.SA
FLOP:1.RTX.INPRC.SA

FLOP:1.DSCPREP.CMPSIM.SA
FLOP:1.DSC.CMPSIM.SA

FLOP:1.DSCPREP.BNRSIM.SA
FLOP:1.DSC.BNRSIM.SA

FLOP:1.DSCPREP.TRBSIM.SA
FLOP:1.DSC.TRBSIM.SA

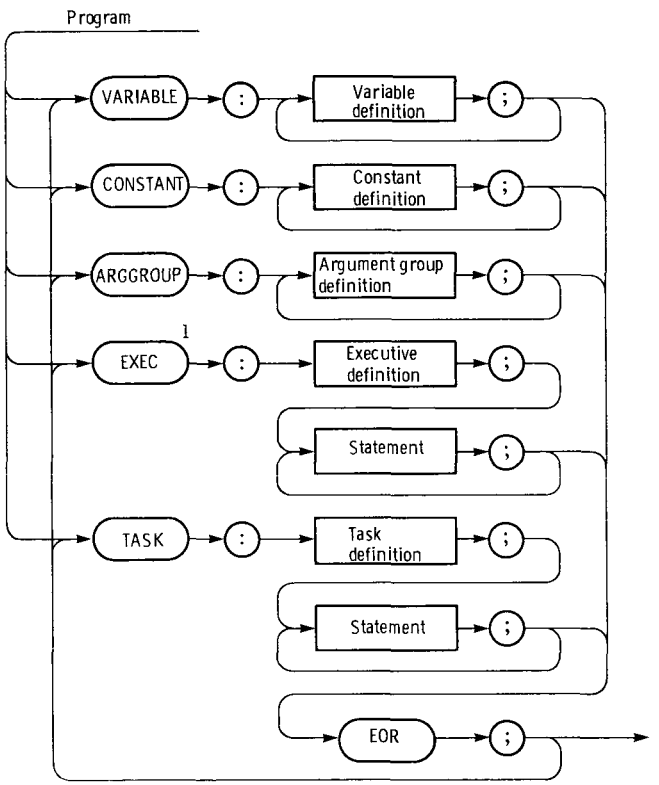


Figure 8. - Program file.

The program file construct is shown in figure 8. Each program file is made up of records. There are five types of record, denoted by the record identifiers VARIABLE, CONSTANT, ARGGROUP, EXEC, and TASK. The identifier is separated from the record content by the colon character. Records are terminated by the end-of-record statement "EOR;". Record types may appear more than once in a file and their order in the file is up to the user. All statements and definitions within the records must be terminated with a semicolon.

The EXEC and TASK records define the execution segment of a processor program. VARIABLE, CONSTANT, and ARGGROUP records define the local data segments for the EXEC and TASK records. Constructs for these records are described in the sections of this manual that discuss the segments. At least one EXEC record is required in each program. The other records are used as necessary to describe the program function. Program files containing the various record types will be shown in detail for the example problem.

Global Data Segment and File

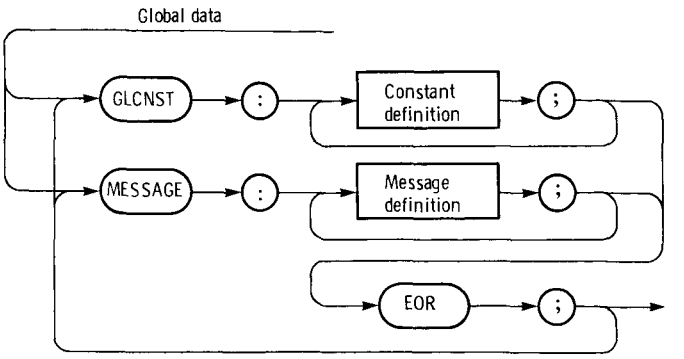
The global data file contains data that may be referenced as execution segment arguments in any or all of the program files. The file for a VERSAdos installation must be named as follows:

VOLUME ID	must be consistent with that specified in control file (entry 5)
USER NUM	must be consistent with that specified in control file (entry 8)
CATALOG ID	must be "GLOBAL"
FILE NAME	user selected
EXTENSION	"SA," indicating a text file

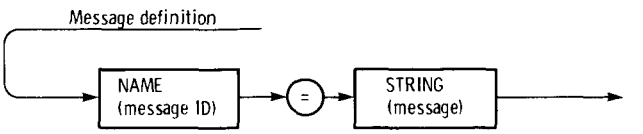
An example of a name for the global data file corresponding to the control file example is

FLOP:1.GLOBAL.INT700.SA

The construct to be followed in generating the global data file is shown in figure 9(a). The file consists of records. There are two different record types, denoted by the



(a) Global file format.



(b) Messages.

Figure 9. - Global data file.

identifiers MESSAGE and GLCNST. The record identifier is separated from the record content by the colon character. Records are terminated by the end-of-record statment "EOR;". These records are used as required to define the global data segment. Remember, if no data segment is required, the NOGLF option must be selected in the control segment. Records may appear more than once in a file, and their order in the file is up to the user. All definitions within records must be terminated with a semicolon. Constructs for these records and their functions are described in the following section.

Chapter 4: Data Segments

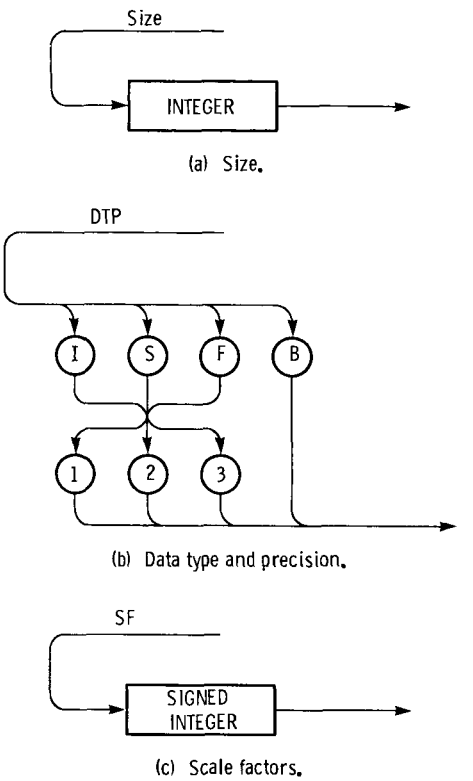
The local and global data segments are used to define simulation variables, constants, argument groups, and run-time messages. These definitions are used to verify semantics and to build assembly language macros when these items are referenced as arguments in the execution segments. Variables are defined as those items that are subject to change as a result of executing the simulation. Constants are those items that do not change as a result of simulation execution. Argument groups are lists of variables and constants. They can be used by RTMPOS for run-time data gathering and display. They may also be used within the simulation to pass arguments and data between the user's programs and target library procedures. Messages are displayed to the user during simulation execution when user-programmed conditions are met.

This section describes the constructs used to define data items. The data attributes are discussed, as are special properties that are useful for simulation.

Data Attributes

All local and global data are assigned names to identify the data item (fig. 6(a)). All names within a local data segment must be unique within the segment. All names of constants within the global data segment must not only be unique within the global data segment but also different from any name assigned in any local data segment in the simulation.

Along with names, RTMPL requires certain other attributes to be specified. The VALUE construct (fig. 6(f)) is used to specify data values. Other attributes are size (SIZE), data type and precision (DTP), and scale factor (SF). Constructs for the specification of these attributes are given in figure 10. SIZE (fig. 10(a)) is used to define the number of elements associated with a data name. Its meaning is data-construct dependent and is described in the discussion of these constructs. DTP (fig. 10(b)) is used to specify the data type and precision of constants, variables, and argument groups. RTMPL allows assignments of four data types:



(a) Size.

(b) Data type and precision.

(c) Scale factors.

Figure 10. - Special data attributes.

(1) Integer (I): integers
(2) Scaled fraction (S): real numbers
(3) Floating point (F): real numbers
(4) Boolean (B): logical (true or false)

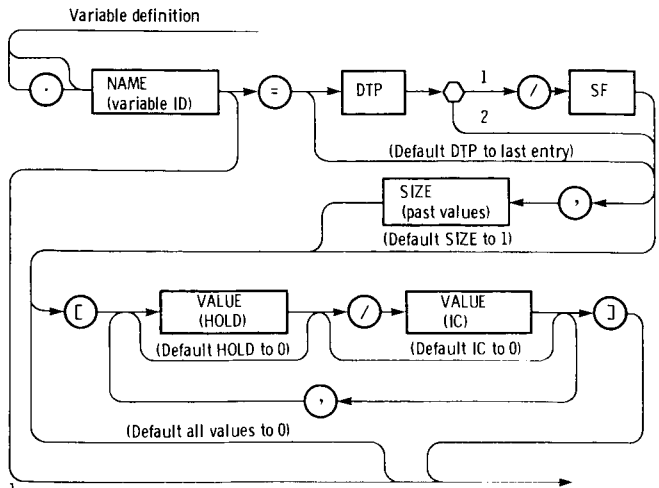
The arithemetic data types—I, S, and F—must be assigned a precision. RTMPL supports single-, double-, and triple-precision data of these types (1, 2, and 3). The selected precision dictates the number of bytes used in the target processor to represent the data and is directly proportional to accuracy and generally inversely proportional to computational speed. Usually single-precision integer data provide two bytes of accuracy, with another two bytes of accuracy added for each increase in precision. The exact implementation of precision by the target processor is specified in the target definition files.

The choice of data type assigned to integer or logical variables and constants is obvious. However, in assigning real variables and constants, the user must decide between the S and F data types. This choice is dictated primarily by the computational speed requirements of the simulation and the computational speed capability of the target processor in performing operations on the data types. Generally scaled-fraction computations are faster but are slightly more difficult to generate since they require scaling of all variables and constants. Since scaled-fraction values must fall between -1 and $+1$, the maximum absolute value of each variable or constant must be determined and specified in its RTMPL definition. RTMPL uses binary scaling. The maximum absolute value must therefore be specified in terms of the minimum power of 2 that exceeds the maximum absolute value. The specification is made by using the construct in figure 10(c). For example, if the maximum absolute value is 31.9, SF becomes 5 ($2^5 = 32$); if the value is 0.1239, SF becomes -3 ($2^{-3} = 0.125$).

While processing equations involving scaled fractions, the RTMPL utility will perform all necessary scale factor manipulations, thereby relieving the user of this chore. The user-assigned scale factor (SF) is referred to as the "nominal" scale factor of a variable or constant. When a operation is performed on the variable or constant, a "required" scale factor is then determined by the utility on the basis of subsequent operations (required to compute the equation) and information concerning the operations obtained from the target definition files. The difference between the "nominal" and "required" scale factors is reconciled by inserting a scaling macro (shift operation) before or after the operation as appropriate. The result of the operation is assigned the "nominal" scale factor for use when the result is an argument of a subsequent operation. The utility will list recommended adjustments in SF or precision specifications that will minimize the computation time (see Chapter 8). Through these and other advisories the user can adjust the variable and constant definitions to eliminate time-consuming scale factor adjustments and potential overflow or underflow problems.

Variables

Variables are defined by using the construct in figure 11. Each variable is assigned a name and set of attributes. DTP and SF were described previously. The SIZE attribute, in this case, determines how many current and past values of a variable are to be kept. For example, if the simulation requires the current and last value of a variable (e.g., if a second-order integration scheme is used), its size would be specified as 2. The minimum variable size is 1 (i.e., only the current value is saved to provide one past value). The RTMPL utility provides for



[1]This path taken if variable defined as scaled fraction.
[2]This path taken if variable not defined as scaled fraction.

Figure 11. - Variable definition.

automatically adjusting the variable's past-value array after its solution in an equivalence statement. All variables must appear on the left of an equivalence statement.

Two values must be assigned to each variable in the variable definition. The hold value is reserved for special operating system (RTMPOS) applications. The initial-condition (IC) value is the starting value of the variable when the program is loaded and whenever the RTMPOS IC mode command is executed (ref. 4). Both are specified in engineering units even if the variable is designated as a scaled fraction.

Note the various default paths through the VARIABLE DEFINITION construct. All or any attributes of a variable may be defaulted (as long as the required default values exist). DTP and SF may be defaulted to those attributes of the last defined variable.

Consider the following examples of variable definitions based on the figure 11 construct:

(1) Variable SPEED is to have the value 5000 rpm in hold and IC. It is a single-precision, scaled-fraction variable and is to be transferred on the interactive bus. Since scale factors $2^{12} = 4096$ and $2^{13} = 8196$, speed would be defined as

```
.SPEED = S1/13, 1 [5000./5000.];
```

or, using defaults,

```
.SPEED = S1/13 [5000./5000];
```

(2) Variable SPEEDOT is to have the value 0.0 in hold and IC. It is a single-precision, scaled-fraction variable and is to be transferred on the real-time bus. Two past values are required for the numerical integration scheme. Therefore SPEEDOT would be defined as

SPEEDOT = S1/10, 2 [0.0/0.0];

or, using defaults,

SPEEDOT = S1/10, 2;

(3) Variable READOT is to have the same attributes as SPEEDOT. Therefore all defaults are used to define READOT after the definition of SPEEDOT.

SPEEDOT = S1/10, 3; READOT;

Two types of processor memory are reserved for the variable values—local memory and transfer memory (fig. 3). Variable values can only be transferred to other programs if they are stored in transfer memory. The RTMPL utility will automatically assign transfer memory if a variable is referenced in another program. Otherwise the variable is assigned to local memory. The user specifies the transfer path for a variable by inserting or omitting a period before the name specification. Omission causes real-time bus transfer; insertion forces interactive bus transfer.

Certain variables are implicitly defined for every program by the target definition files. Called target Boolean variables, their values are generated by the processor's hardware and firmware. They are always local and may not be referenced directly in another program. They are always Boolean (i.e., DTP = B). Examples of target Boolean variables are OVERFLOW, POSITIVE, ZERO, and NEGATIVE, which may be generated by the processor in its status register. Since these variables are processor dependent, the user must refer to the system description of the target definition file to see what variables are available.

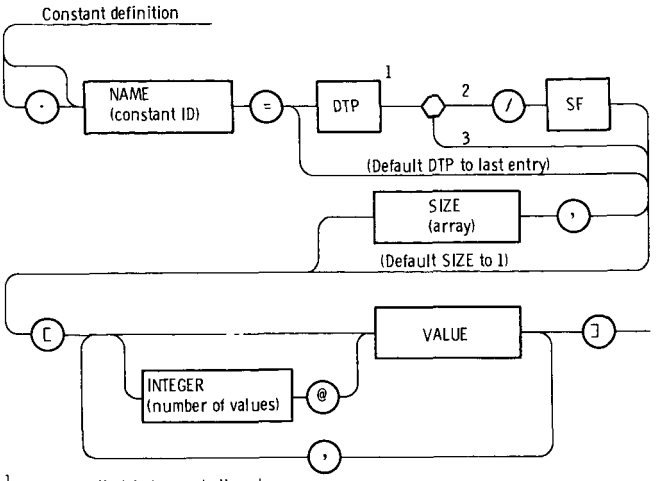
Constants

Constants are defined by using the construct in figure 12. As with variables, constants have DPT, SF, and SIZE attributes. SIZE for constants specifies the number of elements in a multivalued constant array. The minimum constant array size is 1. The construct definition allows a string of N identical values to be extended as N @ value. Therefore three sequential values of 1.25 could be entered as 3 @ 1.25. The number of values entered must correspond to the specified size. No value defaults exist.

RTMPL allows the use of four types of constant:

(1) Local constants
(2) Local parameters
(3) Global constants
(4) Global parameters

Parameters are constants that are adjustable through RTMPOS at run time. They are specified during the



[1]Boolean ("B") data type not allowed.
[2]This path taken if constant defined as scaled fraction.
[3]This path taken if constant not defined as scaled fraction.

Figure 12. - Constant definition (local or global).

constant definition by preceding the constant name with a period. Local constants and parameters are those defined within the program file. Global constants and parameters are those defined within the global definition file, and their use is described in the discussion of that file.

RTMPL does not allow user definition of Boolean constants. The Boolean constants TRUE and FALSE are predefined by the utility and available implicitly to the user.

Global Constants

The GLCNST records in the global data file are used to specify constants and parameters that are global to the simulation. The record content is specified by using the construct of figure 12. Global constants may be referenced as arguments in any program, without being defined in that program. Upon being referenced, the global constant definition is copied into the program. Apart from the advantage that global constant records relieve the user of the task of defining the same constant in a multitude of programs, the global constant has a special meaning to the operating system. If a global constant is defined as a parameter, modifying its value at run time, by using RTMPOS, will cause it to change globally throughout the simulation.

Some examples of constants (both local and global) are

(1) .PI = 3.1416 (scaled to 4; single precision; parameter; single value)
.PI = S1/2, 1 [3.1416];
or
.PI = S1/2 [3.1416]; (using the SIZE default)

(2) K1 = 1 (integer; double precision; not parameter; single value)
K1 = I2[1];
(3) PRXVALS = 1.25,1.25,1.25,4.0,5.0 (floating point; single precision; not parameter; five values)
PRXVALS = F1, 5 [3@1.25, 4.0, 5.];

Messages

Messages can be relayed to the user at run time via the ADVISE command. These messages are defined in the global data file. Figure 9(b) defines the MESSAGE DEFINITION construct. A message is assigned a name, which is used to reference the message in an advisory. A message containing up to 64 characters will be displayed on the user's terminal upon execution of the advisory. A message is global and may be referenced in any program. Note that spaces are ignored in the message format, but asterisks are interpreted as spaces.

Argument Specification

To understand argument groups, it will help to understand how arguments are defined. When a variable or constant is referenced in a statement as an operand or in an argument group, it is called an argument. RTMPL allows any variable or constant, defined within the scope of the simulation (including global constants), to be referenced as an argument. The ARGUMENT construct is defined in figure 13.

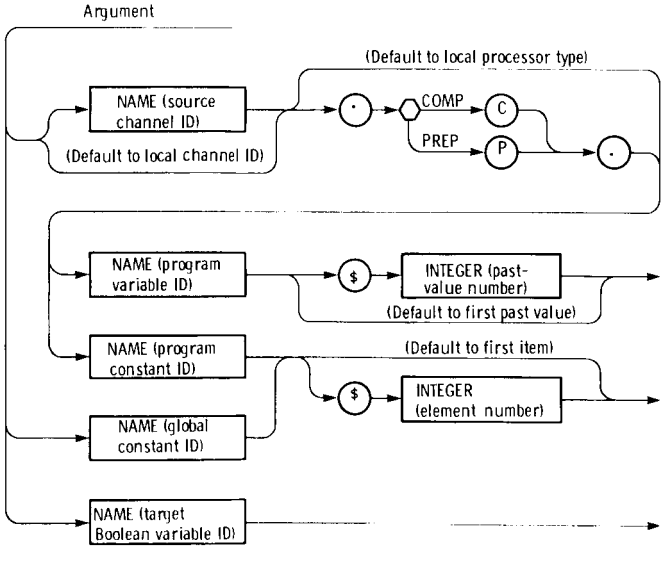


Figure 13. - Argument specifications.

Global constants are referenced only by name. Program-defined constants and variables can have their source program file explicitly specified. This is done by specifying the source channel name and processor type. Thus

TURBINE.C.FLOW

specifies the variable FLOW defined in the COMP program in channel TURBINE. Examples of other source specifications are

FLOW	source of variable FLOW is local program file
.P.FLOW	source of variable FLOW is PREP program in logical channel assigned to local program
TURBINE.FLOW	source of variable FLOW is program for local processor type assigned to logical channel TURBINE

If a constant is specified as an external argument (defined in another program) in the local program and has not been defined in the local program, the RTMPL utility will create a local constant with the name and attributes of the argument specification. This constant will be used as the argument. However, if a constant of the same name has been defined locally, a program error will result. This mechanism can be used to identify identical constants in different programs that should be handled via global constants.

If a variable not defined in the local program (but defined in another program) is specified as an argument, the utility will create an external variable in the local program and assign it a location in external memory. Its value will be transferred to the local program after it has been computed in the external program. The external variable will then be used as the argument (Information Transfer, Chapter 1).

The ARGUMENT construct provides a means for specifying a particular past value of a variable or a particular element within a multivalued constant array. This is done by appending $n to the name, where n is the variable past-value number or constant-element number. For example,

FLOW $2 (second past value of flow)

TABLE $10 (tenth element in constant array TABLE)

Note that the term "past value" refers to the results of computations. When a variable is recomputed, the old

value is assigned as the first past value of the variable. The RTMPL utility will issue a warning if a local variable is referenced as an argument before it has been assigned a value through computation.

Argument Groups

An argument group is a set of arguments grouped together under a single name. It can be used to pass arguments to or from target library procedures (see the CALL command). Argument groups also provide for large-volume data transfers between the FEP and the simulator channels. Argument groups are specified in a program file by using the ARGGROUP construct defined in figure 14.

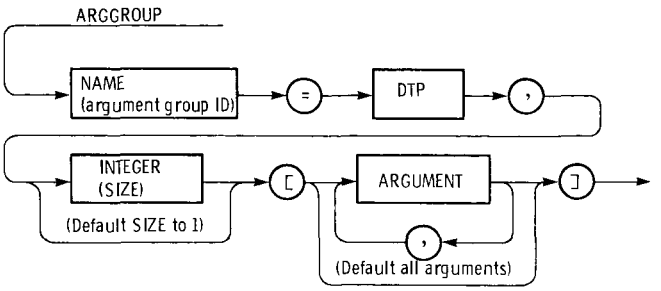


Figure 14. - Argument group definition.

ARGGROUP attributes consist of a data type and precision (DTP) assigned to the group, the maximum number of arguments to be contained in the group (SIZE), and an optional initial set of arguments. Argument groups can be edited at run time by using RTMPOS. Therefore arguments need not be specified in the program. However, since only variables and constants that are available on the argument group's processor can be inserted at run time into the argument group, the user should include, in the ARGGROUP specification, any external variables and constants that may eventually be required in the ARGGROUP. This will allow the RTMPL utility to form these constants and external variables.

All arguments within an argument group must conform to the specified DTP for that group. For examples of argument groups and their uses, see Chapter 6.

Chapter 5: Execution Segment

The execution segment of a program (fig. 15) is made up of two types of records—executives and tasks. Executives are used to provide program control and to perform major simulation functions. Tasks are used to perform services for executives. Two types of executives may be specified by the user: background and foreground (see the section Executives, Chapter 5). Executive and task records are made up of statements that define the required simulation execution.

This chapter discusses the definition and use of executives and tasks. The user should refer to figure 15 in these discussions. The variations of the STATEMENT construct and the use of the EXPRESSION construct are explained.

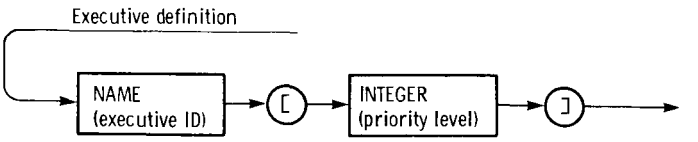


Figure 16. - Executive record specification.

Executives

An executive is defined by using the construct in figure 16. It is assigned a name and a priority level. The first seven characters of the name must be unique within the set of program executive and task names. The priority level is an integer, limited in magnitude to 8, that

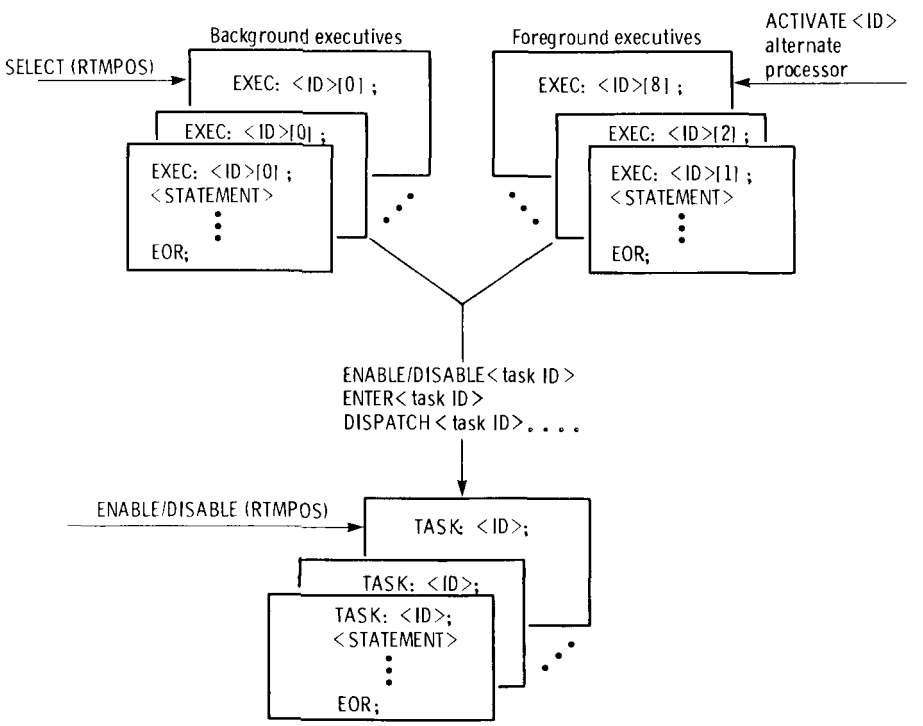


Figure 15. - RTMPL source records - execution segment.

specifies the computational priority of the executive within the program. Lower priority executives may be interrupted for execution of higher priority executives. Priority assignments greater than zero must be unique within the program.

The RTMPL utility modifies the user-specified name, to ensure its uniqueness. (Future versions of the utility will eliminate this annoyance.) A special assembler character (".") is inserted following the name, if the name is seven characters or less. If an eight-character name is specified, the last character is replaced with the special assembler character (e.g., "TURBINES" would become "TURBINE."). (Two special assembler characters are defined in the target definition files. These are used whenever the utility must generate a name. In this manual the period and dollar sign are used.)

Executive execution is governed by two processor firmware programs: the sequencer and the channel interrupter (ref. 1). The sequencer is controlled by the FEP with information provided by the user at run time through RTMPOS. The channel interrupter firmware services user-programmed interrupts between COMP and PREP processors in the same channel.

The user may specify background executives at run time that will control the execution of the simulation programs through the sequencer. RTMPL requires at least one background executive in each program, but more than one is permitted. By using multiple background executives the user may change the simulation subtly or completely at run time (i.e., more than one simulation may be programmed within a single set of RTMPL source files). All background executives must have priority levels of zero.

Executives that are assigned priority levels greater than zero are considered to be foreground executives. Their execution is governed by the local processor's channel interrupter firmware. This firmware functions during program execution and allows user-programmed interrupts between COMP and PREP processors in the same channel. Obviously foreground executives may only be implemented on simulators having both COMP and PREP programs in a channel.

Typically the function of a foreground executive in a particular program will be to service exceptions occurring in the alternate processor in the channel. These exceptions would be generated in the alternate processor by using the ACTIVATE command (see the section Commands) and would be the result of conditional testing in that processor's program. This eliminates duplication of code and data transfer when both processors must respond to the same event. Any number of foreground executives may be activated simultaneously, with the order of execution controlled by the channel interrupter according to predefined priority levels. Upon completion of all activated foreground executives, program control returns to the background executive. If a foreground executive is reactivated while it is still active, the second activation request is ignored.

Tasks

Tasks are defined by using the construct in figure 17. The construct consists merely of identifying the task by name. The first seven characters of the name must be unique within the set of executive and task names used in the program. As with the executive definition the task name is modified by using the special assembler character ".".

Tasks consist of statements structured to do a particular job within a program. They are reenterable and therefore may be initiated in any background or foreground executive (see ENTER and DISPATCH commands). A task can never be initiated by another task.

To enhance their flexibility, tasks may be enabled or disabled at run time by the user through RTMPOS. They may also be enabled or disabled by any executive or task in the program (see ENABLE and DISABLE commands). A task may disable itself. If a task is disabled, it cannot be initiated.

Statements

As shown in figure 15 the functions of executives and tasks are defined in terms of statements. The STATEMENT construct is shown in figure 18. The executive and task statements are processed by the RTMPL utility to formulate the executable part of programs. Three types of statement are defined in RTMPL—assignment, conditional, and command. Assignment statements are used to establish values for local variables (those defined in the program containing the statement). Conditional statements are used to test values and to take specified actions depending on the result of the test. Command statements are used to provide simulation control, sequencing, and linkage to

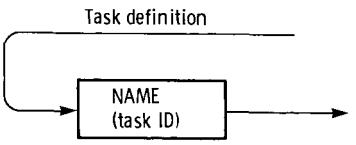


Figure 17. - Task record specification.

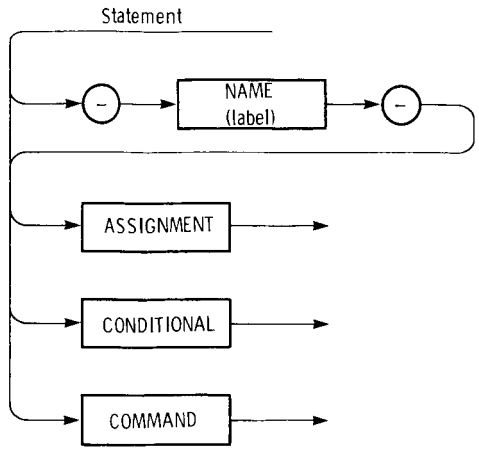


Figure 18. - Statements.

simulator hardware and software components. These statement types are described in detail in the sections Assignments, Conditionals, and Commands.

Statements may be labeled by the user. Labels are names and are enclosed by the underscore character "_". If a statement is not labeled by the user, the utility will supply a label based on the order of the statement in the program. The special assembler character "$" is used to generate this label. For example, the first statement in the program would be labeled by the utility as S$1 if it were not assigned a label by the user. User-assigned labels must be unique names within the program.

Statements are limited to 3200 characters, excluding spaces, returns, and line feeds. These three characters are ignored by the utility, providing flexibility for the user in formatting the source program. The number and complexity of statements are essentially unlimited by the utility. However, they are limited by the amount of user memory available in the host computer and the amount of program memory available in the target computer.

Assignments

The ASSIGNMENT construct is defined in figure 19. The "=" character is used to denote the assignment of the computed value of an expression to a local variable. Although many languages (e.g., Pascal and Ada) denote assignments by the ":=" combination (to differentiate an assignment from an equality), RTMPL depends on the user to properly apply the "=" character. All local variables should appear as the result of assignment. The utility will flag those that do not (Data-Base Files, Chapter 9). Data type (I, S, F, or B) must be maintained within an assignment (see the section Expressions). The expression must result in a value whose data type is the same as that of the local variable.

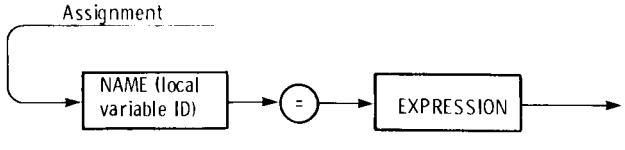


Figure 19. - Value assignment statements.

Conditionals

The CONDITIONAL construct is defined in figure 20. A conditional statement consists of the key word "IF" followed by an underscore character (all RTMPL key words are terminated by an underscore character in source programs), a Boolean expression (true or false value), the key word "THEN" (and its underscore character), a series of statements to be executed if the expression's value is true, and optionally, the key word "ELSE" (and its underscore character) and a series of statements to be executed if the expression's value is false. Note that a Boolean expression may be formed from arithmetic expressions (data type I, S, or F) through conjunction with conditional operators (<, #<, =, #=, >, #>). The conditional operators are defined in table II. For example, for integer expressions A, B, and C, the Boolean expression

$$A<B=C$$

will be true if the value of A is arithmetically less than B and the value of B is equal to the value of C; otherwise the Boolean expression will be false. The Boolean

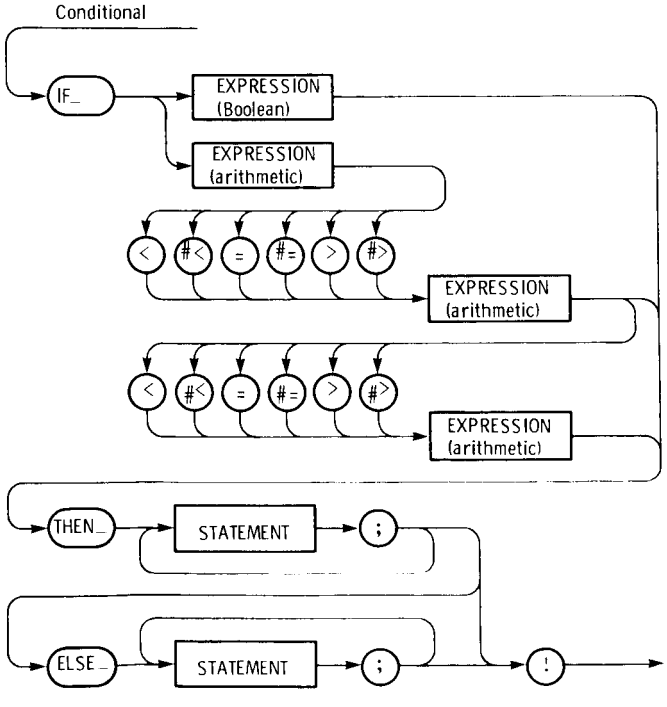


Figure 20. - Conditional statements.

TABLE II.—CONDITIONAL OPERATORS

Conditional operator[a]	RTMPL interpretation
<	Less than
#<	Not less than
=	Equal to
#=	Not equal to
>	Greater than
#>	Not greater than

[a]The "#" character in RTMPL is interpreted as a logical NOT.

expression, including the "IF" key word, has the same length limit as a statement.

The CONDITIONAL construct is terminated with an exclamation point. The conditional statement

```
IF_A THEN_ B=C;
```

is incomplete. The RTMPL utility will expect additional statements in the "then clause" (to be executed if A is true) or an "else clause" (to be executed if A is false). The following statement is a complete conditional:

```
IF_ A THEN_ B=C; !
```

In this case no action is taken if A is false since the "else clause" is omitted.

RTMPL permits nested conditionals. The use of the conditional terminator "!" allows the user to build EXEC and TASK records that have structures similar to the familiar Pascal "begin...end" structure. The structure consists of conditional levels. For example,

```
IF_A              (main stream)
  THEN_...        (level 1)
  IF_B
    THEN_...      (level 2)
    IF_C
      THEN_...    (level 3)
      ELSE_...
!                 (level 3 terminator)
!                 (level 2 terminator)
!                 (level 1 terminator)
```

Three nested conditional levels are shown. The various levels are indented for clarity. The level 1 test will be made if A is true. The level 2 test will be made if B is true. Since the "else clause" occurs before the termination of level 3, it will be assigned to level 3 and executed if A and B are true and if C is false. If the example were rewritten as

```
IF_A              (main stream)
  THEN_...        (level 1)
  IF_B
    THEN_...      (level 2)
    IF_C
      THEN_...    (level 3)
!                 (level 3 terminator)
    ELSE_...      (level 2)
!                 (level 2 terminator)
!                 (level 1 terminator)
```

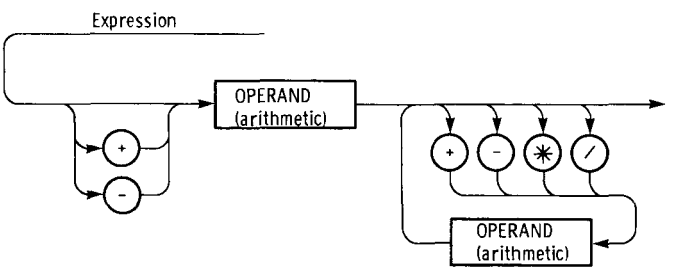
the "else clause" would be executed if A were true and B were false. Remember that the RTMPL format is free. The preceding example could have been written

```
IF_A THEN_... IF_B THEN_... IF_C THEN_...!
ELSE_...!!
```

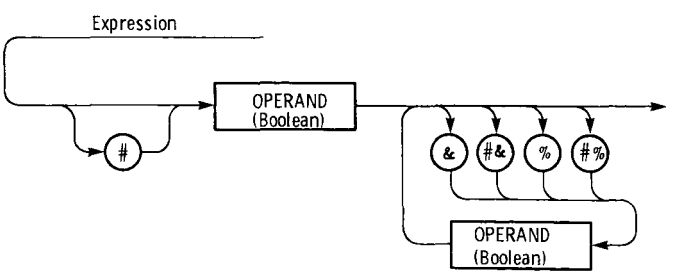
However, the structure is not evident in this form. The RTMPL utility provides a structured listing to aid in program debugging (see Chapter 8).

Expressions

An RTMPL expression is an ordered string of operation/operand pairs that is logically formulated by the user to produce a value. Two types of EXPRESSION constructs are available: the arithmetic expression (fig. 21(a)) and the Boolean expression (fig. 21(b)). They differ in the type of value they produce and in the set of operations available. A Boolean expression can contain only Boolean operands (DTP = B). An arithmetic expression can contain only arithmetic operands of the



(a) Arithmetic expression.



(b) Boolean expression.

Figure 21. - Expression specification.

same data type (integer (I), scaled fraction (S), and floating point (F)).

The data type of the value produced by the expression must be consistent with the data type requirement of the statement containing the expression. The required data type of an assignment statement is always the data type of the local variable receiving the assignment. The required data type of a conditional statement is always Boolean. Arithmetic expressions that are compared, by using conditional operators, must have consistent data types across those operators. For example, an integer value cannot be compared with a scaled-fraction value.

The available RTMPL arithmetic and Boolean operators are given in table III. Both sets contain unary and binary operators. Unary operators have a single operand and must appear only at the beginning of an expression. The "null" unary operator is contained in each set to indicate that a unary operator is not necessary unless a "negate" or "logical not" operation is required. Binary operators have two operands and must always be preceded and followed by an expression.

Each operation is assigned the corresponding parsing value listed in table III. These values are used by the RTMPL utility to establish the order of calculation in the expression. Generally an expression is parsed from right to left. An operator (and associated operands) is placed in the sequence when a subsequent operator has a parsing value not greater than its own. For example, the expression

$(-A * B/C + D - E)$

would be parsed as

$SAVE = D - E$
$RESULT = -A$
$RESULT = RESULT * B$
$RESULT = RESULT/C$
$RESULT = RESULT + SAVE$

This parsing rule should be followed by the user in writing an expression.

Operands for arithmetic and Boolean expressions are defined in figure 22. Four operand types are common to both:

(1) Argument
(2) Multivariable function
(3) Unary function
(4) Parenthetical expression

Boolean expression operands include the implicitly defined Boolean constants TRUE and FALSE and target Boolean variable names.

The basic operand type is the ARGUMENT construct (fig. 13). This allows any constant or variable defined in the simulation to be specified as an operand as long as its data type is consistent with the required data type of the

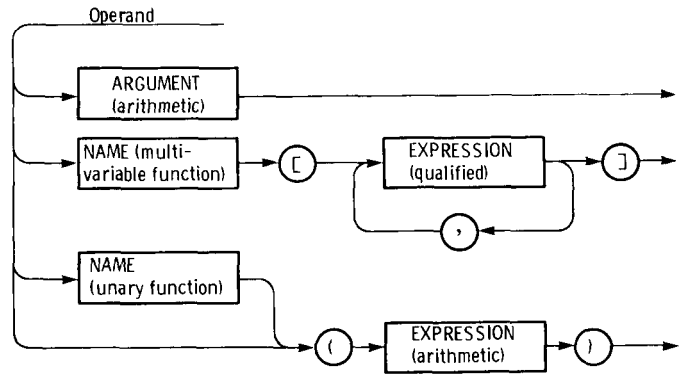
TABLE III.—RTMPL OPERATORS

(a) Arithmetic expression

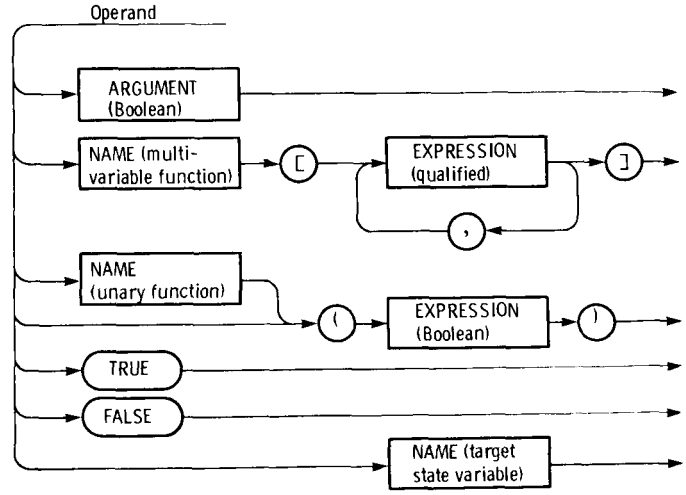
Operator	Type	Interpretation	Parsing value
−	Unary	Negate	0
+	Unary	No operation	0
Null	Unary	No operation	0
+	Binary	Add	1
−	↓	Subtract	2
/	↓	Divide	3
*	↓	Multiply	4

(b) Boolean expression

#	Unary	Logical NOT	0
Null	Unary	No operation	0
$	Binary	Logical AND	1
#$	↓	Logical NAND	↓
%	↓	Logical OR	↓
#%	↓	Logical NOR	↓



(a) Arithmetic expression.



(b) Boolean expression.

Figure 22. - Operand specification.

expression. Specifying an external variable as an operand will cause that variable's value to be transferred to the local program during execution; specification of an external constant as an operand will cause that constant to be defined in the local program file.

An operand may be a parenthetical expression (i.e., an expression enclosed within parentheses). This definition does not preclude nested parentheses. For example, the complex expression

```
(-((A/(B+C)+D)*E))
```

contains the following parenthetical expressions as operands:

```
Operand 1: (B+C)
Operand 2: (A/OPERAND 1+D)
Operand 3: (OPERAND 2*E)
```

Since each parenthetical expression produces a value, the use of parenthetical expressions as operands dictates the parsing sequence of the complex expression. Parenthetical expressions are always parsed from the inside out.

RTMPL supports two types of functional operands: (1) unary functions, which are functions of a single variable (e.g., SINE (ANGLE)) and (2) multivariable functions, which may have eight variables (e.g., INTEGRAL (RESULT, GAIN, DERIVATIVE)). RTMPL does not contain any inherent functions of either type. That is, the specific functions available to the user are those that have been established during the RTMPL utility implementation on the host computer. These functions are implemented as assembly language macros and are defined in the target definition files. The user should become familiar with those functions that are available for the target simulator.

Functions produce values of specified data type. The user must select functions that are compatible with the required data type of the expression. For example, an implementation of the utility might support the sine function for both scaled-fraction and floating-point numbers (named SINSF and SINFP, respectively). Use of SINSF in an expression that is required to produce a floating-point value would be flagged as an error by the utility.

All unary function names must be immediately followed by an expression in parentheses. This expression provides the value of the function argument. The unary function argument must always be of the same data type as the function. For computational sequencing the parsing value of any unary function is assumed by the utility to be zero. The computational sequence of

```
(A+SINE(B+D))
```

would be

```
RESULT=B+D
RESULT=SINE (RESULT)
RESULT=A+RESULT
```

The arguments of multivariable functions are enclosed in brackets and separated by commas. The arguments may be any expression with the following restrictions:

(1) The functional arguments must not contain other multivariable function operands. Therefore multivariable functions may not be nested within expressions.

(2) The data types of the function arguments must correspond to those specified in the target definition of the function. Unlike unary functions the data types of the function arguments have no required relationship to the data type of the expression containing the function.

For computational sequencing the parsing value of any multivariable function is assumed by the utility to be zero. The computational sequence of

```
(A+INTEGRAL [STATE1, K/J, B*C])
```

would be

```
ARG1=B*C
ARG2=K/J
RESULT=INTEGRAL [STATE1, ARG2, ARG1]
RESULT=A+RESULT
```

Note that the value of each functional argument is determined in sequence from right to left.

The RTMPL utility translates operators and functions into target-defined macro operations that are assembly language equivalents to the operator/function for the required data type. If an operator/function does not have a macro equivalent for the required data type, the utility will flag an error in the listing. To produce the desired precision of operator values and to accommodate particular sources of operands (e.g., register and memory), the target definition files can contain more than one macro equivalent for an operator for the required data type. For example, the "+" binary operator for integer data types might be supported by the macro's named

```
ADD$I1RR   ADD$I1RM   ADD$I1RI
ADD$I2RR   ADD$I2RM
ADD$I3RR   ADD$I3RM
```

where ADD$ denotes the operation, I denotes integer data type, the numbers (1,2,3) denote the precision of the result value, and the trailing letters specify the source of the operands. The operand source letters—R, M, and I—specify register, memory, and immediate data sources, respectively. A function can have only a single macro equivalent. The precision of its result and the source of its operands are inherent in its name.

For operators the utility may choose a macro equivalent. In those cases the utility will first look for a macro that provides the maximum precision of the operands. If a macro equivalent providing this required precision does not exist, the required precision will be reduced until one is found. If the precision of the selected macro equivalent is less than the required precision of the expression, a warning is issued in the listing. After the precision-based search is completed, the utility next tries to find a suitable macro equivalent that corresponds to the operand sources.

Once the "best" macro equivalent has been selected, the operands are adjusted accordingly by inserting housekeeping macros into the computational sequence. These housekeeping macros consist of precision conversions, loading registers from memory, loading registers with immediate data, and storing registers into scratch pad memory. These operations, of course, result in less accuracy and longer execution times. To aid the user in reformulating the program to improve accuracy and shorten execution time, a warning is issued each time a precision conversion macro is inserted into the computational sequence. This allows the user to reconsider the precision specified for the constants or variables identified in the warnings.

After the parsing of scaled-fraction expressions the RTMPL utility will scale the computational sequence. Since only binary scale factors are allowed in specifying RTMPL variables and constants, a scaling macro is inserted where required in the computational sequence. This macro shifts the result of the preceding operation to produce the required scaling. Whenever a scaling macro is inserted, a warning is issued in the listing file. The user may use these warnings to improve the accuracy and shorten the execution time of the simulation. Initially the user may specify the scale factors of variables and constants to be just large enough to handle the expected maximum values. Using the scaling warnings from successive passes of the source files through the utility, the user may then adjust the scale factors according to the warning messages. Minimizing the number of warning messages shortens the execution time of the simulation. Special warnings are issued by the utility whenever a scale factor of a variable or constant is required to be larger than that specified by the user. This warning implies a potential overflow and should be given special attention. Underflows may also result from improper scale factor or precision assignments, but these are not detected by the RTMPL utility.

Commands

RTMPL provides 13 command statements to allow the user to implement program control, to interface to target-defined states, to utilize target library procedures, and to communicate between the various processors in the simulator. The availability of these commands to the user requires the generation and definition of the corresponding target macros for the RTMPL utility during system implementation. Since many of the commands depend on the configuration, firmware, and data paths in the simulator, the user should refer to simulator targeting information for command availability and description. Use of undefined command statements in user programs will be flagged as errors by the RTMPL utility.

The COMMAND construct is defined in figure 23. Commands consist of key words (always followed by an underscore) and a command-dependent extension.

REDO and EXIT Commands

The REDO and EXIT commands are provided to enhance the flexibility of conditionals. RTMPL does not provide loop execution constructs such as

```
FOR ... DO ...
REPEAT ... UNTIL ...
WHILE ... DO ...
```

The REDO and EXIT commands, when properly used within a conditional structure, can provide the same effect. For example, to raise a variable A to its N^{th} power ($N \geq 1$), a Pascal programmer could write

```
B := A;
FOR I := 2 TO N DO B := B*A; A := B;
```

An RTMPL equivalent is

```
I = ONE;
_MULTIPLY_ B = B*A;
I = I + ONE;
IF_ I < N THEN_ REDO_ MULTIPLY;! A = B;
```

In general, an RTMPL requires more statements. However, the RTMPL program statements more closely reflect the actual machine operations and usually result in more time-efficient codes. This is important in developing real-time simulations, where minimization of computation time can mean the difference between success and failure.

As an example of the use of the EXIT command, consider the programming of Newton's method for determining the square root of a positive number. In Pascal

```
ROOT := 1;
WHILE ABS(NUMBER/SQR(ROOT) - 1) > =
EPSILON DO
ROOT := (NUMBER/ROOT + ROOT)/2;
```

The RTMPL equivalent is

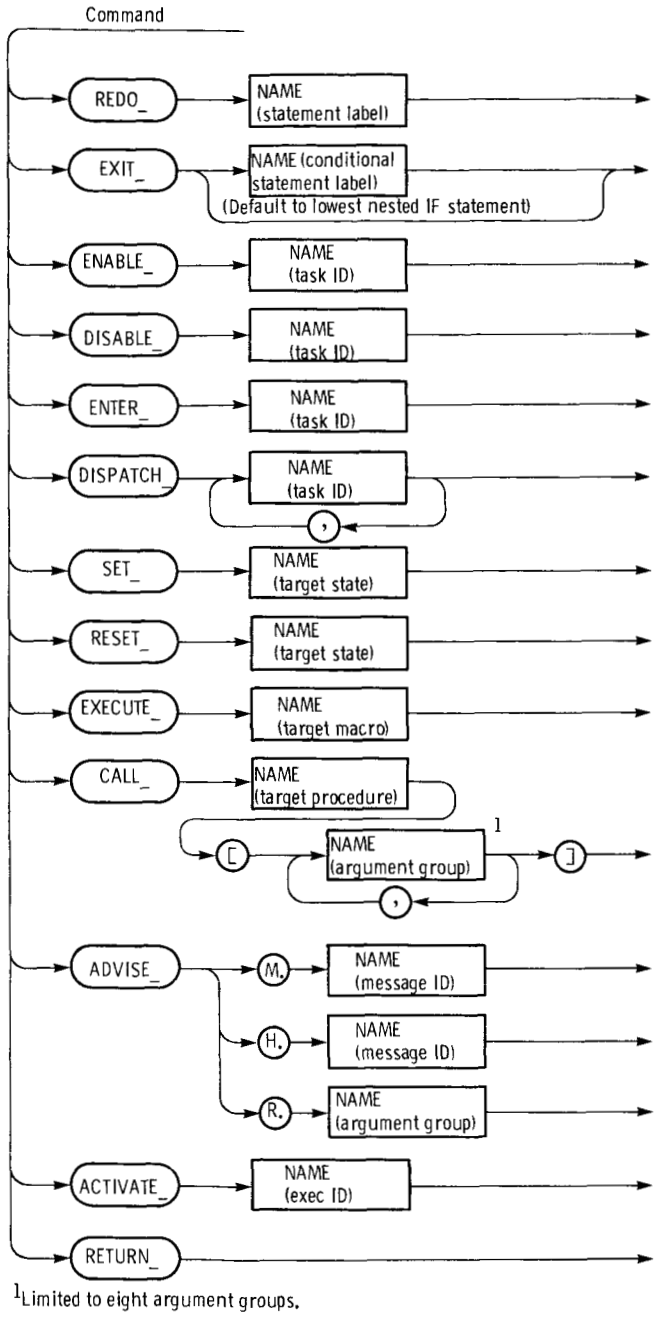


Figure 23. - Commands.

```
ROOT = ONE;
_TESTROOT_ IF_ ABS
(NUMBER/SQR(ROOT) - ONE) < EPSILON
THEN_ EXIT_;
ELSE_ ROOT = (NUMBER/ROOT + ROOT)/TWO;
REDO_TESTROOT;!
```

This example assumes the existence of the unary functions ABS (absolute value) and SQR (square). They must appear in the target definition files for the data type of the arguments. If no operand is supplied with the EXIT command, execution of the EXIT command will transfer program control to the statement following the exclamation point associated with the current conditional level. By supplying an operand, one can specify the conditional level to be exited. The statement

```
EXIT_TESTROOT;
```

would result in the same branching as the previous example.

To program the complex conditional logic contained in the Pascal statement

```
IF ((A < B) AND ((C < D) OR (E < F))) THEN G := H;
```

the RTMPL user could write

```
_DOLOGIC_IF_A < B THEN_
  IF_C# < D THEN_
  IF_E# < F THEN_EXIT_DOLOGIC;!!
  G = H;!
```

The double exclamation point terminates the C# < D and E# < F conditionals so that G = H will be computed if either conditional is false. Otherwise the conditional labeled "DOLOGIC" will be terminated and program control will be passed to the statement following the last conditional terminator(!).

The redo command allows the user to specify any previously defined label as the operand and thus permits backward jumps only. REDO may be used only within a conditional statement. EXIT also may be used only within a conditional statement to terminate the execution of that or a higher level conditional statement.

ENABLE and DISABLE Commands

The ENABLE and DISABLE commands allow the user to enable or disable the execution of any task defined in the program file. A task is executed only if the task is enabled (see ENTER and DISPATCH commands). Tasks may also be enabled or disabled at run time by using RTMPOS.

ENTER Command

The ENTER command causes program control to pass from an executive to a specified task if that task is enabled. It is valid only if used in an EXEC record. That is, a task cannot enter another task. The statements

```
ENTER_TASKA;
ENTER_TASKB;
```

cause sequential execution of TASKA and TASKB. Upon completion of TASKA (see RETURN command)

program control would revert to the executive, which would then transfer control to TASKB.

To minimize execution time, the ENTER command, by itself, does not preserve data registers. However, if executives of different priorities can execute a task, the RTMPL utility will change the ENTER command to a REENTER command. This will cause data registers to be preserved for reentry. The REENTER command is not available to the user but will appear on listings when generated by the utility. Scratch-pad memory conflicts are avoided by having a task scratch pad contained within executive memory space.

DISPATCH Command

The DISPATCH command is used to restrict execution of a task until all of the external variables required by that task are available. It is valid only when used in an EXEC record. The target processor can sense through firmware when variables have been transferred to its memory from another processor. The statement

```
DISPATCH_ TASKA, TASKB, TASKC;
```

will cause the local PREP to cyclicly test required variables for each task. If the variables for a task have arrived, the task is executed and marked complete. If a task is disabled, it is marked complete without variable testing. Upon completion of TASKB, for example, the processor would continue cyclic testing of TASKC and TASKA until they have also been marked complete. When all tasks are completed, they are remarked incomplete and the statement following DISPATCH is executed.

The RTMPL utility determines what external variables are necessary for DISPATCH. The utility does not provide for task reenterability when executed under DISPATCH. To avoid the problem, the user should only dispatch tasks from background executives.

SET and RESET Commands

The SET and RESET commands are used to manipulate the values of target Boolean variables. For example, the target processor may contain a bit in its status register that is a target Boolean variable called OVERFLOW. If the bit is automatically set when an arithmetic operation results in a register overflow condition, the statements

```
A = B + C
IF_ OVERFLOW THEN_
A = AMAX;
RESET_OVERFLOW;!
```

will test the bit for an overflow in the computation of A. If an overflow occurs, A will be limited and the bit reset in the status register. RESET assigns the Boolean value FALSE to a target state variable; SET assigns the Boolean value TRUE to a target state variable.

EXECUTE Command

The EXECUTE command is a general-purpose command that permits the user to execute any target-defined macro. Uses for this command depend strictly on the simulator definition. For example, a simulator macro could be defined that copies the value of the program counter into a preassigned location and terminates simulation processing. The macro could be called HALT. Then the statement

```
EXECUTE_HALT;
```

would cause execution of that macro. The present version of the utility does not permit the use of the EXECUTE command for macros requiring arguments.

CALL Command

The CALL command is used to invoke target library procedures. These procedures are prewritten during system installation and are, as needed, linked to the user's program during generation of the assembly language program. Target library procedures communicate with calling programs via argument groups (Argument Groups, Chapter 4). Argument groups contain constants and variables of the same data type and precision. The RTMPL utility structures the assembly language representations of the argument groups and the CALL command to pass the number of items in the group, the address of each item, and a processed value for each item to the procedure (Assembler Source Files, Chapter 9). The processed value may be used as an input argument to the procedure (formulated by other procedures or by RTMPOS) or as an output argument (formulated by the procedure itself). Processed values should not be confused with assigned values, which are results of assignment statements (variables) or defined values (constants). Assigned values may be used as input arguments for procedures. These are obtained from the specified item address. RTMPL assumes that an assigned value will never be an output argument of a procedure, although this is possible.

Target library procedures are useful for processing large volumes of data for display or analysis. Since their execution will normally be time consuming, they are usually not called from simulation programs but rather from RTX programs. For example, assume a procedure called SAMPLE exists in the target library and that its purpose is to obtain current values of single-precision, scale-fraction variables for output to data files. The user would first define an ARGGROUP to specify those variables:

ARGGROUP: SAMPDATA = S1, 25 [A,B,C,D]; EOR_;

This example specifies SAMPDATA as an argument group of DTP = S1 with a maximum of 25 items and initialized to contain four variables (A, B, C, and D). The procedure SAMPLE would be invoked as

CALL_SAMPLE[SAMPDATA];

When executed, SAMPLE would obtain current assigned values of A, B, C, and D and would store these values in their respective processed-value locations for use by the calling program. The calling program could then output the data to the disk by using the ADVISE command (see the next section). At run time SAMPDATA could be edited by RTMPOS to add up to 21 additional items of the same DTP to the argument group. Item removal and replacement is also possible.

The CALL construct (fig. 23) allows more than one (up to eight) ARGGROUP's to be specified as procedural arguments. This feature accommodates those procedures that require arguments of multiple data type and precision. Before using this construct the user should become familiar with the procedures contained in the target library and with their argument requirements.

ADVISE Command

ADVISE is used to interface a user program to RTMPOS. It consists of the command name, an action code, and an object. The action code is separated from the object by a decimal point. Table IV defines the function of the various action codes. For example, one could display a message, defined in the global data file as

MESSAGE:
T5LIMIT = STATION*5*TEMPERATURE*
LIMIT*EXCEEDED; EOR_;

by using the message advisory command

ADVISE_M.T5LIMIT;

When the command statement is executed, the T5LIMIT message appears on the terminal screen. If the action code were changed to an "H," the simulation would stop. With the "M" action code the simulation continues.

The "R" action code (read advisory) causes the referenced argument group to be read from local memory and processed (filed or displayed) by the operating system (RTMPOS). For example, to send the SAMPDATA argument group to a disk file after sampling, the statements would be

CALL_SAMPLE[SAMPDATA];

ADVISE_R.SAMPDATA;

Because of the multitude of data that can be transferred by using the read advisory, it may only be used in programs that are to reside on processors with direct access to the interactive information bus (COMP's).

Execution of the ADVISE command causes a priority interrupt to be issued from the local processor to the FEP. The FEP stops executing the RTMPOS background executive, obtains the action code and object from the local processor, and takes appropriate action. The RTMPOS background executive is then resumed. This priority processing ensures prompt action when executing the ADVISE command statement. Local processor program execution is delayed, as required, for memory access by the FEP. This delay could be substantial during processing of an "R" action. These delays must be considered by the user when structuring the simulation programs. In most cases it is best to issue advisories from processors intended for the RTX function.

TABLE IV.—ADVISORY ACTIONS

Action code	RTMPOS function
M	Displays message specified as object on user's terminal
H.	Stops simulation execution and displays message specified as object on user's terminal
R.	Reads argument group specified as object from disk data file to local processor

ACTIVATE Command

ACTIVATE provides the mechanism for initiating execution of foreground (priority level > 0) executives on the alternate processor in the local channel (i.e., a PREP may activate a COMP EXEC and vice versa). The operand must be the name of a foreground executive defined in the alternate processor program. For example, suppose that, in the program file DSCPREP. CHANNELA, a foreground EXEC record is defined as

EXEC: DOTASKS [1];
ENTER_TASKA; ENTER_TASKB;
EOR;

and that the companion COMP file (DSC.CHANNELA) contains the statement

ACTIVATE_DOTASKS;

Upon execution of the ACTIVATE command statement, the COMP would issue an interrupt to the PREP. The PREP would respond by reading the desired priority level (1) specified by the operand DOTASKS. From this the

PREP would determine the foreground executive to be executed. If a lower priority executive were active (the background EXEC in this case), its execution would be interrupted, DOTASKS would be executed, and the execution of the interrupted EXEC would be resumed. If a higher priority executive were active, DOTASKS would be placed in a pending que. It would then be executed when all higher priority EXEC's were completed.

RETURN Command

RETURN is used to terminate the execution of executives and tasks and must always be the last executable statement in each. It may also be contained within the body of a task or executive to allow termination as the result of conditionals. Execution of RETURN in a task restores program control to the executive from which the task was entered. If the task is reenterable, the saved registers will be restored. Execution of RETURN in an executive transfers program control to the next lowest priority executive pending (see ACTIVATE command). If no foreground executive is pending, the background executive is resumed. If the RETURN command is encountered in a background executive, program control returns to the processor's firmware.

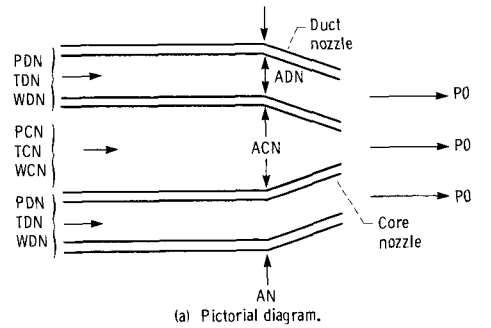
Chapter 6: RTMPL Simulation

The programming of a simple multiprocessor simulation will be used to illustrate some major operational aspects of the RTMPL utility and as an example of the object and listing files. In this chapter the simulation is described, the RTMPL source files are generated, and the use of the RTMPL utility to process these files is presented.

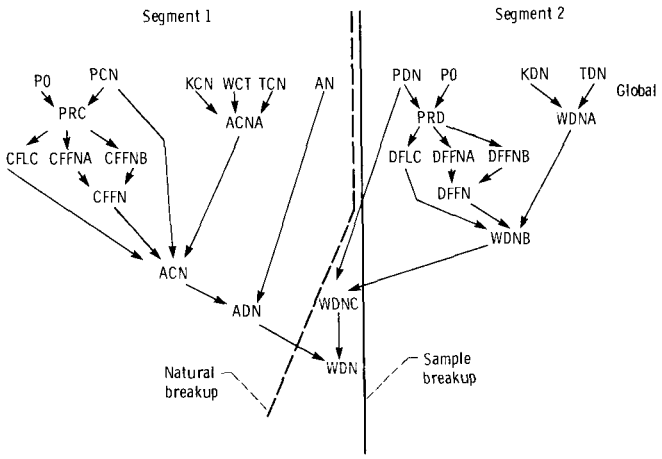
Description

A jet engine dual-exhaust-nozzle system is illustrated in figure 24(a). The nozzle system was chosen as the simulation example because the mathematical model of the nozzle (also shown) is fairly simple and lends itself to straight-forward partitioning into multiprocessor programs. The nozzle is modeled as two separate nozzles fed by separate core (CN) and duct (DN) sections of a turbofan engine (not simulated). It is assumed that the inlet conditions for the core and duct nozzles are known. They are pressures (PCN, PDN), temperatures (TCN, TDN), and weight flow rates (WCN, WDN). Both gas flows exit at the ambient pressure, P0. In the simulation the flow areas of both nozzles (ACN, ADN) are to be calculated with the constraint that the sum of the physical areas for the two nozzles is equal to the actual physical area (AN).

For the purpose of the example it is assumed that AN is specified as an input to the simulation and is read at the start of each computation interval through an analog-to-digital converter (ADC). For this simulation PCN, PDN, TCN, TDN, WCN, and P0 will be parameters. These parameters can be adjusted by the user at run time by using RTMPOS. The duct weight flow (WDN) and the core and duct nozzle areas (ACN, ADN) will be calculated variables. Finally it is assumed that the variables ACN and ADN will be output from the simulation through a digital-to-analog converter (DAC).



(a) Pictorial diagram.



(b) Segments of data flow diagram.

Figure 24. - Dual-exhaust-nozzle system.

Mathematical Model

For the dual-exhaust-nozzle system the given values are P0, PCN, KCN, WCT, TCN, AN, PDN, KDN, and TDN. All variables are expressed as computer variables to avoid dual definitions and are presented in table V. The equations used to model the system are

TABLE V.—SIMULATION VARIABLES AND PARAMETERS

Identification	Description	Type
AN	Total nozzle physical area	Variable
ANF	Total nozzle flow area	↓
CFLC	Flow coefficient (core)	
CFFN	Flow function (core)	
CFFNA	CFFN intermediate calculation (core)	
CFFNB	CFFN intermediate calculation (core)	
DFLC	Flow coefficient (duct)	
DFFN	Flow function (duct)	
DFFNA	DFFN intermediate calculation (duct)	
DFFNB	DFFN intermediate calculation (duct)	
P0	Ambient exhaust pressure	Parameter
PRC	Core nozzle pressure ratio	Variable
PCN	Core nozzle inlet pressure	Parameter
TCN	Core nozzle inlet temperature	Parameter
WCN	Core nozzle inlet weight flow	Parameter
ACN	Core nozzle flow area	Variable
ACNA	ACN intermediate calculation	Variable
PRD	Duct nozzle pressure ratio	Variable
PDN	Duct nozzle inlet pressure	Parameter
TDN	Duct nozzle inlet temperature	Parameter
WDN	Duct nozzle inlet weight flow	Variable
WDNA	WDN intermediate calculation	↓
WDNB	WDN intermediate calculation	
WDNC	WDN intermediate calculation	
ADN	Duct nozzle flow area	
PREPDONE	Interchannel logic variable	
DUCTDONE	Interchannel logic variable	
JOBDONE	Program-complete logic variable	--------

$$AN = AN + 800. \quad \text{where} \quad 50. \le AN \le 1600. \tag{1}$$

$$ANF = 1.049 - 1.622E-4 \times AN \tag{2}$$

$$PRC = P0/PCN \tag{3}$$

$$CFLC = \begin{cases} 0.825 & \text{if } CFLC \le 0.825 \\ 1.3635 - 0.7158 \times PRC & \\ 1.0 & \text{if } CFLC \ge 1.0 \end{cases} \tag{4}$$

$$CFFNA = PRC^{0.7143} \tag{5}$$

$$CFFNB = [1.0 - PRC^{0.2857}]^{1/2} \tag{6}$$

$$CFFN = \begin{cases} 0.2588 & \text{if } PRC \le 0.53 \\ CFFNA \times CFFNB & \text{if } PRC > 0.53 \end{cases} \tag{7}$$

$$PRD = P0/PDN \tag{8}$$

$$ACNA = KCN \times WCN \times TCN^{1/2} \tag{9}$$

$$ACN = \begin{cases} 0.0 & \text{if } CFFN \le 0.0 \\ ACNA \times CFFN \times CFLC/PC & \end{cases} \tag{10}$$

$$ADN = \begin{cases} 0.0 & \text{if } ACN = 0.0 \\ ANF - ACN & \end{cases} \tag{11}$$

$$DFLC = \begin{cases} 0.825 & \text{if } DFLC \le 0.825 \\ 1.575 - PRD & \\ 1.0 & \text{if } DFLC \ge 1.0 \end{cases} \tag{12}$$

$$DFFNB = PRD^{0.7143} \tag{13}$$

$$DFFNB = [1.0 - PRD^{0.2857}]^{1/2} \tag{14}$$

$$WDNA = KDN \times TDN^{1/2} \tag{15}$$

$$DFFN = \begin{cases} 0.2588 & \text{if } PRD \le 0.53 \\ DFFNA \times DFFNB & \text{if } PRD > 0.53 \end{cases} \tag{16}$$

$$WDNB = DFFN \times DFLC/WDNA \tag{17}$$

$$WDNC = PDN \times WDNB \tag{18}$$

$$WDN = ADN \times WDNC \tag{19}$$

Model Partitioning and Allocation

Before the model can be described in RTMPL, it must be partitioned and the resulting segments allocated to the processors in a selected configuration. The partitioning and allocation will depend on the number of channels in the simulator and the availability of COMP and PREP processors in the channels. Although the actual partitioning of mathematical models is not a primary topic of this report, it will be helpful to understand how the example problem was partitioned and how data were transferred between the channels. Figure 24(b) shows a data flow diagram of the equations in the nozzle model. Note from the structure of the diagram that the model naturally breaks up into parallel segments, as indicated by the dashed line. The only "crosstalk" between the segments is the WDN calculation, which needs ADN from segment 1 and WDCN from segment 2. However, to demonstrate the transfer of data in RTMPL, the diagram was broken up into two segments, as indicated by the solid line. The calculations of both WDCN and WDN were put into segment 1.

Since the model breaks up into two segments, three channels were used: two channels for the model calculations (the DSC channels) and one for the input/output and user interaction (the RTX channel). Since it is assumed that there is a COMP and a PREP in each channel, the problem was further broken up into six segments as shown in figure 25, which was derived from

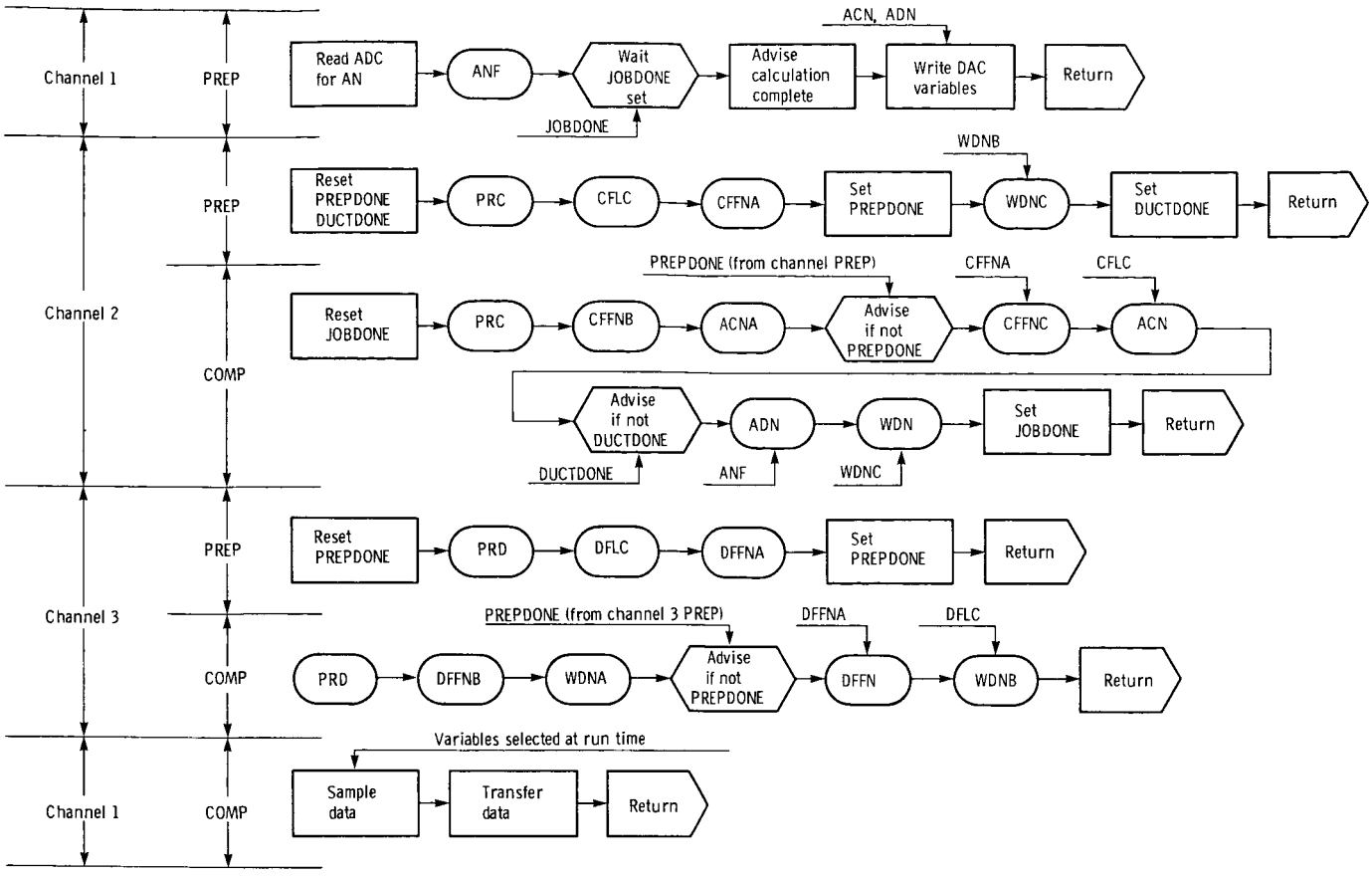


Figure 25. - Computational flow diagram.

the data flow chart (fig. 24(b)). In figure 25, variables within the oval elements represent the equations used to calculate the variables. Other program functions are shown in rectangles. In general, the segments were partitioned and allocated to demonstrate RTMPL rather than the solution of the problem. For a time-critical problem care must be taken in allocating calculations. The philosophy used for the example problem allocation is as follows:

(1) Channel 1 was selected as the RTX channel.

(a) It was assumed that the PREP processor in channel 1 is connected to the outside world through ADC and DAC. Since it is desired to read AN from an ADC, the calculation of ANF was assigned to that channel.

(b) The COMP processor in channel 1 was used for the input and adjustment of PCN, P0, etc., and for user interaction with the simulation.

(2) Channel 2 was designated as the segment 1 channel. In general, calculations were distributed between the COMP and the PREP to demonstrate data transfers. For example, variables CFFNA and WDNC must be transferred from the channel 2 PREP to the channel 2 COMP. Also ANF must be transferred from the channel 1 PREP to the channel 2 COMP.

(3) Channel 3 was used as the segment 2 channel. Here also calculations were distributed between the COMP and the PREP to demonstrate data transfers. For example, DFLC and DFFNA must be transferred from the channel 3 PREP to the channel 3 COMP. Note also that WDNB must be transferred from the channel 3 COMP to the channel 2 PREP.

Thus the arbitrary breakup of the model has variables transferred from (1) a PREP to a COMP in the same channel, (2) a PREP in one channel to a COMP in another channel, and (3) a COMP in one channel to a PREP in another channel.

Channel and processor assignments are shown on the left side in the computational flow diagram (fig. 25). The string of calculations/operations assigned to each processor is shown on the right side. Data transfer between processors is also indicated. Boolean variables PREPDONE, DUCTDONE, and JOBDONE were added to the simulation simply as a mechanism to demonstrate the ADVISE command. Although providing

some data transfer synchronization, they are unnecessary since the RTMPL utility automates this synchronization (Information Transfer, Chapter 1).

Depending on the application of a simulation it is often desirable to include certain analytical computations to permit the gathering of data, the monitoring of simulator performance, and the control of the simulation execution. Under RTMPOS many analytical functions can be performed. Many of the RTMPL constructs are designed to support analytical functions. As discussed earlier, advisories allow the user to display messages and to stop execution on the basis of simulation performance. They also support the gathering of data structured in terms of argument groups. The CALL command allows analytical routines from the target library to be incorporated into the simulation. Task enabling and disabling may be used to control the execution of the analytical functions (as well as the mathematical model). The ACTIVATE command may be used to trigger analytical functions on the alternate processor in a channel on the basis of occurrences in the local processor's calculations. The example simulation incorporates many of these constructs.

At run time the user specifies the parametric values and sets the simulation calculation update interval by using RTMPOS. During execution the simulation is repetitively calculated, once each update interval. The calculations on each processor proceed sequentially, as diagrammed in figure 25. The following paragraphs describe the desired sequence of operations.

The PREP in the RTX channel (channel 1) reads the ADC for AN, and computes the flow area, ANF, for use on the COMP in channel 2. Its task is then complete until the mathematical model has been computed. This is determined by the Boolean variable JOBDONE, calculated on the COMP in channel 2. When JOBDONE becomes true, the channel 1 PREP writes ACN, ADN, CFFNB, and DFFNB to DAC's and advises the operator of the completion of the calculation sequence.

The DSC PREP in channel 2 controls the Boolean variables PREPDONE and DUCTDONE. They are used on the channel 2 COMP to advise the operator of calculation delays if the channel 2 PREP or channel 3 data (respectively) have not arrived when required to complete the channel 2 COMP calculations. Note that CFLC and CFFNA are required for calculating ACN. PREPDONE is set after their calculation. The channel 2 PREP then computes WDNC on the basis of WDNB computed in channel 3. When this calculation is complete, DUCTDONE is set. The channel 2 COMP then completes its calculation sequence by computing ADN and WDN and then setting JOBDONE true.

The channel 3 processors work in tandem to calculate WDNB. Another PREPDONE variable is used to signal the operator of calculation delay due to PREP data (DFFNA) not arriving on time to complete the calculation of WDNB on the COMP.

The COMP processor in the RTX channel (channel 1) is assigned the task of sampling simulation data and transferring these data to the FEP for analysis. Argument groups are used in developing the RTX COMP to present the run-time selection of the data items to be sampled.

Model Translation to RTMPL

RTMPL contains both a programming language for the mathematical model on the parallel processors and a set of commands for coordinating execution and obtaining data and for interaction of the simulation with the user and the real-time world. The different aspects of the language will be covered in the context of the sample problem.

Equations (1) to (19) must be converted to RTMPL by using the constructs defined in figures 10 to 12 and 18 to 22. This will be done for each channel and processor according to the variable distribution in figure 25. Note that no simulation equations are assigned to channel 1 COMP, which is reserved for analysis functions.

Channel 1 PREP

The only two equations solved on the channel 1 PREP are equations (1) and (2). The constants in these equations are 1.049, 1.622E-4, 50., 800., and 1600. From the definitions of figure 12

```
K1P049 = S1/1[1.049];
K1P622M4 = S1/ - 12[1.622E-4];
MINAREA = S1/11[50.];
MAXAREA = S1/11[1600];
K2 = S1/11[800.];
```

The variables are AN and ANF. From the variable definition of figure 11

```
AN = S1/11[1600.,1600.];
ANF = S1/11[1263.168,1263.168];
```

From figures 19 to 21(a) equation (1) becomes

```
AN = AN + K2;
IF_ AN < MINAREA
THEN_ AN = MINAREA;
ELSE_
IF_ AN > MAXAREA
THEN_ AN = MAXAREA; !!                    (20)
```

Equation (2) becomes

```
ANF = K1P049 - K1P622M4 * AN;                    (21)
```

Channel 2 PREP

The equations solved on the channel 2 PREP are (3) to (5) and (18). The constants are P0, PCN, 1.3635, 0.7158, 0.825, 1.0, and PDN. From the construct of figure 12

```
K1P3625 = S1/1[1.3625];
KP7158 = S1/0[.7158];
KP825 = S1/0[.825];
K1P = S1/1[1.0];
P0 = S1/6[14.7];
PCN = S1/7[14.7];
PDN = S1/7[14.7];
```

The variables are PRC, CFLC, CFFNA, WDNC, and WDNB. From the construct in figure 11

```
PRC = S1/1[1./1.];
CFFNA = S1/1[1./1.];
CFLC = S1/1[.825/.825];
WDNC = S1/2[.825/.825];
```

WDNB comes from channel 3 COMP, and from the figure 13 construct it is referenced as

⟨channel 3 name⟩.C.WDNB

The equations are translated to RTMPL. From figures 19 and 21(a) equation (3) becomes

```
PRC = P0/PCN;                                    (22)
```

and equation (4), from figures 19 to 21(a), becomes

```
CFLC = K1P3625 - K1P7158 * PRC;

IF_ CFLC > K1P
THEN_ CFLC = K1P;
ELSE_
IF_ CFLC < KP825
THEN_ CFLC = KP825; !!                           (23)
```

Equation (5) is an exponential and can be solved by using a univariate function. The function will be defined as FUN1 with the construct

```
CFFNA = FUN1[PRXVALS,PRNVALS,XT07143,
                                  PRC];          (24)
```

where

```
PRNVALS = I1[21];

PRXVALS = S1/1,21[0.,.05,.10,.15,.20,.25,.30,.35,
                  .40,.45,.50,.55,.60,.65,.70,.75,.80,.85,
                  .90,.95,1.0];

XT07143 = S1/1,21[0.000,.1177,.1931,.2579,.3166,
                  .3820,.4232,.4724,.5197,.5653,.6095,
                  .6524,.6943,.7351,.7751,.8142,.8527,
                  .8904,.9275,.9640,1.000];
```

The method is to calculate PRC, search the range of PRXVALS, find the corresponding values in XT07143, and interpolate.

Equation (18) involves a transfer variable WDNB:

```
WDNC = PDN * ⟨channel 3 name⟩.C.WDNB;            (25)
```

Channel 2 COMP

The equations solved on the channel 2 COMP are (3), (6), (7), (9) to (11), and (19). The constants are P0, PCN, 1.0, KCN, 0.2588, WCN, TCN, 0.53, and 0.0. From the constructs of figure 12

```
P0 = S1/6[14.7];
PCN = S1/7[14.7];
K1P = S1/1[1.];
KCN = S1/0[.5124];
KP2588 = S1/-1[.2588];
KP53 = S1/0[.53];
ZERO = S1/0[0.];
WCN = S1/8[0.];
TCN = S1/13[900.];
```

The variables are PRC, CFFNB, ACNA, CFFN, ACN, CFLC, CFFNA, ADN, WDN, ANF, and WDNC. From the construct of figure 11

```
PRC = S1/1[1./1.];
CFFNB = S1/1[1.,1.];
ACNA = S1/11[0.,0.];
CFFN = S1/2[0.,0.];
ACN = S1/10[0.,0.];
ADN = S1/10[0.,0.];
WDN = S1/9[0.,0.];
```

CFLC, CFFNA, and WDNC are variables transferred from the channel 2 PREP. Thus from the construct of figure 13 these variables are referenced as .P.CFLC,.P.CFFNA, and.P.WDNE, respectively.

ANF is transferred from the channel 1 PREP and referenced as

⟨channel 1 name⟩.P.ANF

From figures 19 and 21(a), equation (3) becomes

```
PRC = P0/PCN;                                    (26)
```

Note that this equation was also implemented on channel 2 PREP to avoid data transfer delays. Equation (6) uses the FUN1 function and a square root univariate function (SQRT):

```
FFNB = SQRT(K1P – FUN1[PRXVALS,PRNVALS,
                       XT02857,PRC];             (27)
```

where

```
PRNVALS = I1[21];
PRXVALS = S1/1,21[etc.];
XT02857 = S1/1,21[etc.];
```

Equation (9) becomes

```
ACNA = KCN * WCN * SQRT (TCN);                   (28)
```

From figures 19 to 21(a) equation (7) becomes

```
IF_ PRC# > KP53
THEN_ CFFN = KP2588;
ELSE_ CFFN = CFFNB * .P.CFFNA; !                 (29)
```

Equation (10) becomes

```
IF_ CFFN > ZERO
THEN_ ACN = ACNA * CFFN * .P.CFLC/PCN;
ELSE_ ACN = ZERO; !                              (30)
```

Equation (11) becomes

```
IF_ ACN = ZERO
THEN_ ADN = ZERO;
ELSE_ ADN = (CHANNEL 1 name).P.ANF – ACN; !      (31)
```

Equation (19) becomes

```
WDN = ADN * .P.WDNC;                             (32)
```

Channel 3 PREP

Equations (8), (12), and (13) are solved on the channel 3 PREP. The constants are P0, PDN, 1.575, 0.825, and 1.0. From the construct in figure 12

```
P0 = S1/6[14.7];
PDN = S1/7[14.7];
K1P575 = S1/1[1.575];
KP825 = S1/0[.825];
K1P = S1/1[1.];
```

The variables are PRD, DFLC, and DFFNA. From the construct in figure 11

```
PRD = S1/1[1./1.];
DFLC = S1/1[.825/.825];
DFFNA = S1/1[1./1.];
```

From figures 19 and 21(a) equation (8) becomes

```
PRD = P0/PDN;                                    (33)
```

and equation (12) becomes

```
DFLC = K1P575 – PRD;
IF_ DFLC > K1P
THEN_ DFLC = K1P;
ELSE_
IF_ DFLC < KP825
THEN_ DFLC = KP825; !!                           (34)
```

Equation (13) becomes

```
DFFNA = FUN1[PRXVALS,PRNVALS,XT07143,
                                  PRD];          (35)
```

where

```
PRNVALS = I1[21];
PRXVALS = S1/1,21[etc.];
XT07143 = S1/1,21[etc.];
```

Channel 3 COMP

The equations solved on the channel 3 COMP are (8) and (14) to (17). The constants are P0, PDN, 1.0, KDN, TDN, 0.2588, and 0.53. From the constructs in figure 12

```
P0 = S1/6[14.7];
PDN = S1/7[14.7];
TDN = S1/13[900.];
KDN = S1/0[.50655];
K1P = S1/1[1.];
KP2588 = S1/0[.2588];
KP53 = S1/0[.53];
```

The variables are PRD, DFFNB, WDNA, DFFNA, DFFN, WDNB, and DFLC. From figure 11

```
PRD = S1/1[1./1.];
DFFNB = S1/1[1./1.];
WDNA = S1/8[151.665/151.665];
DFFN = S1/2[0./0.];
WDNB = S1/5[0./0.];
```

DFFNA and DFLC are transfer variables from the channel 3 PREP; thus from figure 13, they are referenced

as .P.DFFNA and .P.DFLC, respectively. From figures 19 and 21(a) equation (8) becomes

PRD = P0/PDN; (36)

Using the SQRT and FUN1 functions we get equation (14) as

DFFNB = SQRT(K1P – FUN1[PRXVALS,PRNVALS, XT02857,PRD]; (37)

Equation (15) becomes

WDNA = KDN * SQRT(TDN); (38)

Equation (17) becomes

WDNB = DFFN * .P.DFLC/WDNA; (39)

From figures 19 to 21(a) equation (16) becomes

IF_ PRD# > KP53
THEN_ DFFN = KP2588;
ELSE_ DFFN = DFFNB * .P.DFFNA; ! (40)

Example Source Files

The required RTMPL source files are a control file, a global data file, and program source files for the various processors. The files are shown in figure 26 and will be described in detail for the example problem.

Control Segment Source File

The control file for the simulation is arbitrarily named "DUALSIM" and is shown in figure 26(a). The construct for the file is given in figure 7. In the file the

```
PAGE     1    LIST VERSION  042682 3  12/11/84  08:44:35   DUALSIM


NONE;
DUALNOZZ;
RTMPL*TEST*CASE;
DALE*J.*ARPASI;
DEV1;
DEV1;
DEV1;
0;
3;
RTX.DATAPROC;
DSC.CORESIM;
DSC.DUCTSIM;
GLOBAL.DUALNOZZ;
MG000.MACHCHAR;
```

(a) Control.

```
PAGE     1    LIST VERSION  042682 3  12/11/84  08:05:18   DEV1:0.GLOBAL.DUALNOZZ


MESSAGE:ADCMESS1=SIMULATION*HALT!*ADC*VALUE*OF*AN*<*MINAREA;
  ADCMESS2=SIMULATION*HALT!*ADC*VALUE*OF*AN*>*MAXAREA;
  COREMES1=SIMULATION*HALT!*PRC*OVERFLOW;
  COREMES2=SIMULATION*HALT!*PRD*OVERFLOW;
  COREMES3=DUCTSIM*OR*ADC*DELAY*ENCOUNTERED;
  COREMES4=CORESIM*PRE-PROCESSING*DELAY*ENCOUNTERED;
  DUCTMESS=DUCTSIM*PRE-PROCESSING*DELAY*ENCOUNTERED;
  DFMESS=COMPUTATION*CYCLE*COMPLETE;
EOR;

GLCNST:
  .P0=S1/6[14.7];
  .PCN=S1/7[14.7];
  .PDN=S1/7[14.7];
  .TCN=S1/13[900.];
  .TDN=S1/13[900.];
  .WCN=S1/8[0.];
  PRNVALS=I1[21];
  PRXVALS=S1/1,21[0.,.,05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,.65,.7,.75,
                  .8,.85,.9,.95,1.];
  XT02857=S1/1,21[0.,.,4249,.5180,.5816,.6314,.6730,.7089,.7409,.7697,.7960,
                  .8203,.8430,.8642,.8842,.9031,.9211,.9382,.9546,.9703,
                  .9855,1.];
  XT07143=S1/1,21[0.,.,1177,.1931,.2579,.3166,.3820,.4232,.4724,.5197,.5653,
                  .6095,.6524,.6943,.7351,.7751,.8142,.8527,.8904,.9275,
                  .9640,1.];
EOR;
```

(b) Global data.

Figure 26. - Dual-nozzle simulation source files.

```
PAGE     1     LIST VERSION  042682 3  12/11/84  08:04:12   DEV1:0.RTXPREP.DATAPROC


EXEC:GETAN[0];
  CALL← READ[ADCVAR,ADCCHN];
  AN=AN+K2;
  IF← AN<MINAREA THEN← AN=MINAREA; ACTIVATE← BADADC;
  ELSE← IF← AN>MAXAREA THEN← AN=MAXAREA; ACTIVATE← BADADC;!!
  ANF=K1P049-K1P622M4*AN;
  ENTER← JOBDONE;
  DISPATCH← WRTACN,WRTADN;
  RETURN←;
EOR;

TASK:WRTACN;
  CALL← DAC1[ACNG]; RETURN←;
EOR;

TASK:WRTADN;
  CALL← DAC2[ADNG]; RETURN←;
EOR;

TASK:JOBDONE;
  ←TEST←  IF← #CORESIM.C.JOBDONE THEN← REDO← TEST;
  ELSE← ADVISE←H.DPMESS;!
  RETURN←;
EOR;

ARGGROUP:
  ACNG=S1,1[CORESIM.C.ACN];
  ADNG=S1,1[CORESIM.C.ADN];
EOR;

CONSTANT:
  MINAREA=S1/11[50.];
  MAXAREA=S1/11[1600.];
  K1P049=S1/1[1.049];
  K1P622M4=S1/-12[1.622E-4];
  .K1=I1[1];
  K2=S1/11[800];
EOR;

VARIABLE:
  AN=S1/11[1600/1600]; ANF=[1263.168/1263.168];
EOR;

ARGGROUP:
  ADCVAR=S1,32[AN];
  ADCCHN=I1,32[K1];
EOR;
```

(c) DATAPROC channel, PREP program.

```
PAGE     1     LIST VERSION  042682 3  12/11/84  08:04:02   DEV1:0.RTX.DATAPROC


EXEC:MAIN[0];
  ENTER← GETDATA;
  RETURN←;
EOR;

EXEC:BADADC[1];
  IF← .P.AN=.P.MINAREA THEN← ADVISE←H.ADCMESS1; ELSE← ADVISE←H.ADCMESS2;!
  RETURN←;
EOR;

TASK:GETDATA;
  CALL← SAMPLE[DATA];
  ADVISE←R.DATA;
  RETURN←;
EOR;

ARGGROUP:
  DATA=S1,16[.P.AN,.P.ANF,CORESIM.ACN,CORESIM.ADN,CORESIM.PRC,DUCTSIM.PRD,
            CORESIM.WDN];
EOR;
```

(d) DATAPROC channel, COMP program.

Figure 26. - Continued.

```
PAGE     1    LIST VERSION  042682 3  12/11/84  08:04:43   DEV1:0.DSCPREP.CORESIM

EXEC:PRFCTNS[0];
  PREPDONE=FALSE; DUCTDONE=FALSE;
  PRC=P0/PCN;
  FLC=K1P3625-KP7158*PRC;
  IF← FLC>K1P THEN←FLC=K1P; ELSE← IF← FLC<KP825 THEN← FLC=KP825;!!
  FFNA=FUN1[PRXVALS,PRNVALS,XT07143,PRC];
  PREPDONE=TRUE;
  WDNC=PDN*DUCTSIM.C.WDNB;
  DUCTDONE=TRUE;
  RETURN←;
EOR;

CONSTANT:
  K1P3625=S1/1[1.36253; KP7158=S1/0[.71583; K1P=S1/1[1.3; KP825=S1/0[.8253;
EOR;

VARIABLE:
  PRC=S1/1[1./1.3; FFNA;
  FLC=S1/1[.825/.8253;
  WDNC=S1/2; PREPDONE=B[TRUE/FALSE]; DUCTDONE;
EOR;
```

(e) CORESIM channel, PREP program.

```
PAGE     1    LIST VERSION  042682 3  12/11/84  08:04:29   DEV1:0.DSC.CORESIM

EXEC:MAINSIM[0];
  JOBDONE=FALSE;
  PRC=P0/PCN;
  IF← OVERFLOW THEN← ADVISE←H.COREMES1;!
  FFNB=SQRT(K1P-FUN1[PRXVALS,PRNVALS,XT02857,PRC]);
  ACNA=KCN*WCN*SQRT(TCN);
  IF← #.P.PREPDONE THEN← ADVISE←M.COREMES3;!
  IF← PRC#>KP53 THEN← FFN=KP2588; ELSE← FFN=FFNB*.P.FFNA;!
  IF← OVERFLOW THEN← ADVISE←H.COREMES2;!
  IF← FFN>ZERO THEN← ACN=ACNA/PCN*FFN*.P.FLC; ELSE← ACN=ZERO;!
  IF← #.P.DUCTDONE THEN← ADVISE←M.COREMES4;!
  IF← ACN=ZERO THEN← ADN=ZERO; ELSE← ADN=DATAPROC.P.ANF-ACN;!
  WDN=ADN*.P.WDNC;
  JOBDONE=TRUE;
  RETURN←;
EOR;

CONSTANT:
  KCN=S1/0[.51243; KP53=S1/0[.533; K1P=S1/1[1.3; KP2588=S1/-1[.25883;
  ZERO=S1/0[0.3;
EOR;

VARIABLE:
  JOBDONE=B[TRUE/FALSE]; FFN=S1/2; FFNB=S1/1[1./1.3;
  ACN=S1/10; ADN; WDN=S1/9; ACNA=S1/11;
  .PRC=S1/1[1./1.3;
EOR;
```

(f) CORESIM channel, COMP program.

Figure 26. - Continued.

name of the simulation is DUALNOZZ; the volume (or disk) for the source, object, and data base is DEV1; the user number is 0; the number of channels is three. The first channel is RTX.DATAPROC; the second is DSC.CORESIM; the third is DSC.DUCTSIM. There will be global data, so GLOBAL.DUALNOZZ is specified, and the target file catalog name is MC68000. The global file name and the program file names—one for each of the six processors—must be the same as specified in the control file. The file names are

Channel 1 program file:
DEV1:0.RTXPREP.DATAPROC.SA
DEV1:0.RTX.DATAPROC.SA

Channel 2 program file:
DEV1:0.DSCPREP.CORESIM.SA
DEV1:0.DSC.CORESIM.SA

Channel 3 program file:
DEV1:0.DSCPREP.DUCTSIM.SA
DEV1:0.DSC.DUCTSIM.SA

Global file:
DEV1:0.GLOBAL.DUALNOZZ.SA

This is a VERSAdos format.

```
PAGE     1     LIST VERSION  042682 3  12/11/84  08:05:07   DEV1:0.DSCPREP.DUCTSIM

EXEC:PREFCTNS[0];
  PREPDONE=FALSE;
  PRD=P0/PDN;
  FLC=K1P575-PRD;
  IF← FLC>K1P THEN←FLC=K1P; ELSE← IF← FLC<KP825 THEN← FLC=KP825;!!
  FFNA=FUN1[PRXVALS,PRNVALS,XT07143,PRD];
  PREPDONE=TRUE;
  RETURN←;
EOR;

CONSTANT:
  K1P575=S1/1[1.575]; K1P=S1/1[1.]; KP825=S1/0[.825];
EOR;

VARIABLE:
  PRD=S1/1[1./1.]; FFNA;
  FLC=S1/1[.825/.825];
  PREPDONE=B[TRUE/FALSE];
EOR;
```

(g) DUCTSIM channel, PREP program.

```
PAGE     1     LIST VERSION  042682 3  12/11/84  08:04:55   DEV1:0.DSC.DUCTSIM

EXEC:COPROCESE[0];
  PRD=P0/PDN;
  IF← OVERFLOW THEN← ADVISE←H.COREMES2;!
  FFNB=SQRT(K1P-FUN1[PRXVALS,PRNVALS,XT02857,PRD]);
  WDNA=KDN*SQRT(TDN);
  IF← #.P.PREPDONE THEN← ADVISE←M.DUCTMESS;!
  IF← PRD#>KP53 THEN← FFN=KP2588; ELSE← FFN=FFNB*.P.FFNA;!
  WDNB=FFN*.P.FLC/WDNA;
  RETURN←;
EOR;

CONSTANT:
  K1P=S1/1[1.]; KDN=S1/0[.50655]; KP53=S1/0[.53]; KP2588=S1/0[.2588];
EOR;

VARIABLE:
  FFN=S1/2; WDNB=S1/5;
  FFNB=S1/1[1./1.]; WDNA=S1/8[151.665/151.665];
  .PRD=S1/1[1./1.];
EOR;
```

(h) DUCTSIM channel, COMP program.

Figure 26. - Concluded.

Global Data Segment Source File

The global data file format (fig. 9) consists of global constant (GLCNST) and message (MESSAGE) records. For the dual-nozzle simulation the file is shown in figure 26(b). Note that the MESSAGE record contains eight messages that can be invoked at different parts of the simulation to advise the user of the simulation progress. The GLCNST record consists of all constants to be distributed to all channels. The dot in front of the constant means that the constant is a parameter. Note that the vectors needed for the FUN1 function are also included in the global data.

Program Source Files

RTMPL program files consist of at least one EXEC record and can include VARIABLE, CONSTANT, ARGGROUP, and TASK records. The construct for the program file is shown in figure 8. The construct, along with the computational flow diagram of figure 25, was used to create the program files for the six processors.

Channel 1 PREP.—Channel 1, the RTX channel, is given the name DATAPROC. The file name for the PREP is DEV1:0.RTXPREP.DATAPROC.SA. The file (fig. 26(c)) consists of one EXEC, three TASK, two ARGGROUP, one CONSTANT, and one VARIABLE record. The EXEC record is

```
EXEC:GETAN[0];
```

Since the first part of the diagram in figure 25 says to read an ADC for variable AN, the executive was arbitrarily named GETAN and given a zero priority level (fig. 7), which is a background or lowest priority (fig. 15) job.

The actual reading of the ADC is done with the CALL_ command (fig. 23):

CALL_ READ[ADCVAR,ADCCHN];

where READ is the target procedure, ADCVAR is an argument group containing the variable AN, and ADCCHN is an argument group containing the ADC channel number. ANF is then calculated by using equations (20) and (21). Note that the ACTIVATE_ command is used (fig. 23) if AN>MAXAREA or AN<MINAREA. This sends an interrupt to the DATAPROC COMP and the corresponding EXEC (BADADC) is activated there. Next a Boolean variable is checked to see if the simulation has completed the calculation of WDN in the channel 2 COMP. The ENTER command (fig. 23)

ENTER_ JOBDONE;

is used, where JOBDONE is a task that does the actual testing and must be defined in this source file.

After execution of the JOBDONE task, ACN and ADN are written to DAC's by using the DISPATCH_ command from figure 23:

DISPATCH_ WRTACN,WRTADN;

where WRTACN and WRTADN are tasks that must be records written in this source file. The RETURN_ command from figure 23 terminates execution of the EXEC. Finally an EOR completes the record.

The WRTACN task record is then defined:

TASK:WRTACN;

This is one of the DISPATCH_ tasks. It is formed by using the constructs of figures 8 and 17. It uses the CALL_ command (fig. 23) to call target procedure DAC1 with ARGGROUP ACNG. This procedure writes its argument to DAC one. A similar definition follows for WRTADN to write ADN to DAC two.

The JOBDONE task is then defined. This task tests to see if CORESIM.C.JOBDONE is not set and redoes the test until it is. Note that the construct is from figures 8 and 18 where _TEST_ is the label name and IF, THEN, ELSE is a conditional. Once the test passes, it uses the ADVISE command (fig. 23):

ADVISE_ H.DPMESS: !

where H is an action code to stop the simulation and to print out DPMESS, which is a message in the global data files. This task can be disabled at run time by RTMPOS to enable continuous simulation.

ACNG and ADNG are needed for the write-to-DAC tasks. They are formed by using figures 8, 13, and 14. An additional constant is defined as the AAC channel number:

.K1 = I1[1];

where the "." indicates that it is a parameter, adjustable at run time.

Argument groups are then defined for the read procedure. The construct is shown in figures 8 and 14.

ADCVAR = S1/32[AN];
ADCCHN = I1/32[K1];

For illustration the size of these argument groups is greater than one. Size 32 indicates that 32 variables can be read on 32 channels by adding items to these groups at run time.

Channel 1 COMP.—The file name for the COMP processor is DEV1:0.RTX.DATAPROC.SA. The file (fig. 26(d)) consists of two EXEC's, one TASK, and one ARGGROUP. The constructs for the records are given in figures 16 and 18 to 22.

The first executive is defined as

EXEC:MAIN[0];

This EXEC is given the name MAIN with priority 0 (fig. 17), meaning it is a background EXEC (fig. 15). Its purpose is to sample the values of variables at each cycle of computation. The ENTER_ command (fig. 23) is used to begin execution of GETDATA, which is a task to be defined later. GETDATA will do the actual sampling and transfer the data to RTMPOS.

The second executive

EXEC:BADADC[1];

is defined to service the ACTIVATE command used in the PREP. The priority 1 indicates that this foreground executive will have priority over the background executive. This executive will halt the simulation if the AN value read from an ADC is outside the MINAREA to MAXAREA range. ADVISE_ prints a message from the MESSAGE record in the global data file.

The GETDATA task is then defined:

TASK:GETDATA;

This task uses the CALL_ command (fig. 23) to call a target procedure SAMPLE for the DATA ARGGROUP. ADVISE_ with R.DATA says to read the argument group data, which must be defined. ADVISE_R. must be on a processor tied to the interactive bus. The DATA ARGGROUP (defined by figs. 8 to 14) supports the SAMPLE procedure. Note that the arguments are defined with the argument specifications in figure 13.

Channel 2 PREP.—Channels 2 and 3 are used to simulate the model equations. The channel 2 PREP program file (fig. 26(e)) is called DEV1:0.

DSCPREP.CORESIM.SA. It consists of one EXEC, one CONSTANT, and one VARIABLE record. The executive is defined as

EXEC:PRFCTNS [0];

It is a background executive. The two Boolean variables PREPDONE and DUCTDONE are set false to indicate that calculations have not been completed. Values for PRC, CFLC, and CFFNA are then calculated. These calculations are given in equations (22) to (24). The PREPDONE variable is set true to indicate that the calculations have been completed. WDNC is then calculated from equation (25) (where "channel 3 name" is DUCTSIM). The variable DUCTDONE is set true to indicate that the WDNC calculations have been completed. The Boolean variables are defined, by figure 11, as

PREPDONE = B[TRUE/FALSE];

DUCTDONE = B[TRUE/FALSE]'

Channel 2 COMP.—The channel 2 COMP source file is called DEV1:0.DSC.CORESIM.SA. The file (fig. 26(f)) consists of one EXEC, one CONSTANT, and one VARIABLE record. The executive is defined as

EXEC:MAINSIM[0];

The EXEC is given the name MAINSIM with priority 0. The variable JOBDONE is set to false to indicate that calculations for this pass through the model have not begun. PRC is then calculated from equation (26). If the calculation overflows, the ADVISE command is used to halt the simulation and give the CORESIM1 message from the global data file. CFFNB and ACNA are then calculated from equations (27) and (28). A test is then made to see if the channel 2 PREP Boolean variable PREPDONE has been set true. If PREPDONE = TRUE, no data transfer delay has been encountered. Otherwise

_ADVISE_M.COREMES3;

causes a printout that there is a delay. CFFN and ACN are then calculated from equations (29) and (30). If DUCTDONE is true, calculations can continue; otherwise

ADVISE_M.COREMES4;

causes a printout of the COREMES4 message from the global data file. The variables ADN and WDN are then calculated from equations (31) and (32). Note that DATAPROC is the name for channel 1. The variable JOBDONE (referenced in channel 1) is set true to indicate that the calculations have been completed. The variable

JOBDONE = B[TRUE/FALSE];

is added to the variables for this processor.

Channel 3 PREP.—The channel 3 PREP source file (fig. 26(g)) is called DEV1:0.DCSPREP.DUCTSIM.SA. It consists of one EXEC, one CONSTANT, and one VARIABLE record. The executive is defined as

EXEC:PRFCTNS[0];

This is a background executive. The Boolean variable PREPDONE is set false to indicate that the calculations have not been completed. PRD is calculated from equation (33). The target Boolean variable, OVERFLOW, is checked and, if true, command causes the simulation to halt and CORMES2 from the global data file is printed out. DFLC and DFFNA are calculated from equations (34) to (35). PREPDONE is set true to indicate that these calculations have been completed for the channel 3 COMP check. From the construct in figure 11

PREPDONE = B[TRUE/FALSE];

is added to the program variables.

Channel 3 COMP.—The channel 3 COMP program file (fig. 26(h)) consists of three records: one EXEC, one CONSTANT, and one VARIABLE. The name of the file is DEV1:0.DSC.DUCTSIM.SA. The executive is defined as

EXEC:COPROCES[0];

The EXEC name is COPRESS with priority 0. The variables PRD, DFFNB, and WDNA are calculated from equations (36) to (38). Then if PREPDONE is set true, the calculations can continue; otherwise

ADVISE_.M.DUCTMESS;

causes the DUCTMESS from the global data file to be printed. WDNB and DFFN are calculated from equations (39) and (40).

In this example, it is assumed that the following macros have been written and specified for the appropriate data types in the target definition file:

FUN1	multivariable function that provides table lookup and interpolation according to value of input variable
SQRT	unary function that returns square root of operand

Additionally, it is assumed that target library procedures exist for the following:

READ	reads specified ADC channels and stores values in specified variables
SAMPLE	assembles values of specified variables into argument group format
DAC1, DAC2	writes values to DAC channels

The example source files are intended to illustrate many aspects of RTMPL. However, it was impractical to develop an example that illustrated all aspects. The RTMPL utility is designed to aid the inexperienced user in becoming proficient in the language (e.g., it contains a multitude of warnings and error messages). True competence will come only with hands-on programming experience.

Chapter 7: Using the RTMPL Utility

The RTMPL utility is designed to function under a disk operating system (DOS). The DOS file and input/output handlers are used to read and write the files required by the utility. The utility provides one link in a chain of DOS services necessary to bring a simulation from concept to execution. It is assumed that the initial simulation concept has been developed. That is,

(1) The equations describing the system to be simulated have been partitioned and assigned to simulation channels. The equations for each channel have been partitioned (if required) and assigned to the COMP and PREP.

(2) The necessary real-time data analysis and input/output computations have been defined for the RTX channel.

(3) The complete set of RTMPL source files has been generated.

The source files are then processed by the RTMPL utility, producing data-base files for use by the RTMPOS utility and translated source files for the target assembler. The listing files produced by the RTMPL and assembler utilities and the results from their execution are available to the user for documenting and refining the simulation (if necessary).

The RTMPL utility is invoked by using the DOS command

RTMPL⟨DEF⟩, ⟨LIST⟩; Z = ⟨SIZE⟩

This invocation is the form used in the VERSAdos disk operating system. The user should refer to specific system documentation if another DOS is used. Although command formats may differ between installations, the information required in the command line is generic (with the possible exception of SIZE). DEF is the identification of the simulation control file and LIST is the name of the file designated to receive the listing. Both of these files may be devices (the former being an input/output device, such as the user's terminal, and the latter an output device, such as a printer). SIZE is a designation used by the DOS in assigning memory segments for utility use. The amount of memory required depends on the size of the simulation and must be determined by the user. Generally 500K bytes are sufficient for a typical simulation.

After the command has been invoked, the utility will read the simulation control file. If the file has been assigned a device name (e.g., "#", representing the user's terminal), the utility will prompt the user for the required information. The simulation control file, DUALSIM, for the dual-nozzle example is given in figure 26(a), where the entries correspond to the utility control segment structure in figure 7 and the options listed in table I.

For the dual-nozzle example the RTMPL invocation

RTMPL DUALSIM, #PR; Z = 100

will process the source files of figure 26 according to the simulation control file, DUALSIM, and will provide a listing file on the printer device (#PR). It reserves 100K bytes of memory for RTMPL use. In this case the input and output files are defaulted to the user's terminal.

In the resulting user-terminal display (fig. 27) the RTMPL header is followed by the RTMPL interpretation of the simulation control file. The general format for each line is

⟨PROMPT⟩ = ⟨TERMINAL ENTRY⟩; ⟨RETURN⟩****
⟨file entry⟩

If the user terminal, rather than the DUALSIM file, had been identified as the simulation control file in the RTMPL invocation, the display would pause after " = " for user entry at the terminal. After each "*****", the RTMPL interpretation of the entry is displayed. Any detected entry errors are displayed after the entry.

After all entries have been processed, the total number of errors detected is displayed. If any errors have been detected, RTMPL processing is aborted. Otherwise utility execution pauses at this point (for 15 sec) to allow the user to review the simulation definition. If the user does not intervene (e.g., to redefine the simulation), the RTMPL utility processes the simulation source files.

```
RTMPL TRANSLATOR      VERSION 1.00
NASA LEWIS RESEARCH CENTER

PROCESSING RTMPL SET UP FILE ...
OPTIONS=
**** NONE
SIM.ID=
**** DUALNOZZ
SIM.DESCR=
**** RTMPL*TEST*CASE
SIM.ORIGN=
**** DALE*J.*ARPASI
SIM.SRCEVOL=
**** DEV1
SIM.OBJVOL=
**** DEV1
SIM.DBVOL=
**** DEV1
SIM.UNUM=
**** 0
SIM.NCH=
**** 3
CHANNEL[1] (<TYPE>.<ID>) =
**** RTX.DATAPROC
CHANNEL[2] (<TYPE>.<ID>) =
**** DSC.CORESIM
CHANNEL[3] (<TYPE>.<ID>) =
**** DSC.DUCTSIM
GLOBAL FILE (GLOBAL.<ID>)=
**** GLOBAL.DUALNOZZ
TARGET FILE #1 (M68000.<ID>)=
**** M68000.MACHCHAR
NO SETUP ERRORS ENCOUNTERED
```

Figure 27. - User terminal display (dual-nozzle simulation control file).

```
PROCESSING DUALNOZZ GLOBAL DEFINITIONS...
PROCESSING DATAPROC(COMP) PROGRAM DEFINITIONS...
PROCESSING DATAPROC(PREP) PROGRAM DEFINITIONS...
PROCESSING CORESIM(COMP) PROGRAM DEFINITIONS...
PROCESSING CORESIM(PREP) PROGRAM DEFINITIONS...
PROCESSING DUCTSIM(COMP) PROGRAM DEFINITIONS...
PROCESSING DUCTSIM(PREP) PROGRAM DEFINITIONS...
PROCESSING DATAPROC(COMP) ARGUMENT GROUPS...
PROCESSING DATAPROC(PREP) ARGUMENT GROUPS...
PROCESSING CORESIM(COMP) ARGUMENT GROUPS...
PROCESSING CORESIM(PREP) ARGUMENT GROUPS...
PROCESSING DUCTSIM(COMP) ARGUMENT GROUPS...
PROCESSING DUCTSIM(PREP) ARGUMENT GROUPS...
PROCESSING DATAPROC(COMP) PROGRAM STATEMENTS...
PROCESSING DATAPROC(PREP) PROGRAM STATEMENTS...
PROCESSING CORESIM(COMP) PROGRAM STATEMENTS...
PROCESSING CORESIM(PREP) PROGRAM STATEMENTS...
PROCESSING DUCTSIM(COMP) PROGRAM STATEMENTS...
PROCESSING DUCTSIM(PREP) PROGRAM STATEMENTS...
GENERATING SIMULATION TRANSFER MAPS...
GENERATING ASSEMBLER SOURCE FILE: DEV1:0000.OBJCOMP.DATAPROC.SA
GENERATING ASSEMBLER SOURCE FILE: DEV1:0000.OBJPREP.DATAPROC.SA
GENERATING ASSEMBLER SOURCE FILE: DEV1:0000.OBJCOMP.CORESIM.SA
GENERATING ASSEMBLER SOURCE FILE: DEV1:0000.OBJPREP.CORESIM.SA
GENERATING ASSEMBLER SOURCE FILE: DEV1:0000.OBJCOMP.DUCTSIM.SA
GENERATING ASSEMBLER SOURCE FILE: DEV1:0000.OBJPREP.DUCTSIM.SA
ESTABLISHING DATA-BASE FOR DUALNOZZ SIMULATION ...

DATA BASE AND ASSEMBLER SOURCE FILE(S) GENERATED
```

Figure 28. - User terminal display (dual-nozzle simulation processing).

As shown in figure 28, each major step in processing the simulation source files is displayed on the terminal. The global data file GLOBAL.DUALNOZZ is processed first. At this point the specified listing file #PR is opened. The global data file is then read. If errors are detected in the file structure, RTMPL processing is aborted. Otherwise each record in the global data file is syntactically tested and transferred to the simulation data base being established by the utility. All messages, tasks, and global constants are thereby established for later reference by the program files. If any syntactical errors are detected, RTMPL processing is aborted. Otherwise the utility processes each program source file. Again, file structure is tested. If the structure is correct, the utility processes the local data segment definitions (i.e., variables and constants) and establishes these definitions in the data base. It also sets up executive and task definitions in the data base in preparation for statement translation. After these actions have been taken for each program, and if no errors have been detected, the utility rereads the programs to process argument group definitions. Again, the utility will abort processing if any errors are detected. If no errors have been detected, the utility begins processing the source statements contained in the execution segment of each source program.

Statement processing consists of syntax and semantic testing in conjunction with parsing of the expression operations and operands. The parsed expression is then translated into assembly language macros for inclusion in the object file. If the SCAN option has been selected in the control file, RTMPL processing is complete. If the SCAN option has not been selected, and all statements in all source files have been processed without error (errors cause RTMPL to abort), the RTMPL object files (assembler source and data base) are generated.

Chapter 8: RTMPL Listing

The RTMPL utility provides an extensive listing designed to aid the user in developing accurate time-optimized simulations. The listing also provides the documentation necessary to track simulation development and to allow engineering level interactive execution. The listing is generated concurrently with the processing of the simulation source files. Therefore, if the utility aborts because of source file errors, sufficient information should be available in the listing to aid in correcting the errors.

The listing consists of two major parts—scan and documentation. These parts are further divided into a number of phases. The parts and phases are discussed here in terms of their relationships to simulation development and documentation. The discussion will use the dual-nozzle example listing (appendix A) as illustration. References to that listing include the listing page number, which appears on the the header line on each page. The header line also lists the simulation name (DUALNOZZ) and the date and time of the listing generation.

Scan Listing

This part of the listing provides the user with information obtained during syntactic and semantic verification of the simulation by the utility. It has two phases: the scan of the data segments, and the scan of the execution segments. The second phase is obtained only if no errors are recorded in the first phase.

Listing pages 1 to 3 in appendix A show the results of the dual-nozzle data segment scan. The global messages and constants are scanned first. The listing of global operational tasks is included to service future extensions of RTMPL and should be ignored. After the global data are scanned, the local constants and variables in each program file are processed. (The execution segments (executives and tasks) are also identified although their statements are not processed until the next scan phase.) Finally the argument groups in each source program are processed.

The simulation example contains no errors. If errors were detected, they would be listed in the appropriate segment of the scan in a self-descriptive format. All RTMPL error messages are listed in appendix B. The general error and warning format is

⟨SOURCE DEFINITION⟩

⟨ERROR!⟩ ⟨MESSAGE⟩ ⟨specifics⟩

The offending source definition is listed verbatim. The errors are then listed; the designation (ERROR!) is followed by a message and specifics. The message describes the offense in general terms. The specifics relate the message to the specific part of the definition that is causing the problem. For example, the constant definition

```
CONSTANT:
ABC?DEF12 = S1/0, 2 [2., O.L] EOR;
```

would produce the following error listing in the scan:

```
ABC?DEF12 = S1/0, 2 [2., O.L.];

ERROR! NAME EXCEEDS 8 CHARACTERS;
ABC?DEF12

ERROR! NONALPHANUMERIC CHAR(S) IN
NAME: ABC?DEF12

ERROR! NUMBER TOO LARGE FOR SCALE
FACTOR: 2

ERROR! ILLEGAL CHAR IN INTEGER: L
```

In this example the constant name exceeds eight characters and contains an nonalphanumeric character. The scale factor (2°) is not large enough to scale the specified value, 2. Finally the use of an alphabetic character (L) in the integer value was flagged as an error.

For the sake of clarity some messages will contain specifics within the message. For example,

ERROR! UNDEFINED FOREGROUND EXEC (FOREEXEC) IN PROGRAM

where "FOREEXEC" and "PROGRAM" would be specific source program names. The second phase of the scan is the processing of the statements that are in the execution segment of each source program. This phase is illustrated on listing pages 4 to 6. The results for each executive and task in each source program are listed. Again no errors were encountered in the example. However, a multitude of warning messages were issued. Their format is similar to that for error messages. All RTMPL warnings are listed in appendix B.

Warnings are included in the listing to advise the user of potential problems or to suggest possible changes to the source programs that could reduce the time or improve the accuracy of the computations. On listing page 4, statement 2 of the GETAN executive causes the warning

WARNING! AN MAY NOT BE COMPUTED YET (PAST VALUE USED)

This occurs because the utility has not detected AN as the result (i.e., appearing on the left) of a previous equivalence statement. In this case, AN was derived by using the target procedure READ and the user would simply ignore the warning. Statement 9 in the GETAN executive causes the following to be issued:

WARNING: CONSTANT RESCALING ENCOUNTERED:K1P049 RSF = 11 NSF = 1

*** CREATED SC$1 FOR RESCALING OF K1P049

The nominal scale factor (NSF) is that assigned to a variable or constant by the user in the data segment. The required scale factor (RSF) is that required to make the resulting scale factor of an expression compatible with the required scale factor of the statement containing the expression. This warning indicates that the constant K1P049 requires a scale factor of 2^{11} (scale factor on ANF) instead of the specified 2^{1}. No user action on this warning is required since a system constant (SC$1) is automatically created by the utility with the proper value and scale factor. Note that a similar warning is issued for statement 2 of the MAINSIM executive (listing page 5). In this case, however, a system constant is not created by the utility since P0 has been defined as a global parameter. The utility will scale as required to produce the specified result. The user could, however, improve the computational speed (eliminate an internal scaling operation) by redefining the nominal scale factor assigned to P0.

Statement 2 of the MAINSIM executive also produces the warning

WARNING! CONSTANT PRECISION ADJUSTMENT:P0 RPRC = 2 NPRC = 1

The nominal precision (NPRC) is that assigned to a variable or constant by the user in the data segment. The required precision (RPRC) is that required to make the resulting precision of an expression compatible with the required scale factor of the statement containing the expression. This warning is issued because the divide macro, selected from the target definition files, requires a double-precision operand but P0 was defined as a single-precision parameter. The utility will insert the proper precision conversion macro. However, calculation time will be reduced and perhaps accuracy will be improved if the user redefines P0 as a double-precision parameter.

A final illustration of RTMPL warnings is shown for statement 23 of the MAINSIM executive.

WARNING! MULTIPLE ASSIGNMENT OF JOBDONE

is issued to indicate that the variable JOBDONE has appeared more than once on the left side of an equivalence (statement 1 also). In this case the dual assignment was intentional and the warning would be ignored.

Documentation Listing

Documentation, the second part of the listing, is provided if RTMPL processing has not encountered errors during scan. This part is made up of a number of phases, depending on the number of source programs in the simulation. They are

(1) General simulation information
(2) Global data segment
(3) Local data segment
(4) Executable statement segment

Phases 3 and 4 are repeated for each source program.

In the first phase (listing pages 7 to 9) information contained in the control segment is listed and all files used during utility operation are identified. The source, target, and object files are listed. The object files consist of assembler source files and data-base files. Each assembler source file corresponds to an RTMPL program file. Two types of data-base files are identified: global and program specific. The latter are produced for each source program. The number of records in each file is also

identified. These files need not concern the general user since they are used at run time by the RTMPOS operating system (see the section Data-Base Files).

The global data segment listing is shown on listing pages 10 to 12. The operational messages are identified and listed. The global constants are identified as well as their data type and precision, values, scale factors, if any, and whether or not the constant is parametric. If a constant is multivalued, the values are tabulated below the main identification table. The first value in the multivalue table for PRXVALS is read as "one value at zero."

The last table in the global data segment listing identifies the transfer maps for data transmission in the simulation. The table specifies the source channel and processor and the variable name and destination channel. If a variable is only transferred to the other processor in its channel, that is designated "local." The table further specifies the transfer address in the destination channel's external memory and the address where the transfer map is stored. Finally it specifies the data path to be used to implement the transfer.

The local data segment for the COMP in the DATAPROC channel is given on listing pages 13 to 14. Other local data segments are given on subsequent listing pages to illustrate this listing phase. During this phase the characteristics of local variables, external variables, local constants, arguments, executives, and tasks are tabulated.

Local variables are listed in terms of name, data type and precision, values, and scale factor, if any (e.g., listing page 16). Additionally the entry under the XREF header will indicate "YES" if the variable is referenced in another program. The entries under the PVAL header indicate the number of past values associated with the variable. Finally the absolute location of the variable is given. Note that in the DATAPROC (COMP) listing no user-defined variables are listed (listing page 13). Four target state variables are listed. All target state variables defined in the target definition files will always appear in this table. As is true with all variables and constants, an asterisk preceding the name indicates that the item is not used in the execution segment of the simulation.

Variables external to the local program are identified in terms of name and assigned location. The "$0" appendage to the name indicates that the current value is used.

Local constants (listing page 16) are tabulated in a format similar to that for global constants. In addition, the size (arrayness) and assigned location of the constant are provided. If a local constant is defined globally, it will be so indicated in the value entry. If the value of a constant is used as an immediate data operand only, it will have no location assigned. In this case "IM-DATA" will appear as its location entry.

Argument groups (listing page 16) are tabulated in terms of name, data type and precision, location, maximum size, number of programmed entries, and an item list. The item list shows each entry to be initially contained in the group. The list may of course be changed at run time. For each entry the type and name are given. The types XV, LV, and CN indicate external variable, local variable, and constant, respectively.

Tasks (listing page 17) are tabulated in terms of name, initial enable latch setting, reenterability, amount of scratch pad memory required, and the locations of the external variables required if the task is referenced in the DISPATCH command.

Executives (listing page 17) are tabulated in terms of name, priority level, service tasks, and scratch pad memory requirements. Three types of scratch pad memory (exec, task, macro) are defined. These need not concern the user: they indicate the amount of temporary memory required to compute the executive proper and its service tasks and macros.

An execution segment listing is illustrated on listing pages 18 and 19. It is the listing for the DATAPROC PREP. The executable statements of each task and executive in the program are interpretively listed to help ensure that the utilities interpretation of the program corresponds to the user's intention. The general listing format is

⟨STATEMENT NUMBER⟩ ⟨OBJECT LABEL⟩ ⟨STATEMENT⟩ ⟨CALCULATION TIME⟩

The statement number is relative to its location in the task or executive. The object label is either the statement label assigned in the source or a sequential utility assignment. Utility label assignment is sequential within a program (as opposed to statement numbers, which are sequential within a record). The utility label, S$N, would be assigned to the n^{th} statement in a program if it was not labeled by the user. The statement, as listed, is an edited version of the corresponding source file statement. Unnecessary constant delimiters (semicolon, underscore, exclamation point) are removed to enhance readability. Furthermore statements within conditionals are indented proportionally to the conditional level to aid in identifying the structure of the program. Heavily nested conditionals can be easily identified.

The calculation time listed to the right of the statement is an estimate (in machine cycles) of how long it will take to compute the statement on the target processor. This number is obtained by adding the calculation time (from target definition files) of each macro used in generating the assembly source for the statement. To convert this number to seconds, the user should multiply it by the cycle time of the target processor. The "MAX PATH EXECUTION TIME" appearing after the listing of the

statements is the number of cycles estimated for the calculation executive or task when the maximum time path is taken through the conditional statements. Note that in the case of the GETAN. executive the user should add the times for the library subroutine READ and the tasks JOBDONE., WRTACN., and WRTADN. to get the total estimated calculation time of GETAN.

Finally the execution segment phase of the listing lists both the variables computed locally and transferred to other programs and those computed externally and referenced as operands in the local program. This listing is handy when accounting for data transfer times in partitioning the simulation.

The RTMPL listing file was designed to aid the user in creating efficient simulation code. An attempt has been made to have the utility generate meaningful messages to facilitate debugging and program optimization. The information in the listing file, coupled with that in the RTMPL object files, provides a comprehensive description of the simulation.

Chapter 9: RTMPL Object Files

If no errors are encountered during translation of the RTMPL source files and the scan option is not selected, the RTMPL utility will generate two sets of object files—data-base files and assembler source files. The data-base files describe all elements in the simulation. These files are compatible with the RTMPOS operating system and furnish it with the information necessary to allow the user to interactively execute the simulation on the target simulator. The assembler source files provide a fully coded assembly language source program for each processor in the simulation. These files must be assembled on the target macro assembler and linked to form load modules. The load modules are then available for loading through RTMPOS at run time. This section describes these files from a user's standpoint. For illustration the assembler source files for the DATAPROC channel of the dual-nozzle simulation are listed in appendix C. No illustrations of the data-base files are provided since these are not text files. Note that the format of the assembler source files depends on the requirements of the target processor assembler as specified in the target definition files. The files in appendix C were generated for the MC68000 assembler.

Assembler Source Files

The assembler source files are text files and may be listed by using the DOS. The resource name string used to identify an assembler source file contains the following components:

VOLUME ID	object volume name specified in simulation definition file
USER NUM	user number specified in simulation definition file
CATALOG ID	"OBJCOMP" or "OBJPREP" depending on whether the source program is a COMP or PREP program
FILE NAME	logical name assigned to source program channel
EXTENSION	"SA," indicating a text file

The assembler source file names for the dual-nozzle simulation are

DEV1:0.OBJPREP.DATAPROC
DEV1:0.OBJCOMP.DATAPROC
DEV1:0.OBJPREP.CORESIM
DEV1:0.OBJCOMP.CORESIM
DEV1:0.OBJPREP.DUCTSIM
DEV1:0.OBJCOMP.DUCTSIM

The OBJCOMP.DATAPROC and OBJPREP.DATAPROC files are listed in appendix C. The line numbers given are for listing purposes only and they are referenced in the following discussion of the example.

The assembler source files consist of statements and comments. Comments are denoted by an asterisk in the first character location in the line. The comments are ignored by the assembler. Each statement is broken down into the following fields:

LOCATION	OPERATION	OPERAND	COMMENT
(8 characters)	(8 characters)	(variable)	(remainder)

Each field is separated by one or more space characters. The location field contains mnemonic descriptors of the statement that are used for memory referencing. If referencing is not required, this field is filled with spaces. The operation field contains a macro name. The target code defined for that name will be substituted by the assembler for that name in the target macro file. The macro names used in the dual-nozzle simulation are defined in table VI. The available macros would be defined in the systems manual generated for the target simulator during RTMPL installation. The operand field contains the operands to be used by the macro. Multiple

TABLE VI.—DESCRIPTION OF MACROS USED IN DUAL-NOZZLE SIMULATION

Macro	Description
ACTIVAT$	Implement ACTIVATE
ADD$S1RI	Add immediate data to register (DTP = S1)
ADVISEH$	Implement ADVISE.H
ADVISEM$	Implement ADVISE.M
ADVISER$	Implement ADVISE.R
BACKEXEC	Executive initialization
CALL$	Implement CALL
CVP$S12	Convert precision (1→2)
CVP$S21	Convert precision (2→1)
DC$	Define constant
DISPATC$	Implement DISPATCH
DIV$S1RM	Divide register by memory (DTF = S1)
DIV$S2RM	Divide register by memory (DTP = S2)
DL$	Define 32-bit word
DN$	Define name
DS$	Reserve storage
EQU$	Define address relationships to firmware
ENTER$	Implement ENTER
EXIT$	Jump to specified argument
FUN1	Multivariable function
FOREEXEC	Executive initialization
HLD$S1	Store value for comparison
INCLUDE$	Specify macro file
JF$OVERF	Jump if no overflow
JGTH$S1	Compare and jump if greater than
JNEQ$S1	Compare and jump if not equal
JNGT$S1	Compare and jump if not greater than
JNLT$S1	Compare and jump if not less than
JTR$	Jump if expression true
LDI$S1	Load register with immediate data
LDM$BV	Load register from memory (DTP = B)
LDM$S1	Load register from memory (DTP = S1)
MUL$S2RI	Multiply register by immediate data (DTP = S2)
MUL$S2RM	Multiply register by memory (DTP = S2)
MUL$S2RR	Multiply register by register (DTP = S2)
NOT$BVR	Logical NOT
ORG$	Define load start address
REDO$	Implement REDO
RETURN$E	Return from background executive
RETURN$I	Return from foreground executive
RETURN$T	Return from task
SCL$S1L	Scale register left (DTP = S1)
SCL$S1R	Scale register right (DTP = S1)
SCL$S2L	Scale register left (DTP = S2)
SCL$S2R	Scale register right (DTP = S2)
SLV$S1	Store register in local memory
SSP$S1	Store register in scratch pad memory
STV$BV	Send value via local bus (DTP = BV)
STV$S1	Send value via local bus (DTP = S1)
STX$BV	Send value via external bus (DTP = BV)
STX$S1	Send value via external bus (DTP = S1)
SUB$S1IR	Subtract immediate data from register (DTP = S1)
SUB$S1RM	Subtract memory from register (DTP = S1)
SUB$S1RR	Subtract memory from register (DTP = S2)
SUB$S2RR	Subtract memory from register (DTP = S2)
SQRT	Unary function
TSTXVA$	Test currency from alternate processor
TSTXVL$	Test currency from local bus
XREF	External reference

operands are separated by commas. In the example listings, "Dn" denotes the n^{th} data register and "An" denotes the n^{th} address register. Register mnemonics are target dependent. The comment field is not processed by the assembler and should be self-explanatory.

Each assembler source file is arranged as follows:

Required target macros (e.g., lines 9 to 176, OBJCOMP.DATAPROC)

Control and initialization (e.g., lines 177 to 181, OBJCOMP.DATAPROC)

Foreground executive maps (e.g., lines 182 to 203, OBJCOMP.DATAPROC)

Entry addresses (e.g., lines 204 to 210, OBJCOMP.DATAPROC)

Simulation transfer maps (e.g., lines 211 to 273, OBJCOMP.DATAPROC)

Transfer memory (e.g., lines 274 to 362, OBJCOMP.DATAPROC)

Local variables (e.g., lines 502 to 509, OBJPREP.DATAPROC)

Program constants (e.g., lines 510 to 513, OBJPREP.DATAPROC)

Dispatch task list (e.g., lines 514 to 518, OBJPREP.DATAPROC)

Argument groups (e.g., lines 519 to 572, OBJPREP.DATAPROC)

Executable segment (e.g., lines 573 to 641, OBJPREP.DATAPROC)

Target procedures (e.g., lines 642 to 647, OBJPREP.DATAPROC)

The following paragraphs describe these components of the assembler source file.

The target macros define the assembly language equivalents for all RTMPL-generated macros in the program. These include assembly instruction macros such as INITIAL, DATASEG, and DATAEND (lines 9 to 52, OBJCOMP.DATAPROC), RTMPL data transfer operation and command macros (lines 53 to 151, OBJCOMP.DATAPROC), and target library procedures referenced through the use of the CALL command or internally by other macros (lines 152 to 175, OBJCOMP.DATAPROC). The structure and content of these macros are arbitrarily set up by systems programmers to meet specific target simulator requirements.

The first statements to be assembled in every RTMPL-generated program are the control and initialization statements (INITIAL$ and DATASEG$). These statements set up the program for assembly by defining absolute interfaces with the simulator hardware and resident software and by furnishing any other initialization required by the target assembler.

The foreground executive maps are used by the processor's firmware to service the ACTIVATE

command that appears in the program for the alternate processor in the channel. The first location is reserved to hold the entry address of the active executive. All foreground executives are then listed in decreasing order of priority. For each, the entry address is followed by a busy flag and a pending flag. The busy flag is set by the firmware if that executive is being executed. The pending flag is set if the execution of that executive is being delayed due to the execution of a higher priority foreground executive.

The entry address part of the program contains the entry addresses of all executives and tasks in the program. They are used for various operating system functions at run time.

The simulation transfer maps are used for interprocessor data transmission by the STV$/STX$ macros. They list the destination channels for each transfer variable in the simulation. The format is

Destination channel codes (none, 0, 4, 8, ...)
Expansion (−2)
End of map (−1)

A destination channel code is included for every channel requiring reception of the variable ("0" representing the first channel in the simulation, "4" representing the second, etc.). Since the value of a transfer variable is always stored in the transfer and external memory of the local channel, no destination code is required for the local channel. Expansion (−2) is included to allow the addition of another destination channel to the map at run time. The feature supports the run-time manipulation of argument group entries. The end of the map is signified by −1.

The transfer memory allocation details the variables assigned to transfer and external memory. This allocation as well as the transfer maps is global. Therefore the allocations are identical in all assembler source files in the simulation. Each transfer memory allocation consists of a "CALC FLAG" (set appropriately by the firmware to indicate value currency), an initial-value assignment, and the variable's transfer map address. These allocations are used by the TSTXVS$/TSTXVL$/TSTXVA$ macros to test the currency of data transfer.

Local variable assignment details the memory assignments of all program variables not assigned to either transfer memory or external memory. Program constant assignment details the memory assignments of all program constants. The dispatch task list provides the argument for the DISPATCH command and lists the tasks to be dispatched.

The argument group is represented in the assembler source file by a translation of the ARGGROUP construct. It contains the information supplied by the user in the RTMPL source program formatted into a data record compatible with both the argument requirements of target library procedures and the data-handling requirements of the RTMPOS operating system. In the OBJPREP.DATAPROC listing, lines 523 to 526 reserve memory for record identification information supplied by RTMPOS. This information identifies the record when it is downloaded into a run-time-generated disk file. It contains, among other things, the name of the argument group and the channel. Line 527 specifies the maximum number of items that can be contained in the group. Lines 528 and 529 are for RTMPOS record-keeping. Line 530 specifies the current number of items contained in the group. Line 531 specifies the number of words needed for a value of a group item. Following this are the addresses of all items presently contained in the group and memory reservation for the addresses of items that may be added at run time. Finally memory is reserved for values of each item in the group (line 533). These values are determined by the calling procedure using the group as an argument.

Each target library procedure, invoked by the CALL command, that uses the group as an argument may handle the information described differently. For example, the read procedure invoked in line 579 uses the item addresses in ADCCHN to obtain ADC channel numbers and the addresses in ADCVAR to determine where the data are stored. This procedure does not use the memory reserved for values at all. The sample procedure invoked in OBJCOMP.DATAPROC obtains values from the specified items' addresses and stores them in the value locations reserved within the data argument group. It is therefore necessary that the user understand the action of all target library procedures invoked.

The executable code segment contains the sequence of assembly language macros corresponding to the executable statements of the source file. This code is separated into executives and tasks as specified in the source file. Each block of code contains overhead information necessary for execution. Executive overhead is as follows:

(1) Macro scratch pad allocation—memory reserved for scratch pad required internally by macros resident in either the executive or its service tasks

(2) Task scratch pad allocation—memory reserved for scratch pad required for translation of service tasks (reserving task scratch pad here allows tasks to be reenterable)

(3) Executive scratch pad allocation—memory reserved for scratch pad required for translation of the executive

(4) Entry overhead—memory reserved for register and miscellaneous storage necessary to enter and return from the executive

Memory is allocated by using one of the housekeeping macros BACKEXEC or FOREEXEC.

An example of task overhead is given in lines 415 to 425 of OBJCOMP.DATAPROC. First, the addresses of all external variables referenced in the task are listed, followed by the number of these variables. These overhead items are used in executing task DISPATCH command. The task overhead continues with the location of the task enable and complete latches (see ENABLE and DISABLE commands). Finally, a task entry overhead of two memory words is reserved and assigned the task name. Task execution begins at line 412.

Finally, as shown in lines 645 to 647 of OBJPREP.DATAPROC, all referenced library procedures are invoked by RTMPL. This causes the procedures to be attached to the end of the program and simplifies the program linking process.

Note that the example programs shown in appendix C were targeted to the MC68000 macro assembler. For other processors the basic program structure would be the same, but the macro content could be significantly different.

Data-Base Files

Data-base files are not text files and therefore may not be listed by using a standard DOS list command. They are files of Pascal records. Their format should be of no concern to the user (for more information, see ref. 3). Both RTMPL and RTMPOS contain facilities for listing the information contained in the data-base files in a usable form. All pertinent data-base information is given in the RTMPL listing.

The data-base files generated for the dual-nozzle simulation are identified on listing pages 7 to 9 in appendix A. The catalog name (e.g., SIMDEF) indicates the type of information contained in the file. Global data-base files contain information pertaining to the simulation as a whole. Program-specific data-base files contain information pertaining to a particular program.

Six global data-base files may be generated for a simulation. The SIMDEF file always contains one record and roughly corresponds to the simulation control segment. The MESSDEF file contains the global messages. The VALUEDEF file contains all values referenced in the simulation. The GLCDEF file contains the global constants. The OSTSKDEF file is not currently used but is reserved for RTMPL expansion. The PRGDEF file defines each program in the simulation in general terms. Note that, in the listing, if the number of records in a file is 0, the file was not generated for the simulation.

Up to eight program-specific data-base files may be generated for each program in a simulation. LVAR, XVAR, CNST, and AGRP files define the program's local variables, external variables, constants, and argument groups, respectively. The ALST file contains the items assigned to each argument group. The EXEC and TASK files provide pertinent information about the program's execution segment. The TKLB file is used to support the DISPATCH command.

The contents of these files may be modified to some extent by the user, through RTMPOS, to form a run-time data base. Copies of the run-time data base may be saved and used for different simulation starting conditions and to support simulation documentation (ref. 3).

Chapter 10: Concluding Remarks

The real-time multiprocessor programming language (RTMPL) combines the efficiency and versatility of assembly language programming with the advantages of having an easily understood, engineering-oriented, high-level programming language. RTMPL is intended for time-critical applications (e.g., real-time simulation) and is suitable for programming the following simulator configurations:

1. Multiprocessors with dual-bus communication
2. Multiprocessors with single-bus communication
3. Multiprocessors with shared-memory communication
4. Single processor

RTMPL is a structured language that provides many program-development aids to help in producing time-efficient code.

The language is targetable, not only to various simulator configurations, but also to various processors and macroassemblers. A targeting utility is available to automate the targeting procedures. The power of a macro-based language is determined by the assembly language macros written to support it. Multivariable and unary functions, firmware interfacing commands, and target library procedures have been incorporated in RTMPL. Other macros may be incorporated within the language structure to meet specific installation requirements. Since RTMPL supports both fixed-point (scaled fraction) and floating-point data types, operations on these data types can be incorporated (or not) to meet installation needs. For example, if a simulator includes an efficient floating-point processor, macros to support fixed-point operations might not be needed at all.

The versatile targeting capability of RTMPL eliminates the need for designing special compilers or translators. If a new processor and its companion macroassembler fall within the targeting restrictions of the language, RTMPL can easily be targeted to this processor, thereby saving many hours of software development.

RTMPL, coupled with its companion operating system, RTMPOS, provides for interactive execution of multiprocessor simulators at a level comparable to analog computers. Data-base files are generated by the RTMPL utility for use by RTMPOS. This provides an extensive engineering-level interface between the users and the simulation at run time. Listings are provided to establish the dialog and to summarize and document the characteristics of the simulation.

As is the case with initial versions of any language, there is room for improvement in RTMPL. At this time the following enhancements are contemplated:

1. Reducing and streamlining the specifications of data type and precision and scale factor required for defining variables and constants
2. Allowing the use of direct-value specification within expressions
3. Extending the EXECUTE command to accept target-specified arguments
4. Reducing RTMPL processing time by reducing the number of target definition file reads

Since RTMPL is a research language, the implementation of these improvements will depend on user acceptance and response.

The author and other members of the staff at the Lewis Research Center have a continuing interest in improving the cost effectiveness and utilization of real-time simulation. We hope that RTMPL and other RTMPS developments will provide a vehicle for constructive discussion and development of simulation techniques and standardizations to meet these goals.

Lewis Research Center
National Aeronautics and Space Administration
Cleveland, Ohio, January 14, 1985

Appendix A—Listing for Dual-Nozzle Simulation

```
   RTMPL LISTING : DUALNOZZ  11/27/84  10:28:52                PAGE 1

* * * * * * * * * * * * * * * * * * * * * * *
*                                             *
*   SCAN - SOURCE FORMAT, DATA SEGMENTS       *
*    ...DUALNOZZ                              *
*                                             *
* * * * * * * * * * * * * * * * * * * * * * *

GLOBAL OPERATIONAL MESSAGES: DEV1:0000.GLOBAL.DUALNOZZ.SA
---------------------------------------------------------------------------

...8 MESSAGE(S)
...0 ERROR(S)

GLOBAL OPERATIONAL TASKS: DEV1:0000.GLOBAL.DUALNOZZ.SA
---------------------------------------------------------------------------

... NONE ENCOUNTERED

GLOBAL CONSTANTS: DEV1:0000.GLOBAL.DUALNOZZ.SA
---------------------------------------------------------------------------

...10 CONSTANT(S)
...0 ERROR(S)

PROGRAM CONSTANTS: DEV1:0000.RTX.DATAPROC.SA
---------------------------------------------------------------------------

... NONE ENCOUNTERED

PROGRAM VARIABLES: DEV1:0000.RTX.DATAPROC.SA
---------------------------------------------------------------------------

... NONE ENCOUNTERED

PROGRAM EXECUTIVES: DEV1:0000.RTX.DATAPROC.SA
---------------------------------------------------------------------------

...2 EXECUTIVE(S)
...0 ERROR(S)

PROGRAM TASKS: DEV1:0000.RTX.DATAPROC.SA
---------------------------------------------------------------------------

...1 TASK(S)
...0 ERROR(S)

PROGRAM CONSTANTS: DEV1:0000.RTXPREP.DATAPROC.SA
---------------------------------------------------------------------------

...6 CONSTANT(S)
...0 ERROR(S)

PROGRAM VARIABLES: DEV1:0000.RTXPREP.DATAPROC.SA
---------------------------------------------------------------------------

...2 VARIABLE(S)
...0 ERROR(S)

PROGRAM EXECUTIVES: DEV1:0000.RTXPREP.DATAPROC.SA
---------------------------------------------------------------------------

...1 EXECUTIVE(S)
...0 ERROR(S)

PROGRAM TASKS: DEV1:0000.RTXPREP.DATAPROC.SA
```

```
...3 TASK(S)
...0 ERROR(S)

PROGRAM CONSTANTS: DEV1:0000.DSC.CORESIM.SA
...5 CONSTANT(S)
...0 ERROR(S)

PROGRAM VARIABLES: DEV1:0000.DSC.CORESIM.SA
...8 VARIABLE(S)
...0 ERROR(S)

PROGRAM EXECUTIVES: DEV1:0000.DSC.CORESIM.SA
...1 EXECUTIVE(S)
...0 ERROR(S)

PROGRAM TASKS: DEV1:0000.DSC.CORESIM.SA
... NONE ENCOUNTERED

PROGRAM CONSTANTS: DEV1:0000.DSCPREP.CORESIM.SA
...4 CONSTANT(S)
...0 ERROR(S)

PROGRAM VARIABLES: DEV1:0000.DSCPREP.CORESIM.SA
...6 VARIABLE(S)
...0 ERROR(S)

PROGRAM EXECUTIVES: DEV1:0000.DSCPREP.CORESIM.SA
...1 EXECUTIVE(S)
...0 ERROR(S)

PROGRAM TASKS: DEV1:0000.DSCPREP.CORESIM.SA
... NONE ENCOUNTERED

PROGRAM CONSTANTS: DEV1:0000.DSC.DUCTSIM.SA
...4 CONSTANT(S)
...0 ERROR(S)

PROGRAM VARIABLES: DEV1:0000.DSC.DUCTSIM.SA
...5 VARIABLE(S)
...0 ERROR(S)

PROGRAM EXECUTIVES: DEV1:0000.DSC.DUCTSIM.SA
...1 EXECUTIVE(S)
...0 ERROR(S)
```

```
PROGRAM TASKS: DEV1:0000.DSC.DUCTSIM.SA
---------------------------------------
... NONE ENCOUNTERED

PROGRAM CONSTANTS: DEV1:0000.DSCPREP.DUCTSIM.SA
-----------------------------------------------
...3 CONSTANT(S)
...0 ERROR(S)

PROGRAM VARIABLES: DEV1:0000.DSCPREP.DUCTSIM.SA
-----------------------------------------------
...4 VARIABLE(S)
...0 ERROR(S)

PROGRAM EXECUTIVES: DEV1:0000.DSCPREP.DUCTSIM.SA
------------------------------------------------
...1 EXECUTIVE(S)
...0 ERROR(S)

PROGRAM TASKS: DEV1:0000.DSCPREP.DUCTSIM.SA
-------------------------------------------
... NONE ENCOUNTERED

PROGRAM ARGUMENT GROUPS: DEV1:0000.RTX.DATAPROC.SA
--------------------------------------------------
...1 ARGUMENT GROUP(S)
...0 ERROR(S)

PROGRAM ARGUMENT GROUPS: DEV1:0000.RTXPREP.DATAPROC.SA
------------------------------------------------------
...4 ARGUMENT GROUP(S)
...0 ERROR(S)

PROGRAM ARGUMENT GROUPS: DEV1:0000.DSC.CORESIM.SA
-------------------------------------------------
... NONE ENCOUNTERED

PROGRAM ARGUMENT GROUPS: DEV1:0000.DSCPREP.CORESIM.SA
-----------------------------------------------------
... NONE ENCOUNTERED

PROGRAM ARGUMENT GROUPS: DEV1:0000.DSC.DUCTSIM.SA
-------------------------------------------------
... NONE ENCOUNTERED

PROGRAM ARGUMENT GROUPS: DEV1:0000.DSCPREP.DUCTSIM.SA
-----------------------------------------------------
... NONE ENCOUNTERED
```

```
* * * * * * * * * * * * * * * * * *
*                                   *
*   SCAN - EXECUTABLE SEGMENTS      *
*     ...DUALNOZZ                   *
*                                   *
* * * * * * * * * * * * * * * * * *

MAIN. EXECUTIVE: DEV1:0000.RTX.DATAPROC.SA
-------------------------------------------------------------------

...2 STATEMENT(S)
...0 ERROR(S)

BADADC. EXECUTIVE: DEV1:0000.RTX.DATAPROC.SA
-------------------------------------------------------------------

...4 STATEMENT(S)
...0 ERROR(S)

GETDATA. TASK: DEV1:0000.RTX.DATAPROC.SA
-------------------------------------------------------------------

...3 STATEMENT(S)
...0 ERROR(S)

GETAN. EXECUTIVE: DEV1:0000.RTXPREP.DATAPROC.SA
-------------------------------------------------------------------

  2  AN=AN+K2;
       WARNING! AN MAY NOT BE COMPUTED YET (PAST VALUE USED)
  9  !!ANF=K1P049-K1P622M4*AN;
       WARNING! CONSTANT RESCALING ENCOUNTERED:K1P049   RSF=11 NSF=1
       *** CREATED SC$1 FOR RESCALING OF K1P049

...12 STATEMENT(S)
...0 ERROR(S)

WRTACN. TASK: DEV1:0000.RTXPREP.DATAPROC.SA
-------------------------------------------------------------------

...2 STATEMENT(S)
...0 ERROR(S)

WRTADN. TASK: DEV1:0000.RTXPREP.DATAPROC.SA
-------------------------------------------------------------------

...2 STATEMENT(S)
...0 ERROR(S)

JOBDONE. TASK: DEV1:0000.RTXPREP.DATAPROC.SA
-------------------------------------------------------------------

...4 STATEMENT(S)
...0 ERROR(S)

MAINSIM. EXECUTIVE: DEV1:0000.DSC.CORESIM.SA
```

```
   2  PRC=P0/PCN;
       WARNING! CONSTANT PRECISION ADJUSTMENT :P0          RPRC=2 NPRC=1
       WARNING! CONSTANT RESCALING ENCOUNTERED:P0          RSF=8 NSF=6
   5  !FFNB=SQRT(K1P-FUN1[PRXVALS,PRNVALS,XTO2857,PRC]);
       WARNING! CONSTANT RESCALING ENCOUNTERED:K1P         RSF=2 NSF=1
       *** CREATED SC$1 FOR RESCALING OF K1P
   6  ACNA=KCN*WCN*SQRT(TCN);
       WARNING! CONSTANT RESCALING ENCOUNTERED:TCN         RSF=12 NSF=13
  10  THEN←FFN=KP2588;
       WARNING! CONSTANT RESCALING ENCOUNTERED:KP2588      RSF=2 NSF=-1
       *** CREATED SC$2 FOR RESCALING OF KP2588
  15  THEN←ACN=ACNA/PCN*FFN*.P.FLC;
       WARNING! VARIABLE PRECISION ADJUSTMENT :ACNA        RPRC=2 NPRC=1
       WARNING! VARIABLE RESCALING ENCOUNTERED:ACNA        RSF=20 NSF=11
  16  ELSE←ACN=ZERO;
       WARNING! CONSTANT RESCALING ENCOUNTERED:ZERO        RSF=10 NSF=0
       *** CREATED SC$3 FOR RESCALING OF ZERO
  20  THEN←ADN=ZERO;
       WARNING! CONSTANT RESCALING ENCOUNTERED:ZERO        RSF=10 NSF=0
       *** USING SC$3 FOR RESCALING OF ZERO
  21  ELSE←ADN=DATAPROC.P.ANF-ACN;
       WARNING! VARIABLE RESCALING ENCOUNTERED:ACN         RSF=11 NSF=10
  23  JOBDONE=TRUE;
       WARNING! MULTIPLE ASSIGNMENTS OF JOBDONE

...24 STATEMENT(S)
...0 ERROR(S)

PRFCTNS. EXECUTIVE: DEV1:0000.DSCPREP.CORESIM.SA
-------------------------------------------------------------------------------
   3  PRC=P0/PCN;
       WARNING! CONSTANT PRECISION ADJUSTMENT :P0          RPRC=2 NPRC=1
       WARNING! CONSTANT RESCALING ENCOUNTERED:P0          RSF=8 NSF=6
   8  THEN←FLC=KP825;
       WARNING! CONSTANT RESCALING ENCOUNTERED:KP825       RSF=1 NSF=0
       *** CREATED SC$1 FOR RESCALING OF KP825
  10  PREPDONE=TRUE;
       WARNING! MULTIPLE ASSIGNMENTS OF PREPDONE
  12  DUCTDONE=TRUE;
       WARNING! MULTIPLE ASSIGNMENTS OF DUCTDONE

...13 STATEMENT(S)
...0 ERROR(S)

COPROCE. EXECUTIVE: DEV1:0000.DSC.DUCTSIM.SA
-------------------------------------------------------------------------------
   1  PRD=P0/PDN;
       WARNING! CONSTANT PRECISION ADJUSTMENT :P0          RPRC=2 NPRC=1
       WARNING! CONSTANT RESCALING ENCOUNTERED:P0          RSF=8 NSF=6
   4  !FFNB=SQRT(K1P-FUN1[PRXVALS,PRNVALS,XTO2857,PRD]);
       WARNING! CONSTANT RESCALING ENCOUNTERED:K1P         RSF=2 NSF=1
       *** CREATED SC$1 FOR RESCALING OF K1P
   5  WDNA=KDN*SQRT(TDN);
       WARNING! CONSTANT RESCALING ENCOUNTERED:TDN         RSF=12 NSF=13
   9  THEN←FFN=KP2588;
```

```
      WARNING! CONSTANT RESCALING ENCOUNTERED:KP2588    RSF=2 NSF=0
      *** CREATED SC$2 FOR RESCALING OF KP2588

...12 STATEMENT(S)
...0 ERROR(S)

PRFCTNS. EXECUTIVE: DEV1:0000.DSCPREP.DUCTSIM.SA
-------------------------------------------------------------------------------
  2  PRD=P0/PDN;
      WARNING! CONSTANT PRECISION ADJUSTMENT :P0         RPRC=2 NPRC=1
      WARNING! CONSTANT RESCALING ENCOUNTERED:P0         RSF=8 NSF=6
  7  THEN←FLC=KP825;
      WARNING! CONSTANT RESCALING ENCOUNTERED:KP825      RSF=1 NSF=0
      *** CREATED SC$1 FOR RESCALING OF KP825
  9  PREPDONE=TRUE;
      WARNING! MULTIPLE ASSIGNMENTS OF PREPDONE

...10 STATEMENT(S)
...0 ERROR(S)
```

```
 * * * * * * * * * * * * * * * * * * * *
 *                                       *
 *  GENERAL SIMULATION INFORMATION       *
 *    ...DUALNOZZ                        *
 *                                       *
 * * * * * * * * * * * * * * * * * * * *

DESCRIPTION : RTMPL TEST CASE
ORIGINATORS NAME : DALE J. ARPASI
USER NUMBER : 0000
SIMULATOR CONFIGURATION : DUAL
NUMBER OF CHANNELS IN SIMULATION : 3

RTMPL SOURCE FILES
******************
           DEV1:0000.RTX.DATAPROC.SA
           DEV1:0000.RTXPREP.DATAPROC.SA
           DEV1:0000.DSC.CORESIM.SA
           DEV1:0000.DSCPREP.CORESIM.SA
           DEV1:0000.DSC.DUCTSIM.SA
           DEV1:0000.DSCPREP.DUCTSIM.SA

GLOBAL SOURCE FILE
******************
           DEV1:0000.GLOBAL.DUALNOZZ.SA

TARGET FILES
************
           DEV1:0000.M68000.MACHCHAR.TD
           DEV1:0000.M68000.TGTSPCCM.TD
           DEV1:0000.M68000.TGTPRCDR.TD
           DEV1:0000.M68000.TGTOPDEF.TD
           DEV1:0000.M68000.TGTHKDEF.TD
           DEV1:0000.M68000.TGTSTDEF.TD

ASSEMBLER SOURCE FILES
**********************
           DEV1:0000.OBJCOMP.DATAPROC.SA
           DEV1:0000.OBJPREP.DATAPROC.SA
           DEV1:0000.OBJCOMP.CORESIM.SA
           DEV1:0000.OBJPREP.CORESIM.SA
           DEV1:0000.OBJCOMP.DUCTSIM.SA
           DEV1:0000.OBJPREP.DUCTSIM.SA

GLOBAL DATA BASE FILES
**********************

                RESOURCE NAME STRING          # OF RECORDS
                --------------------          -----------
           DEV1:0000.SIMDEF.DUALNOZZ.DB             1
           DEV1:0000.MESSDEF.DUALNOZZ.DB            8
           DEV1:0000.VALUEDEF.DUALNOZZ.DB         144
           DEV1:0000.GLCDEF.DUALNOZZ.DB            12
           DEV1:0000.OSTSKDEF.DUALNOZZ.DB           0
           DEV1:0000.PRGDEF.DUALNOZZ.DB             6
```

PROGRAM SPECIFIC DATA-BASE FILES: DATAPROC (COMP)
****

RESOURCE NAME STRING	# OF RECORDS
DEV1:0000.LVAR.DATAPROC.DB	4
DEV1:0000.XVAR.DATAPROC.DB	7
DEV1:0000.CNST.DATAPROC.DB	1
DEV1:0000.AGRP.DATAPROC.DB	1
DEV1:0000.ALST.DATAPROC.DB	16
DEV1:0000.EXEC.DATAPROC.DB	2
DEV1:0000.TASK.DATAPROC.DB	1
DEV1:0000.TKLB.DATAPROC.DB	1

PROGRAM SPECIFIC DATA-BASE FILES: DATAPROC (PREP)
****

RESOURCE NAME STRING	# OF RECORDS
DEV1:0000.LVARPREP.DATAPROC.DB	6
DEV1:0000.XVARPREP.DATAPROC.DB	3
DEV1:0000.CNSTPREP.DATAPROC.DB	7
DEV1:0000.AGRPPREP.DATAPROC.DB	4
DEV1:0000.ALSTPREP.DATAPROC.DB	66
DEV1:0000.EXECPREP.DATAPROC.DB	1
DEV1:0000.TASKPREP.DATAPROC.DB	3
DEV1:0000.TKLBPREP.DATAPROC.DB	3

PROGRAM SPECIFIC DATA-BASE FILES: CORESIM (COMP)
****

RESOURCE NAME STRING	# OF RECORDS
DEV1:0000.LVAR.CORESIM.DB	12
DEV1:0000.XVAR.CORESIM.DB	6
DEV1:0000.CNST.CORESIM.DB	17
DEV1:0000.AGRP.CORESIM.DB	0
DEV1:0000.ALST.CORESIM.DB	0
DEV1:0000.EXEC.CORESIM.DB	1
DEV1:0000.TASK.CORESIM.DB	0
DEV1:0000.TKLB.CORESIM.DB	0

PROGRAM SPECIFIC DATA-BASE FILES: CORESIM (PREP)
****

RESOURCE NAME STRING	# OF RECORDS
DEV1:0000.LVARPREP.CORESIM.DB	10
DEV1:0000.XVARPREP.CORESIM.DB	1
DEV1:0000.CNSTPREP.CORESIM.DB	13
DEV1:0000.AGRPPREP.CORESIM.DB	0
DEV1:0000.ALSTPREP.CORESIM.DB	0
DEV1:0000.EXECPREP.CORESIM.DB	1
DEV1:0000.TASKPREP.CORESIM.DB	0
DEV1:0000.TKLBPREP.CORESIM.DB	0

PROGRAM SPECIFIC DATA-BASE FILES: DUCTSIM (COMP)
****

RESOURCE NAME STRING	# OF RECORDS
DEV1:0000.LVAR.DUCTSIM.DB	9
DEV1:0000.XVAR.DUCTSIM.DB	3
DEV1:0000.CNST.DUCTSIM.DB	12
DEV1:0000.AGRP.DUCTSIM.DB	0
DEV1:0000.ALST.DUCTSIM.DB	0
DEV1:0000.EXEC.DUCTSIM.DB	1
DEV1:0000.TASK.DUCTSIM.DB	0
DEV1:0000.TKLB.DUCTSIM.DB	0

PROGRAM SPECIFIC DATA-BASE FILES: DUCTSIM (PREP)
****

RESOURCE NAME STRING	# OF RECORDS
DEV1:0000.LVARPREP.DUCTSIM.DB	8
DEV1:0000.XVARPREP.DUCTSIM.DB	0
DEV1:0000.CNSTPREP.DUCTSIM.DB	11
DEV1:0000.AGRPPREP.DUCTSIM.DB	0
DEV1:0000.ALSTPREP.DUCTSIM.DB	0
DEV1:0000.EXECPREP.DUCTSIM.DB	1
DEV1:0000.TASKPREP.DUCTSIM.DB	0
DEV1:0000.TKLBPREP.DUCTSIM.DB	0

```
* * * * * * * * * * * * * *
*                         *
*  GLOBAL DATA SEGMENT    *
*    ...DUALNOZZ          *
*                         *
* * * * * * * * * * * * * *
```

DUALNOZZ OPERATIONAL MESSAGES
****

NAME	OP-SYS INTERUPT MESSAGE
ADCMESS2	SIMULATION HALT! ADC VALUE OF AN > MAXAREA
ADCMESS1	SIMULATION HALT! ADC VALUE OF AN < MINAREA
COREMES4	CORESIM PRE-PROCESSING DELAY ENCOUNTERED
COREMES3	DUCTSIM OR ADC DELAY ENCOUNTERED
COREMES2	SIMULATION HALT! PRD OVERFLOW
COREMES1	SIMULATION HALT! PRC OVERFLOW
DPMESS	COMPUTATION CYCLE COMPLETE
DUCTMESS	DUCTSIM PRE-PROCESSING DELAY ENCOUNTERED

DUALNOZZ GLOBAL CONSTANTS
****

NAME	TYP	VALUE	SCAL FAC	PRMTR
FALSE	B	0.00000000E+000	NONE	NO
P0	S1	1.47000000E+001	B+6	YES
PCN	S1	1.47000000E+001	B+7	YES
PDN	S1	1.47000000E+001	B+7	YES
PRNVALS	I1	2.10000000E+001	NONE	NO
PRXVALS	S1	MULTI VALUES	B+1	NO
TCN	S1	9.00000000E+002	B+13	YES
TDN	S1	9.00000000E+002	B+13	YES
TRUE	B	1.00000000E+000	NONE	NO
WCN	S1	0.00000000E+000	B+8	YES
XT02857	S1	MULTI VALUES	B+1	NO
XT07143	S1	MULTI VALUES	B+1	NO

DUALNOZZ MULTIVALUED GLOBAL CONSTANTS
****

PRXVALS			PRXVALS (CONT.)		
1	@	0.00000000E+000	1	@	4.50000000E-001
1	@	5.00000000E-002	1	@	5.00000000E-001
1	@	1.00000000E-001	1	@	5.50000000E-001
1	@	1.50000000E-001	1	@	6.00000000E-001
1	@	2.00000000E-001	1	@	6.50000000E-001
1	@	2.50000000E-001	1	@	7.00000000E-001
1	@	3.00000000E-001	1	@	7.50000000E-001
1	@	3.50000000E-001	1	@	8.00000000E-001
1	@	4.00000000E-001	1	@	8.50000000E-001

DUALNOZZ MULTIVALUED GLOBAL CONSTANTS (CONTINUED)
****

PRXVALS (CONT.)

1 @ 9.00000000E-001
1 @ 9.50000000E-001
1 @ 1.00000000E+000

XT02857

1 @ 0.00000000E+000
1 @ 4.24900000E-001
1 @ 5.18000000E-001
1 @ 5.81600000E-001
1 @ 6.31400000E-001
1 @ 6.73000000E-001
1 @ 7.08900000E-001
1 @ 7.40900000E-001
1 @ 7.69700000E-001
1 @ 7.96000000E-001
1 @ 8.20300000E-001
1 @ 8.43000000E-001
1 @ 8.64200000E-001
1 @ 8.84200000E-001
1 @ 9.03100000E-001
1 @ 9.21100000E-001
1 @ 9.38200000E-001
1 @ 9.54600000E-001
1 @ 9.70300000E-001
1 @ 9.85500000E-001
1 @ 1.00000000E+000

XT07143

1 @ 0.00000000E+000
1 @ 1.17700000E-001
1 @ 1.93100000E-001
1 @ 2.57900000E-001
1 @ 3.16600000E-001
1 @ 3.82000000E-001
1 @ 4.23200000E-001
1 @ 4.72400000E-001
1 @ 5.19700000E-001
1 @ 5.65300000E-001
1 @ 6.09500000E-001
1 @ 6.52400000E-001
1 @ 6.94300000E-001
1 @ 7.35100000E-001
1 @ 7.75100000E-001
1 @ 8.14200000E-001
1 @ 8.52700000E-001
1 @ 8.90400000E-001
1 @ 9.27500000E-001
1 @ 9.64000000E-001
1 @ 1.00000000E+000

DUALNOZZ TRANSFER MAPS
****

SOURCE PROCESSOR	TRANSFER VARIABLE	DEST. CHANNEL	TRANSFER ADDRESS	MAP ADDRESS	TRANSFER PATH
DATAPROC.P.	AN	LOCAL	7938		
DATAPROC.P.	ANF	CORESIM	7946	5892	RTBUS
CORESIM.C.	JOBDONE	DATAPROC	7954	5898	RTBUS
CORESIM.C.	ACN	DATAPROC	7962	5904	RTBUS
CORESIM.C.	ADN	DATAPROC	7970	5910	RTBUS
CORESIM.C.	WDN	DATAPROC	7978	5916	RTBUS
CORESIM.C.	PRC	DATAPROC	7986	5922	I-BUS
CORESIM.P.	FFNA	LOCAL	7994		
CORESIM.P.	FLC	LOCAL	8002		
CORESIM.P.	WDNC	LOCAL	8010		
CORESIM.P.	PREPDONE	LOCAL	8018		
CORESIM.P.	DUCTDONE	LOCAL	8026		
DUCTSIM.C.	WDNB	CORESIM	8034	5948	RTBUS
DUCTSIM.C.	PRD	DATAPROC	8042	5954	I-BUS
DUCTSIM.P.	FFNA	LOCAL	8050		
DUCTSIM.P.	FLC	LOCAL	8058		
DUCTSIM.P.	PREPDONE	LOCAL	8066		

```
* * * * * * * * * * * * * *
*                         *
*  LOCAL DATA SEGMENT     *
*    ...DATAPROC (COMP)   *
*                         *
* * * * * * * * * * * * * *
```

DATAPROC COMPUTATIONAL PROCESSOR LOCAL VARIABLES

NAME	DTP	IC VALUE	HOLD VALUE	SCAL FAC	XREF	PVAL	LOCATION
* NEGATIVE	BT						
* OVERFLOW	BT						
* POSITIVE	BT						
* ZERO	BT						

DATAPROC COMPUTATIONAL PROCESSOR EXTERNAL VARIABLES

EXTERNAL VARIABLE NAME	LOCATION
CORESIM.C.ACN$0	7962
CORESIM.C.ADN$0	7970
DATAPROC.P.AN$0	7938
DATAPROC.P.ANF$0	7946
CORESIM.C.PRC$0	7986
DUCTSIM.C.PRD$0	8042
CORESIM.C.WDN$0	7978

DATAPROC COMPUTATIONAL PROCESSOR LOCAL CONSTANTS

NAME	TYP	VALUE	SCAL FAC	PRMTR	SIZE	LOCATION
MINAREA	S1	5.00000000E+001	B+11	NO	1	IM-DATA

DATAPROC COMPUTATIONAL PROCESSOR ARGUMENT GROUP VARIABLES

NAME	TYP	LOCATION	SIZE	USED	ITEM LIST
DATA	S1	10000	16	7	1) XV: DATAPROC.P.AN$0
					2) XV: DATAPROC.P.ANF$0
					3) XV: CORESIM.C.ACN$0
					4) XV: CORESIM.C.ADN$0
					5) XV: CORESIM.C.PRC$0
					6) XV: DUCTSIM.C.PRD$0
					7) XV: CORESIM.C.WDN$0

DATAPROC COMPUTATIONAL PROCESSOR TASKS
****

TASK NAME	ENABLE LATCH	TASK DISPATCHED	EXT. VARIABLE INVENTORY
GETDATA.	TRUE	YES	DATAPROC.P.AN
			DATAPROC.P.ANF
			CORESIM.C.ACN
			CORESIM.C.ADN
			CORESIM.C.PRC
			DUCTSIM.C.PRD
			CORESIM.C.WDN

DATAPROC COMPUTATIONAL PROCESSOR EXECUTIVES
****

EXEC NAME	PRIORITY LEVEL	SERVICE TASK(S)	SCRATCH-PAD-MEMORY EXEC / TASK / MACRO		
BADADC.	1	NONE	0	0	7
MAIN.	0	GETDATA.	0	0	7

```
   * * * * * * * * * * * * * * * * * * * *
   *                                       *
   *     EXECUTABLE STATEMENT SEGMENT      *
   *        ...DATAPROC (COMP)             *
   *                                       *
   * * * * * * * * * * * * * * * * * * * *
```

MAIN. EXECUTIVE: DEV1:0000.RTX.DATAPROC.SA

```
1   S$1       ENTER GETDATA
2   S$2       RETURN
```

MAX PATH EXECUTION TIME: 56 CYCLES (WITHOUT COMPUTE DELAYS)
+GETDATA.

BADADC. EXECUTIVE: DEV1:0000.RTX.DATAPROC.SA

```
1   S$3       IF .P.AN=.P.MINAREA
              THEN
2   S$4         ADVISE H.ADCMESS1
              ELSE
3   S$5         ADVISE H.ADCMESS2
4   S$6       RETURN
```

MAX PATH EXECUTION TIME: 346 CYCLES (WITHOUT COMPUTE DELAYS)

GETDATA. TASK: DEV1:0000.RTX.DATAPROC.SA

```
1   S$7       CALL SAMPLE[DATA]
2   S$8       ADVISE R.DATA
3   S$9       RETURN
```

MAX PATH EXECUTION TIME: 272 CYCLES (WITHOUT COMPUTE DELAYS)
+SAMPLE

TRANSFERRED VARIABLES

...NONE

OPERANDS SUBJECT TO COMPUTATIONAL DELAY (EXT MEM)

DATAPROC.P.AN
DATAPROC.P.ANF
CORESIM.C.ACN
CORESIM.C.ADN
CORESIM.C.PRC
DUCTSIM.C.PRD
CORESIM.C.WDN

```
* * * * * * * * * * * * * * *
*                           *
*  LOCAL DATA SEGMENT       *
*    ...DATAPROC (PREP)      *
*                           *
* * * * * * * * * * * * * * *
```

DATAPROC PRE-PROCESSOR LOCAL VARIABLES

	NAME	DTP	IC VALUE	HOLD VALUE	SCAL FAC	XREF	PVAL	LOCATION
	AN	S1	1.60000E+003	1.60000E+003	B+11	YES	1	10002
	ANF	S1	1.26317E+003	1.26317E+003	B+11	YES	1	10006
*	NEGATIVE	BT						
*	OVERFLOW	BT						
*	POSITIVE	BT						
*	ZERO	BT						

DATAPROC PRE-PROCESSOR EXTERNAL VARIABLES

EXTERNAL VARIABLE NAME	LOCATION
CORESIM.C.ACN$0	7962
CORESIM.C.ADN$0	7970
CORESIM.C.JOBDONE$0	7954

DATAPROC PRE-PROCESSOR LOCAL CONSTANTS

NAME	TYP	VALUE	SCAL FAC	PRMTR	SIZE	LOCATION
K1	I1	1.00000000E+000	NONE	YES	1	10008
K1P049	S1	1.04900000E+000	B+1	NO	1	IM-DATA
SC$1	S1	1.04900000E+000	B+11	NO	1	IM-DATA
K1P622M4	S1	1.62200000E-004	B-12	NO	1	IM-DATA
K2	S1	8.00000000E+002	B+11	NO	1	IM-DATA
MAXAREA	S1	1.60000000E+003	B+11	NO	1	IM-DATA
MINAREA	S1	5.00000000E+001	B+11	NO	1	IM-DATA

DATAPROC PRE-PROCESSOR ARGUMENT GROUP VARIABLES

NAME	TYP	LOCATION	SIZE	USED	ITEM LIST
ACNG	S1	10014	1	1	1) XV: CORESIM.C.ACN$0
ADCCHN	I1	10400	32	1	1) CN: DATAPROC.P.K1$1
ADCVAR	S1	10106	32	1	1) LV: DATAPROC.P.AN$1
ADNG	S1	10060	1	1	1) XV: CORESIM.C.ADN$0

DATAPROC PRE-PROCESSOR TASKS
****

TASK NAME	ENABLE LATCH	TASK DISPATCHED	EXT. VARIABLE INVENTORY
JOBDONE.	TRUE	YES	CORESIM.C.JOBDONE
WRTACN.	TRUE	NO	CORESIM.C.ACN
WRTADN.	TRUE	NO	CORESIM.C.ADN

DATAPROC PRE-PROCESSOR EXECUTIVES
****

EXEC NAME	PRIORITY LEVEL	SERVICE TASK(S)	SCRATCH-PAD-MEMORY EXEC	/ TASK	/ MACRO
GETAN.	0	JOBDONE. WRTACN. WRTADN.	0	0	3

```
   * * * * * * * * * * * * * * * * * * * * *
   *                                         *
   *     EXECUTABLE STATEMENT SEGMENT        *
   *        ...DATAPROC (PREP)               *
   *                                         *
   * * * * * * * * * * * * * * * * * * * * *
```

GETAN. EXECUTIVE: DEV1:0000.RTXPREP.DATAPROC.SA

```
 1   S$1       CALL READ[ADCVAR,ADCCHN]                          64
 2   S$2       AN=AN+K2                                         186
 3   S$3       IF AN<MINAREA                                     26
               THEN
 4   S$4         AN=MINAREA                                     176
 5   S$5         ACTIVATE BADADC                                182
               ELSE
 6   S$6         IF AN>MAXAREA                                   26
                 THEN
 7   S$7           AN=MAXAREA                                   176
 8   S$8           ACTIVATE BADADC                              174
 9   S$9       ANF=K1P049-K1P622M4*AN                           284
10   S$10      ENTER JOBDONE                                     40
11   S$11      DISPATCH WRTACN,WRTADN                           999
12   S$12      RETURN                                            16

    MAX PATH EXECUTION TIME: 1991 CYCLES (WITHOUT COMPUTE DELAYS)
    +READ
    +JOBDONE.
    +WRTACN.
    +WRTADN.
```

WRTACN. TASK: DEV1:0000.RTXPREP.DATAPROC.SA

```
 1   S$13      CALL DAC1[ACNG]                                   32
 2   S$14      RETURN                                            16

    MAX PATH EXECUTION TIME: 48 CYCLES (WITHOUT COMPUTE DELAYS)
    +DAC1
```

WRTADN. TASK: DEV1:0000.RTXPREP.DATAPROC.SA

```
 1   S$15      CALL DAC2[ADNG]                                   32
 2   S$16      RETURN                                            16

    MAX PATH EXECUTION TIME: 48 CYCLES (WITHOUT COMPUTE DELAYS)
    +DAC2
```

JOBDONE. TASK: DEV1:0000.RTXPREP.DATAPROC.SA

```
 1   TEST      IF #CORESIM.C.JOBDONE                            122
               THEN
 2   S$18        REDO TEST                                       18
               ELSE
 3   S$19        ADVISE H.DFMESS                                178
```

```
 4  S$20      RETURN                                                    16

    MAX PATH EXECUTION TIME: 316 CYCLES (WITHOUT COMPUTE DELAYS)

  TRANSFERRED VARIABLES
  ---------------------

    AN                  ANF

  OPERANDS SUBJECT TO COMPUTATIONAL DELAY (EXT MEM)
  -------------------------------------------------

    CORESIM.C.ACN
    CORESIM.C.ADN
    CORESIM.C.JOBDONE
```

```
* * * * * * * * * * * * * * *
*                           *
*  LOCAL DATA SEGMENT       *
*    ...CORESIM  (COMP)     *
*                           *
* * * * * * * * * * * * * * *
```

CORESIM COMPUTATIONAL PROCESSOR LOCAL VARIABLES

	NAME	DTP	IC VALUE	HOLD VALUE	SCAL FAC	XREF	PVAL	LOCATION
	ACN	S1	0.00000E+000	0.00000E+000	B+10	YES	1	10014
	ACNA	S1	0.00000E+000	0.00000E+000	B+11	NO	1	10026
	ADN	S1	0.00000E+000	0.00000E+000	B+10	YES	1	10018
	FFN	S1	0.00000E+000	0.00000E+000	B+2	NO	1	10006
	FFNB	S1	1.00000E+000	1.00000E+000	B+1	NO	1	10010
	JOBDONE	B	FALSE	TRUE	NONE	YES	1	10002
	PRC	S1	1.00000E+000	1.00000E+000	B+1	YES	1	10030
	WDN	S1	0.00000E+000	0.00000E+000	B+9	YES	1	10022
*	NEGATIVE	BT						
	OVERFLOW	BT						
*	POSITIVE	BT						
*	ZERO	BT						

CORESIM COMPUTATIONAL PROCESSOR EXTERNAL VARIABLES

EXTERNAL VARIABLE NAME	LOCATION
DATAPROC.P.ANF$0	7946
CORESIM.P.DUCTDONE$0	8026
CORESIM.P.FFNA$0	7994
CORESIM.P.FLC$0	8002
CORESIM.P.PREPDONE$0	8018
CORESIM.P.WDNC$0	8010

CORESIM COMPUTATIONAL PROCESSOR LOCAL CONSTANTS

NAME	TYP	VALUE	SCAL FAC	PRMTR	SIZE	LOCATION
FALSE	B	GLOBAL VALUE	NONE	NO	1	10032
K1P	S1	1.00000000E+000	B+1	NO	1	IM-DATA
SC$1	S1	1.00000000E+000	B+2	NO	1	IM-DATA
KCN	S1	5.12400000E-001	B+0	NO	1	IM-DATA
KP2588	S1	2.58800000E-001	B-1	NO	1	IM-DATA
SC$2	S1	2.58800000E-001	B+2	NO	1	IM-DATA
KP53	S1	5.30000000E-001	B+0	NO	1	IM-DATA
P0	S1	GLOBAL VALUE	B+6	YES	1	10036
PCN	S1	GLOBAL VALUE	B+7	YES	1	10034
PRNVALS	I1	GLOBAL VALUE	NONE	NO	1	10080

CORESIM COMPUTATIONAL PROCESSOR LOCAL CONSTANTS (CONTINUED)
****

NAME	TYP	VALUE	SCAL FAC	PRMTR	SIZE	LOCATION
PRXVALS	S1	GLOBAL VALUE	B+1	NO	21	10082
TCN	S1	GLOBAL VALUE	B+13	YES	1	10124
TRUE	B	GLOBAL VALUE	NONE	NO	1	10128
WCN	S1	GLOBAL VALUE	B+8	YES	1	10126
XTO2857	S1	GLOBAL VALUE	B+1	NO	21	10038
ZERO	S1	0.00000000E+000	B+0	NO	1	IM-DATA
SC$3	S1	0.00000000E+000	B+10	NO	1	IM-DATA

CORESIM COMPUTATIONAL PROCESSOR EXECUTIVES
****

EXEC NAME	PRIORITY LEVEL	SERVICE TASK(S)	SCRATCH-PAD-MEMORY EXEC / TASK / MACRO		
MAINSIM.	0	NONE	0	0	2

```
    * * * * * * * * * * * * * * * * * * * * *
    *                                       *
    *     EXECUTABLE STATEMENT SEGMENT      *
    *        ...CORESIM  (COMP)             *
    *                                       *
    * * * * * * * * * * * * * * * * * * * * *

  MAINSIM. EXECUTIVE: DEV1:0000.DSC.CORESIM.SA
```

1	S$1	JOBDONE=FALSE	270
2	S$2	PRC=P0/PCN	376
3	S$3	IF OVERFLOW	10
		THEN	
4	S$4	ADVISE H.COREMES1	180
5	S$5	FFNB=SQRT(K1P-FUN1[PRXVALS,PRNVALS,XT02857,PRC])	1156
6	S$6	ACNA=KCN*WCN*SQRT(TCN)	780
7	S$7	IF $.P.PREPDONE	122
		THEN	
8	S$8	ADVISE M.COREMES3	180
9	S$9	IF PRC$>KP53	38
		THEN	
10	S$10	FFN=KP2588	44
		ELSE	
11	S$11	FFN=FFNB*.P.FFNA	222
12	S$12	IF OVERFLOW	10
		THEN	
13	S$13	ADVISE H.COREMES2	180
14	S$14	IF FFN>ZERO	38
		THEN	
15	S$15	ACN=ACNA/PCN*FFN*.P.FLC	736
		ELSE	
16	S$16	ACN=ZERO	258
17	S$17	IF $.P.DUCTDONE	122
		THEN	
18	S$18	ADVISE M.COREMES4	180
19	S$19	IF ACN=ZERO	38
		THEN	
20	S$20	ADN=ZERO	268
		ELSE	
21	S$21	ADN=DATAPROC.P.ANF-ACN	410
22	S$22	WDN=ADN*.P.WDNC	458
23	S$23	JOBDONE=TRUE	270
24	S$24	RETURN	16

MAX PATH EXECUTION TIME: 5792 CYCLES (WITHOUT COMPUTE DELAYS)

TRANSFERRED VARIABLES

JOBDONE ACN ADN WDN
PRC

OPERANDS SUBJECT TO COMPUTATIONAL DELAY (EXT MEM)

CORESIM.P.PREPDONE

```
CORESIM.P.FFNA
CORESIM.P.FLC
CORESIM.P.DUCTDONE
DATAPROC.P.ANF
CORESIM.P.WDNC
```

```
* * * * * * * * * * * * * * *
*                           *
*  LOCAL DATA SEGMENT       *
*    ...CORESIM  (PREP)      *
*                           *
* * * * * * * * * * * * * * *
```

CORESIM PRE-PROCESSOR LOCAL VARIABLES

	NAME	DTP	IC VALUE	HOLD VALUE	SCAL FAC	XREF	PVAL	LOCATION
	DUCTDONE	B	FALSE	FALSE	NONE	YES	1	10022
	FFNA	S1	0.00000E+000	0.00000E+000	B+1	YES	1	10006
	FLC	S1	8.25000E-001	8.25000E-001	B+1	YES	1	10010
	PRC	S1	1.00000E+000	1.00000E+000	B+1	NO	1	10002
	PREPDONE	B	FALSE	TRUE	NONE	YES	1	10018
	WDNC	S1	0.00000E+000	0.00000E+000	B+2	YES	1	10014
*	NEGATIVE	BT						
*	OVERFLOW	BT						
*	POSITIVE	BT						
*	ZERO	BT						

CORESIM PRE-PROCESSOR EXTERNAL VARIABLES

EXTERNAL VARIABLE NAME	LOCATION
DUCTSIM.C.WDNB$0	8034

CORESIM PRE-PROCESSOR LOCAL CONSTANTS

NAME	TYP	VALUE	SCAL FAC	PRMTR	SIZE	LOCATION
FALSE	B	GLOBAL VALUE	NONE	NO	1	10024
K1P	S1	1.00000000E+000	B+1	NO	1	IM-DATA
K1P3625	S1	1.36250000E+000	B+1	NO	1	IM-DATA
KP7158	S1	7.15800000E-001	B+0	NO	1	IM-DATA
KP825	S1	8.25000000E-001	B+0	NO	1	IM-DATA
SC$1	S1	8.25000000E-001	B+1	NO	1	IM-DATA
P0	S1	GLOBAL VALUE	B+6	YES	1	10028
PCN	S1	GLOBAL VALUE	B+7	YES	1	10026
PDN	S1	GLOBAL VALUE	B+7	YES	1	10118
PRNVALS	I1	GLOBAL VALUE	NONE	NO	1	10072
PRXVALS	S1	GLOBAL VALUE	B+1	NO	21	10074
TRUE	B	GLOBAL VALUE	NONE	NO	1	10116
XT07143	S1	GLOBAL VALUE	B+1	NO	21	10030

CORESIM PRE-PROCESSOR EXECUTIVES
****

EXEC NAME	PRIORITY LEVEL	SERVICE TASK(S)	SCRATCH-PAD-MEMORY EXEC / TASK / MACRO
PRFCTNS.	0	NONE	0 0 2

```
* * * * * * * * * * * * * * * * * * * *
*                                       *
*     EXECUTABLE STATEMENT SEGMENT      *
*        ...CORESIM  (PREP)             *
*                                       *
* * * * * * * * * * * * * * * * * * * *
```

PRFCTNS, EXECUTIVE: DEV1:0000.DSCPREP.CORESIM.SA

```
 1   S$1       PREPDONE=FALSE                                          186
 2   S$2       DUCTDONE=FALSE                                          186
 3   S$3       PRC=P0/PCN                                              236
 4   S$4       FLC=K1P3625-KP7158*PRC                                  274
 5   S$5       IF FLC>K1P                                               26
               THEN
 6   S$6         FLC=K1P                                               184
               ELSE
 7   S$7         IF FLC<KP825                                           26
                 THEN
 8   S$8           FLC=KP825                                           176
 9   S$9       FFNA=FUN1[PRXVALS,PRNVALS,XT07143,PRC]                  750
10   S$10      PREPDONE=TRUE                                           186
11   S$11      WDNC=PDN*DUCTSIM.C.WDNB                                 374
12   S$12      DUCTDONE=TRUE                                           186
13   S$13      RETURN                                                   16
```

MAX PATH EXECUTION TIME: 2622 CYCLES (WITHOUT COMPUTE DELAYS)

TRANSFERRED VARIABLES

FFNA FLC WDNC PREPDONE
DUCTDONE

OPERANDS SUBJECT TO COMPUTATIONAL DELAY (EXT MEM)

DUCTSIM.C.WDNB

```
* * * * * * * * * * * * * *
*                         *
*  LOCAL DATA SEGMENT     *
*    ...DUCTSIM  (COMP)   *
*                         *
* * * * * * * * * * * * * *
```

DUCTSIM COMPUTATIONAL PROCESSOR LOCAL VARIABLES

	NAME	DTP	IC VALUE	HOLD VALUE	SCAL FAC	XREF	PVAL	LOCATION
	FFN	S1	0.00000E+000	0.00000E+000	B+2	NO	1	10002
	FFNB	S1	1.00000E+000	1.00000E+000	B+1	NO	1	10010
	FRD	S1	1.00000E+000	1.00000E+000	B+1	YES	1	10018
	WDNA	S1	1.51665E+002	1.51665E+002	B+8	NO	1	10014
	WDNB	S1	0.00000E+000	0.00000E+000	B+5	YES	1	10006
*	NEGATIVE	BT						
	OVERFLOW	BT						
*	POSITIVE	BT						
*	ZERO	BT						

DUCTSIM COMPUTATIONAL PROCESSOR EXTERNAL VARIABLES

EXTERNAL VARIABLE NAME	LOCATION
DUCTSIM.P.FFNA$0	8050
DUCTSIM.P.FLC$0	8058
DUCTSIM.P.PREPDONE$0	8066

DUCTSIM COMPUTATIONAL PROCESSOR LOCAL CONSTANTS

NAME	TYP	VALUE	SCAL FAC	PRMTR	SIZE	LOCATION
K1P	S1	1.00000000E+000	B+1	NO	1	IM-DATA
SC$1	S1	1.00000000E+000	B+2	NO	1	IM-DATA
KDN	S1	5.06550000E-001	B+0	NO	1	IM-DATA
KP2588	S1	2.58800000E-001	B+0	NO	1	IM-DATA
SC$2	S1	2.58800000E-001	B+2	NO	1	IM-DATA
KP53	S1	5.30000000E-001	B+0	NO	1	IM-DATA
P0	S1	GLOBAL VALUE	B+6	YES	1	10022
PDN	S1	GLOBAL VALUE	B+7	YES	1	10020
PRNVALS	I1	GLOBAL VALUE	NONE	NO	1	10066
PRXVALS	S1	GLOBAL VALUE	B+1	NO	21	10068
TDN	S1	GLOBAL VALUE	B+13	YES	1	10110
XT02857	S1	GLOBAL VALUE	B+1	NO	21	10024

DUCTSIM COMPUTATIONAL PROCESSOR EXECUTIVES
****

EXEC NAME	PRIORITY LEVEL	SERVICE TASK(S)	SCRATCH-PAD-MEMORY EXEC / TASK / MACRO		
COFROCE.	0	NONE	0	0	2

```
    * * * * * * * * * * * * * * * * * * *
    *                                     *
    *     EXECUTABLE STATEMENT SEGMENT    *
    *        ...DUCTSIM  (COMP)           *
    *                                     *
    * * * * * * * * * * * * * * * * * * *
```

COPROCE. EXECUTIVE: DEV1:0000.DSC.DUCTSIM.SA

```
 1   S$1        PRD=P0/PDN                                          376
 2   S$2        IF OVERFLOW                                          10
                THEN
 3   S$3          ADVISE H.COREMES2                                 180
 4   S$4        FFNB=SQRT(K1P-FUN1(PRXVALS,PRNVALS,XT02857,PRD))   1156
 5   S$5        WDNA=KDN*SQRT(TDN)                                  686
 6   S$6        IF #.P.PREPDONE                                     122
                THEN
 7   S$7          ADVISE M.DUCTMESS                                 180
 8   S$8        IF PRD#>KP53                                         38
                THEN
 9   S$9          FFN=KP2588                                         44
                ELSE
10   S$10         FFN=FFNB*.P.FFNA                                  222
11   S$11       WDNB=FFN*.P.FLC/WDNA                                432
12   S$12       RETURN                                               16
```

MAX PATH EXECUTION TIME: 3618 CYCLES (WITHOUT COMPUTE DELAYS)

TRANSFERRED VARIABLES

```
WDNB                PRD
```

OPERANDS SUBJECT TO COMPUTATIONAL DELAY (EXT MEM)

```
DUCTSIM.P.PREPDONE
DUCTSIM.P.FFNA
DUCTSIM.P.FLC
```

```
* * * * * * * * * * * * * * *
*                           *
*  LOCAL DATA SEGMENT       *
*    ...DUCTSIM  (PREP)      *
*                           *
* * * * * * * * * * * * * * *
```

DUCTSIM PRE-PROCESSOR LOCAL VARIABLES

	NAME	DTP	IC VALUE	HOLD VALUE	SCAL FAC	XREF	PVAL	LOCATION
	FFNA	S1	0.00000E+000	0.00000E+000	B+1	YES	1	10006
	FLC	S1	8.25000E-001	8.25000E-001	B+1	YES	1	10010
	PRD	S1	1.00000E+000	1.00000E+000	B+1	NO	1	10002
	PREPDONE	B	FALSE	TRUE	NONE	YES	1	10014
*	NEGATIVE	BT						
*	OVERFLOW	BT						
*	POSITIVE	BT						
*	ZERO	BT						

DUCTSIM PRE-PROCESSOR LOCAL CONSTANTS

NAME	TYP	VALUE	SCAL FAC	PRMTR	SIZE	LOCATION
FALSE	B	GLOBAL VALUE	NONE	NO	1	10016
K1P	S1	1.00000000E+000	B+1	NO	1	IM-DATA
K1P575	S1	1.57500000E+000	B+1	NO	1	IM-DATA
KP825	S1	8.25000000E-001	B+0	NO	1	IM-DATA
SC$1	S1	8.25000000E-001	B+1	NO	1	IM-DATA
P0	S1	GLOBAL VALUE	B+6	YES	1	10020
PDN	S1	GLOBAL VALUE	B+7	YES	1	10018
PRNVALS	I1	GLOBAL VALUE	NONE	NO	1	10064
PRXVALS	S1	GLOBAL VALUE	B+1	NO	21	10066
TRUE	B	GLOBAL VALUE	NONE	NO	1	10108
XT07143	S1	GLOBAL VALUE	B+1	NO	21	10022

DUCTSIM PRE-PROCESSOR EXECUTIVES

EXEC NAME	PRIORITY LEVEL	SERVICE TASK(S)	SCRATCH-PAD-MEMORY EXEC / TASK / MACRO
PRFCTNS.	0	NONE	0 0 2

```
* * * * * * * * * * * * * * * * * * * *
*                                       *
*     EXECUTABLE STATEMENT SEGMENT      *
*        ...DUCTSIM  (PREP)             *
*                                       *
* * * * * * * * * * * * * * * * * * * *
```

PRFCTNS. EXECUTIVE: DEV1:0000.DSCPREP.DUCTSIM.SA

```
 1   S$1       PREPDONE=FALSE                                    186
 2   S$2       PRD=P0/PDN                                        236
 3   S$3       FLC=K1P575-PRD                                    174
 4   S$4       IF FLC>K1P                                         26
               THEN
 5   S$5         FLC=K1P                                         184
               ELSE
 6   S$6         IF FLC<KP825                                     26
                 THEN
 7   S$7           FLC=KP825                                     176
 8   S$8       FFNA=FUN1[PRXVALS,PRNVALS,XT07143,PRD]            750
 9   S$9       PREPDONE=TRUE                                     186
10   S$10      RETURN                                             16
```

MAX PATH EXECUTION TIME: 1776 CYCLES (WITHOUT COMPUTE DELAYS)

TRANSFERRED VARIABLES

FFNA FLC PREPDONE

OPERANDS SUBJECT TO COMPUTATIONAL DELAY (EXT MEM)

...NONE

Appendix B—Error and Warning Messages

SPECIFIC INFORMATION PERTAINING TO THE PROGRAM IS ENCLOSED WITHIN ANGLE BRACKETS (<...>) IN THE MESSAGES.

RTMPL ERROR MESSAGES

1. STATEMENT SIZE (3200):<NUMBER OF CHARACTERS ENTERED>
2. EXCESS STATEMENT DELIMITERS (120):<NUMBER OF DELIMITERS ENTERED>
3. SIMULATION DESCRIPTION SIZE (64):<ENTRY>
4. ORIGINATOR ID SIZE (32):<ENTRY>
5. NUMBER OF SIMULATOR CHANNELS (1..<MAX NUMBER>):<ENTRY>
6. CONFIGURATION ENTRY (SINGLE OR DUAL):<ENTRY>
7. PROGRAM CATALOG ID (RTX,IOP,DSC):<ENTRY>
8. GLOBAL CATALOG ID (GLOBAL):<ENTRY>
9. TARGET CATALOG ID (M68000):<ENTRY>
10. VOLUME SPECIFICATION:<ENTRY>
11. RESOURCE NAME STRING:<ENTRY>
12. USER NUMBER (0..9999):<ENTRY>
13. INCOMPLETE FILE
14. OPTION SPECIFICATION:<ENTRY>
15. ONLY 1 RTX PROCESSOR ALLOWED:<NUMBER SPECIFIED>
16. ONLY 1 IOP PROCESSOR ALLOWED:<NUMBER SPECIFIED>
17. MULT-DEFINED CHANNEL ID:<CHANNEL ID>
18. ILLEGAL CHAR IN INTEGER:<INTEGER AS SPECIFIED>
19. NON-ALPHABETIC NAME:<NAME AS SPECIFIED>
20. NAME EXCEEDS 8 CHARACTERS:<NAME AS SPECIFIED>
21. INTEGER ENTRY EXPECTED BETWEEN <CHARACTER> AND <CHARACTER>
22. NON-ALPHANUMERIC CHAR(S) IN NAME:<NAME AS SPECIFIED>
23. NAME ENTRY EXPECTED BETWEEN <CHARACTER> AND <CHARACTER>
24. NULL GLOBAL DEFINITION:??
25. MULT-DEFINED OF-SYS TASK:<TASK ID>
26. TOO MANY DELIMITERS:<ENTRY>
27. UNDEFINED SYSTEM TASK:<TASK ID>
28. NAME=MESSAGE FORMAT REQUIRED:<ENTRY>
29. MULT-DEFINED MESSAGE ID:<MESSAGE ID>
30. MESSAGE EXCEEDS 64 CHARACTERS:<ENTRY>
31. NAME=SYS-TASK ID REQUIRED:<ENTRY>
32. EXTRANEOUS EOR:<ENTRY>
33. TOO MANY COLONS:<ENTRY>
34. UNDEFINED RECORD NAME:<ENTRY>
35. EOR MISSING:<ENTRY>
36. MULT-DEFINED CONSTANT ID:<CONSTANT ID>
37. = EXPECTED:<ENTRY>
38. CONSTANT DEFINITION FORMAT:<ENTRY>
39. SCALE FACTOR REQUIRED!
40. ILLEGAL VALUE DEFAULT!
41. ILLEGAL DTF DEFAULT!
42. TOO MANY VALUES ENTERED: SIZE =<NUMBER OF VALUES ENTERED>
43. VALUE EXPECTED FOR ITEM NUMBER <ITEM NUMBER>
44. CONSTANT CANNOT BE BOOLEAN!';
45. <COUNT> @S IN VALUE SPECIFICATION!';
46. MISMATCH OF CONSTANT SIZE/VALUES:<SIZE SPECIFIED>/<NUMBER OF VALUES>

```
 47. <COUNT> ILLEGAL CHAR(S) IN REAL NUM:<REAL NUMBER AS SPECIFIED>
 48. TOO MANY DECIMAL PTS (<COUNT>):<ENTRY>
 49. TOO MANY EXPONENTS (<COUNT>):<ENTRY>
 50. REAL NUMBER FORMAT:<REAL NUMBER AS SPECIFIED>
 51. NO DTP ENTERED:<ENTRY>
 52. ILLEGAL DTP:<DTP AS SPECIFIED>
 53. ILLEGAL DATA TYPE(I,S,F,B):<DATA TYPE AS SPECIFIED>
 54. ILLEGAL PRECISION:<PRECISION SPECIFIED>
 55. ARG GROUP FORMAT:<ARGGROUP AS SPECIFIED>
 56. MULT-DEFINED ARG GROUP:<ARGGROUP ID>
 57. ARG GROUP DEFINITION EXPECTED!
 58. ARG GROUP SIZE EXCEEDED:SIZE=<SIZE> :CNT=<NUMBER OF ITEMS SPECIFIED>
 59. EXTERNAL ARG GROUP REFERENCE:<CHANNEL ID>,<ARGGROUP ID>
 60. ARG GROUP ITEM IN ARG GROUP:<ITEM ID>
 61. EXTERNAL PREP VRBL REFERENCE:<VARIABLE ID>
 62. ARGGROUP/ARGUMENT DTP CONFLICT:<VARIABLE ID>
 63. EXTERNAL TARGET STATE REFERENCE:<TARGET STATE ID>
 64. NON-PARAMETRIC CONSTANT REFERENCE:<CONSTANT ID>
 65. TOO MANY PERIODS:<ARGUMENT AS SPECIFIED>
 66. TOO MANY DOLLARS:<ARGUMENT AS SPECIFIED>
 67. UNDEFINED ARGUMENT SOURCE:<ARGUMENT AS SPECIFIED>
 68. ILLEGAL PROCESSOR TYPE:<ARGUMENT AS SPECIFIED>
 69. UNDEFINED ARGUMENT:<CHANNEL ID>,<PROCESSOR TYPE>,<NAME>
 70. ARG GROUP ITEM SPECIFICATION:<ARGGROUP ID>
 80. ITEM NUMBER > SPECIFIED SIZE:<CONSTANT OR VARIABLE ID>
 81. CONFIGURATION CONFLICT (SINGLE CHANNEL)
 82. UNDEFINED DEFAULT SOURCE PROGRAM
 83. GLOBAL CNST SOURCE SPECIFIED:<ENTRY>
 84. CONSTANT FORMATION ID CONFLICT WITH <CONSTANT ID>
 85. ID CONFLICT WITH GLOBAL CONSTANT:<CONSTANT ID>
 86. NULL RECORD IN PROGRAM
 87. FOREGROUND EXEC PRIORITY DUPLICATION WITH <EXEC ID>
 88. MULT-DEFINED TASK:<TASK ID>
 89. NAME[PRIORITY] FORMAT REQUIRED:<ENTRY>
 90. MULT-DEFINED EXECUTIVE:<EXEC ID>
 91.  / EXPECTED:<ENTRY>
 92. NO EXECUTIVES IN <CHANNEL ID>,<PROCESSOR TYPE>
 93.  , EXPECTED:<ENTRY>
 94.  [ EXPECTED:<ENTRY>
 96.  ] EXPECTED:<ENTRY>
 97. VARIABLE DEFINITION FORMAT:<ENTRY>
 98. VARIABLE ID REQUIRED:<ENTRY>
 99. MULT-DEFINED VARIABLE:<VARIABLE ID>
100. BOOLEAN VALUE REQUIRED:<ENTRY>
101. VRBL DEFINED IN COMP PROCESSOR:<CHANNEL ID>
102. VARIABLE/CONSTANT ID CONFLICT:<CONSTANT ID>
103. VARIABLE/GLBCNST ID CONFLICT:<CONSTANT ID>
104. TASK STATEMENTS REQUIRED:<TASK ID>
105. EXEC STATEMENTS REQUIRED:<EXEC ID>
106. UNRECONCILED IF STATEMENT(S):<LIST OF IF-STATEMENT LABELS>
107. UNDEFINED STATEMENT LABEL(S):<LIST OF STATEMENT LABELS>
108. RETURN EXPECTED AT END OF EXEC/TASK!
109. NO IF STATEMENT TO END (! ILLEGAL)
110. STATEMENT FORMAT! ONLY <COUNT> DELIMITERS FOUND
111. UNDEFINED RESULT VARIABLE:<VARIABLE ID>
112. OMITTED LABEL END DELIMITER:<LABEL AS SPECIFIED>
113. UNDEFINED COMMAND/CONDITIONAL:<COMMAND AS SPECIFIED>
```

```
114. DUPLICATE LABEL (TASK/EXEC):<LABEL>
115. COMMAND DISALLOWED IN TASK:<COMMAND>
116. UNDEFINED ARGUMENT TASK:<TASK ID>
117. THEN OR ELSE CLAUSE EXPECTED!
118. EXTRANEOUS INFO IN STATEMENT:<EXTRANEOUS INFORMATION>
119. THEN NOT EXPECTED HERE:<ENTRY>
120. ELSE NOT EXPECTED HERE:<ENTRY>
121. UNDEFINED DESTINATION (REDO/EXIT)
122. COMMAND ILLEGAL UNLESS IN IF STATEMENT:<COMMAND ID>
123. RESULT IS TARGET STATE VARIABLE:<VARIABLE ID>
124. FORMAT OF CALL COMMAND:<ENTRY>
125. UNDEFINED TARGET PROCEDURE:<NAME>
126. UNDEFINED ADVISORY TYPE (USE ,M,,H,,R,,W,,E)
127. NUMBER OF TASKS EXCEEDS 99999
128. UNDEFINED ARGUMENT GROUP:<NAME>
129. PROCEDURE/ARG GROUP DTP CONFLICT:<ARGGROUP ID>
130. UNDEFINED ADVISORY ARGUMENT:<ENTRY>
131. BE MORE ORIGINAL WITH ERRORS!
132. DISPATCH COMMAND ILLEGAL ON COMPUTATIONAL PROCESSOR!
133. YOU ARE GENERATING AN INFINITE LOOP!
134. UNDEFINED COMMAND:<COMMAND ID>
135. MVF ARGUMENT COUNT MISMATCH (REQ=<COUNT>) <> (<NUMBER OF ARGUMENTS>)
136. ARG-SUPPLIED<>ARG-REQUIRED: (<COUNT>) <> (<COUNT>)
137. NULL EXPRESSION ENCOUNTERED
138. ILLEGAL RESCALING OF CONSTANT REQUIRED:<SCALING INFORMATION>
139. REQUIRED MACRO NOT TARGET SUPPORTED:<MACRO ID>
140. MULTIPLE ] ENCOUNTERED IN MVF
141.  [ MISSING IN MVF
142. MVF NAME MISSING
143. EXPRESSION PROCESSING ABORTED
144.  ( NOT EXPECTED ,LOCATION:<STATEMENT CHARACTER INDEX>
145. OPERATOR MISSING ,LOCATION:<STATEMENT CHARACTER INDEX>
146.  [ NOT EXPECTED ,LOCATION:<STATEMENT CHARACTER INDEX>
147. MISSING OPERAND AFTER <STATEMENT CHARACTER> LOCATION <INDEX>
148. UNDEFINED MVF NAME:<NAME>
149. MVF/RESULT DATA TYPE CONFLICT:<FUNCTION ID> #<DTP>
150. UNDEFINED UF NAME:<NAME>
151. UF/RESULT DATA TYPE CONFLICT:<FUNCTION ID> #<DTP>
152. OPERAND MISSING, LOCATION:<STATEMENT CHARACTER INDEX>
153. EXTRANEOUS ) ENCOUNTERED!
154. OPERAND/RESULT DATA TYPE CONFLICT:<OPERAND ID>
155. OPERATION NOT TARGET SUPPORTED:<OPERATION ID>
156. OPERATION/OPERAND DATA TYPE CONFLICT:<OPERATION ID> #<DTP>
157. RTMFL SCRATCH PAD MEMORY OVERFLOW
158. RTMFL SCALE FACTOR SHIFT OVERFLOW
159. RTMFL EXTERNAL VARIABLE MEMORY OVERFLOW
160. NUMBER TOO LARGE FOR SCALE FACTOR:<NUMBER SPECIFIED>
161. INTEGER TYPE HAS FRACTIONAL VALUE:<INTEGER SPECIFIED>
162. INTEGER TOO LARGE FOR PRECISION:<INTEGER SPECIFIED>
163. DECODED INTEGER EXCEEDS 999999999:<INTEGER SPECIFIED>
164. LOCAL VARIABLE SPECIFICATION REQUIRED INSTEAD OF <NAME>
165. PAST VALUE CANNOT BE SENT:<ENTRY>
166. EXTERNAL VARIABLE SPECIFICATION REQUIRED INSTEAD OF <NAME>
167. SOURCE/DESTINATION DTP CONFLICT:<NAME>/<NAME>
168. PROCESSOR TYPE CONFLICT:<CHANNEL ID>.<PROC TYPE>/<CHANNEL ID>.<PROC TYP>
169. UNDEFINED LOCAL VARIABLE:<NAME>
170. CONFIGURATION (SINGLE) DOES NOT SUPPORT INTRA-PROCESSOR INTERRUPTS!
```

171. UNDEFINED FOREGROUND EXECUTIVE (<NAME>) IN <CHANNEL ID>
172. BACKGROUND EXECUTIVE CANNOT BE ACTIVATED:<EXEC ID>

RTMPL WARNING MESSAGES

1. END OF IF-STATEMENT EXPECTED
2. UNNECESSARY COMA ENCOUNTERED
3. MULTIPLE ASSIGNMENTS OF <VARIABLE ID>
4. STRUCTURE VIOLATION! REDO BEING USED AS GOTO
5. VARIABLE RE-SCALING ENCOUNTERED:<VARIABLE ID> RSF=<N> NSF=<N>
6. CONSTANT RE-SCALING ENCOUNTERED:<CONSTANT ID> RSF=<N> NSF=<N>
7. VARIABLE PRECISION ADJUSTMENT:<VARIABLE ID> RPRC=<N> NPRC=<N>
8. CONSTANT PRECISION ADJUSTMENT:<CONSTANT ID> RPRC=<N> NPRC=<N>
9. DECIMAL POINT ENCOUNTERED IN INTEGER:<ENTRY>
10. DECIMAL POINT EXPECTED IN SCALED-FRACTION:<ENTRY>
11. DECIMAL POINT EXPECTED IN FLOATING POINT NUMBER:<ENTRY>

Appendix C—Assembler Source Files for Dual-Nozzle Simulation

```
PAGE  1    LIST VERSION  042682 3  11/30/84  13:02:25   DEV1:0.OBJCOMP.DATAPROC
  1  * RTMPL: DUALNOZZ
  2  *        DALE J. ARPASI
  3  * PROGRAM: DATAPROC - COMP
  4  * FUNCTION: RTX
  5  * GENERATED: 11/27/84  10:28:52
  6  *
  7  *
  8  * REQUIRED TARGET MACRO FILE
  9  INITIAL$ MACRO
 10  .CHB     EQU     $81C
 11  .AVT     EQU     $844
 12  .AVX     EQU     $845
 13  .AVS     EQU     $846
 14  .AVB     EQU     $847
 15  .AWC     EQU     $84C
 16  .NOFE    EQU     ($918)*2
 17  .XA1$P   EQU     ($9B2)*2
 18  .XA1$C   EQU     ($9B6)*2
 19  .XA2$P   EQU     ($9BA)*2
 20  .XV2$P   EQU     ($9BC)*2
 21  .XA2$C   EQU     ($9C2)*2
 22  .XV2$C   EQU     ($9C4)*2
 23  .FE$P    EQU     ($9C8)*2
 24  .FE$C    EQU     ($9CC)*2
 25  .AK$P    EQU     ($9D4)*2
 26  .AK$C    EQU     ($9D6)*2
 27  .PB$P    EQU     ($9D8)*2
 28  .PB$C    EQU     ($9D9)*2
 29  .PI4B    EQU     ($9DA)*2
 30  .EAD     EQU     $1400
 31  .XVM     EQU     $1F00
 32  .XVC     EQU     $1F01
 33           ENDM
 34  ORG$     MACRO
 35           ORG     \1
 36           ENDM
 37  DC$      MACRO
 38           DC      \1
 39           ENDM
 40  DS$      MACRO
 41           DS      \1
 42           ENDM
 43  DL$      MACRO
 44           DC.L    \1
 45           ENDM
 46  DN$      MACRO
 47           DC.L    '\1'
 48           ENDM
 49  DATASEG$ MACRO
 50           ENDM
 51  DATAEND$ MACRO
 52           ENDM
 53  TSTXVA$  MACRO
 54           CMP.B   (\1-1),D7
 55           BEQ.S   TA3\@+4
 56           CMP.B   ((\1-.XVC)*2)(A4),D7
 57           BEQ.S   TA2\@
```

```
 58                CLR       D5
 59   TA1\@        ADDQ      #1,D5
 60                TRAPV
 61                CMPA      #0,A4
 62                CMP.B     ((\1-.XVC)*2)(A4),D7
 63                NOP
 64                BNE.S     TA1\@
 65                CMP       (\1+$1D100),D5
 66                BLE.S     TA2\@
 67                MOVE      D5,\1+$1D100
 68   TA2\@        MOVEP.L   ((\1-.XVM)*2)(A4),D5
 69                MOVE.L    D5,\1
 70                MOVE.B    #0,((\1-.XVC)*2)(A4)
 71   TA3\@        MOVE.B    D7,(\1-1)
 72                ENDM
 73   ADVISE$H     MACRO
 74                MOVE.B    #\1,D5
 75                TRAP      #9
 76                ENDM
 77   ADVISE$R     MACRO
 78                MOVE.B    #1,.PI4B(A4)
 79                CLR.L     D5
 80   AR1\@        TST.B     .AVB
 81                NOP
 82                BEQ.S     AR2\@
 83                ADDQ.L    #1,D5
 84                TRAPV
 85                CMPA      D7,A4
 86                BRA.S     AR1\@
 87   AR2\@        CMP.L     .AWC,D5
 88                BLE.S     AR3\@
 89                MOVE.L    D5,.AWC
 90   AR3\@        MOVE.B    #3,.AVB
 91                MOVE.B    #3,.AVT
 92                MOVE.B    #\1,.AVX
 93                MOVE.B    #1,.AVS
 94                ORI       #6,$1FFF0
 95                ENDM
 96   RETURN$I     MACRO
 97                MOVEA.L   \1-4,A0
 98                JMP       (A0)
 99                ENDM
100   RETURN$E     MACRO
101                RTS
102                ENDM
103   RETURN$T     MACRO
104                RTS
105                ENDM
106   BACKEXEC     MACRO
107                LEA       \1-\2,A3
108                ENDM
109   FOREEXEC     MACRO
110                LEA       \1-\2,A3
111                ENDM
112   ENTER$       MACRO
113                TST       \1-4
114                BEQ.S     *+4
```

```
115                JSR       \1+4
116                ENDM
117   EXIT$        MACRO
118                JMP       \1
119                ENDM
120   CALL$        MACRO
121                IFNC      '',\2'
122                LEA       \2,A0
123                MOVE.L    A0,-(A7)
124                ENDC
125                IFNC      '',\3'
126                LEA       \3,A0
127                MOVE.L    A0,-(A7)
128                ENDC
129                IFNC      '',\4'
130                LEA       \4,A0
131                MOVE.L    A0,-(A7)
132                ENDC
133                IFNC      '',\5'
134                LEA       \5,A0
135                MOVE.L    A0,-(A7)
136                ENDC
137                JSR       \1
138                ENDM
139   MLD$S1       MACRO
140                MOVE      \1,-(A7)
141                ENDM
142   JNEQ$S1      MACRO
143                CMP       (A7)+,\2
144                BNE       \1
145                ENDM
146   LDI$S1       MACRO
147                MOVE      #\2,\1
148                ENDM
149   LDM$S1       MACRO
150                MOVE      \2,\1
151                ENDM
152   RCNT$        EQU         898
153   SAMPLE       MOVEA.L     (A7)+,A0
154                MOVEA.L     (A7)+,A1
155                MOVE.L      A0,-(A7)
156                ADDA        #28,A1
157                MOVE        (A1)+,D0
158                ASL         #2,D0
159                MOVEA       D0,A0
160                TST         (A1)+
161                BEQ.S       SM1\@
162                RTS
163   SM1\@        MOVE        #1,-2(A1)
164                MOVE.L      RCNT,(A1)+
165                MOVE        (A1)+,D1
166                MOVE        (A1)+,D0
167                ADDA        A1,A0
168   SM2\@        MOVE.L      (A1)+,A2
169   SM3\@        MOVE        D0,D5
170                MOVE        (A2)+,(A0)+
171                SUBQ        #1,D5
```

```
172             BNE.S     SM3\@
173             SUBQ      #1,D1
174             BNE.S     SM2\@
175             RTS
176
177   *
178   *
179   * CONTROL AND INITIALIZATION
180             INITIAL$
181             DATASEG$
182   *
183   *
184   * FOREGROUND EXECUTIVE MAPS
185             ORG$      5120
186             DL$       0          *ACTIVE EXECUTIVE
187             DL$       BADADC.    *PRIORITY=1
188             DC$       0          *..BUSY FLAG
189             DC$       0          *..PENDING FLAG
190             DL$       0          *NOT USED
191             DL$       0          *NOT USED
192             DL$       0          *NOT USED
193             DL$       0          *NOT USED
194             DL$       0          *NOT USED
195             DL$       0          *NOT USED
196             DL$       0          *NOT USED
197             DL$       0          *NOT USED
198             DL$       0          *NOT USED
199             DL$       0          *NOT USED
200             DL$       0          *NOT USED
201             DL$       0          *NOT USED
202             DL$       0          *NOT USED
203             DL$       0          *NOT USED
204   *
205   *
206   * EXECUTABLE SEGMENT ENTRY ADDRESSES
207             ORG$      5376
208             DL$       MAIN.
209             DL$       BADADC.
210             DL$       GETDATA.
211   *
212   *
213   * SIMULATION TRANSFER MAPS
214             ORG$      5888
215   *   5888                       FROM DATAPROC.P.AN
216             DC$       -2          *...RESERVED
217             DC$       -1          *...END OF MAP
218   *   5892                       FROM DATAPROC.P.ANF
219             DC$       4           *..TO CORESIM RT-BUS
220             DC$       -2          *...RESERVED
221             DC$       -1          *...END OF MAP
222   *   5898                       FROM CORESIM.C.JOBDONE
223             DC$       0           *..TO DATAPROC RT-BUS
224             DC$       -2          *...RESERVED
225             DC$       -1          *...END OF MAP
226   *   5904                       FROM CORESIM.C.ACN
227             DC$       0           *..TO DATAPROC RT-BUS
228             DC$       -2          *...RESERVED
```

```
229                DC$       -1          *...END OF MAP
230   *    5910                          FROM CORESIM.C.ADN
231                DC$       0           *..TO DATAPROC RT-BUS
232                DC$       -2          *...RESERVED
233                DC$       -1          *...END OF MAP
234   *    5916                          FROM CORESIM.C.WDN
235                DC$       0           *..TO DATAPROC RT-BUS
236                DC$       -2          *...RESERVED
237                DC$       -1          *...END OF MAP
238   *    5922                          FROM CORESIM.C.PRC
239                DC$       0           *..TO DATAPROC I-BUS
240                DC$       -2          *...RESERVED
241                DC$       -1          *...END OF MAP
242   *    5928                          FROM CORESIM.P.FFNA
243                DC$       -2          *...RESERVED
244                DC$       -1          *...END OF MAP
245   *    5932                          FROM CORESIM.P.FLC
246                DC$       -2          *...RESERVED
247                DC$       -1          *...END OF MAP
248   *    5936                          FROM CORESIM.P.WDNC
249                DC$       -2          *...RESERVED
250                DC$       -1          *...END OF MAP
251   *    5940                          FROM CORESIM.P.PREPDONE
252                DC$       -2          *...RESERVED
253                DC$       -1          *...END OF MAP
254   *    5944                          FROM CORESIM.P.DUCTDONE
255                DC$       -2          *...RESERVED
256                DC$       -1          *...END OF MAP
257   *    5948                          FROM DUCTSIM.C.WDNB
258                DC$       4           *..TO CORESIM RT-BUS
259                DC$       -2          *...RESERVED
260                DC$       -1          *...END OF MAP
261   *    5954                          FROM DUCTSIM.C.PRD
262                DC$       0           *..TO DATAPROC I-BUS
263                DC$       -2          *...RESERVED
264                DC$       -1          *...END OF MAP
265   *    5960                          FROM DUCTSIM.P.FFNA
266                DC$       -2          *...RESERVED
267                DC$       -1          *...END OF MAP
268   *    5964                          FROM DUCTSIM.P.FLC
269                DC$       -2          *...RESERVED
270                DC$       -1          *...END OF MAP
271   *    5968                          FROM DUCTSIM.P.PREPDONE
272                DC$       -2          *...RESERVED
273                DC$       -1          *...END OF MAP
274   *
275   *
276   * CHANNEL TRANSFER MEMORY ALLOCATION
277                ORG$      7936
278   *    7936                          XFER IMAGE OF DATAPROC.P.AN
279                DC$       0           *..CALC FLAG
280                DC$       25600       *  ..S1 B+11
281                DC$       0           *..FILLER WORD
282                DC$       5888        *..XFER MAP ADDR
283   *    7944                          XFER IMAGE OF DATAPROC.P.ANF
284                DC$       0           *..CALC FLAG
285                DC$       20210       *  ..S1 B+11
```

```
286                DC$       0          *..FILLER WORD
287                DC$       5892       *..XFER MAP ADDR
288   *    7952                          XFER IMAGE OF CORESIM.C.JOBDONE
289                DC$       0          *..CALC FLAG
290                DC$       0          * ..BOOLEAN
291                DC$       0          *..FILLER WORD
292                DC$       5898       *..XFER MAP ADDR
293   *    7960                          XFER IMAGE OF CORESIM.C.ACN
294                DC$       0          *..CALC FLAG
295                DC$       0          * ..S1 B+10
296                DC$       0          *..FILLER WORD
297                DC$       5904       *..XFER MAP ADDR
298   *    7968                          XFER IMAGE OF CORESIM.C.ADN
299                DC$       0          *..CALC FLAG
300                DC$       0          * ..S1 B+10
301                DC$       0          *..FILLER WORD
302                DC$       5910       *..XFER MAP ADDR
303   *    7976                          XFER IMAGE OF CORESIM.C.WDN
304                DC$       0          *..CALC FLAG
305                DC$       0          * ..S1 B+9
306                DC$       0          *..FILLER WORD
307                DC$       5916       *..XFER MAP ADDR
308   *    7984                          XFER IMAGE OF CORESIM.C.PRC
309                DC$       0          *..CALC FLAG
310                DC$       16384      * ..S1 B+1
311                DC$       0          *..FILLER WORD
312                DC$       5922       *..XFER MAP ADDR
313   *    7992                          XFER IMAGE OF CORESIM.P.FFNA
314                DC$       0          *..CALC FLAG
315                DC$       0          * ..S1 B+1
316                DC$       0          *..FILLER WORD
317                DC$       5928       *..XFER MAP ADDR
318   *    8000                          XFER IMAGE OF CORESIM.P.FLC
319                DC$       0          *..CALC FLAG
320                DC$       13516      * ..S1 B+1
321                DC$       0          *..FILLER WORD
322                DC$       5932       *..XFER MAP ADDR
323   *    8008                          XFER IMAGE OF CORESIM.P.WDNC
324                DC$       0          *..CALC FLAG
325                DC$       0          * ..S1 B+2
326                DC$       0          *..FILLER WORD
327                DC$       5936       *..XFER MAP ADDR
328   *    8016                          XFER IMAGE OF CORESIM.P.PREPDONE
329                DC$       0          *..CALC FLAG
330                DC$       0          * ..BOOLEAN
331                DC$       0          *..FILLER WORD
332                DC$       5940       *..XFER MAP ADDR
333   *    8024                          XFER IMAGE OF CORESIM.P.DUCTDONE
334                DC$       0          *..CALC FLAG
335                DC$       0          * ..BOOLEAN
336                DC$       0          *..FILLER WORD
337                DC$       5944       *..XFER MAP ADDR
338   *    8032                          XFER IMAGE OF DUCTSIM.C.WDNB
339                DC$       0          *..CALC FLAG
340                DC$       0          * ..S1 B+5
341                DC$       0          *..FILLER WORD
342                DC$       5948       *..XFER MAP ADDR
```

```
343  *    8040                        XFER IMAGE OF DUCTSIM.C.PRD
344               DC$       0          *..CALC FLAG
345               DC$       16384      * ..S1 B+1
346               DC$       0          *..FILLER WORD
347               DC$       5954       *..XFER MAP ADDR
348  *    8048                        XFER IMAGE OF DUCTSIM.P.FFNA
349               DC$       0          *..CALC FLAG
350               DC$       0          * ..S1 B+1
351               DC$       0          *..FILLER WORD
352               DC$       5960       *..XFER MAP ADDR
353  *    8056                        XFER IMAGE OF DUCTSIM.P.FLC
354               DC$       0          *..CALC FLAG
355               DC$       13516      * ..S1 B+1
356               DC$       0          *..FILLER WORD
357               DC$       5964       *..XFER MAP ADDR
358  *    8064                        XFER IMAGE OF DUCTSIM.P.PREPDONE
359               DC$       0          *..CALC FLAG
360               DC$       0          * ..BOOLEAN
361               DC$       0          *..FILLER WORD
362               DC$       5968       *..XFER MAP ADDR
363  *
364  *
365  * LOCAL VARIABLE ALLOCATION
366               ORG$      10000
367  *
368  *
369  * PROGRAM CONSTANT ALLOCATION
370  *
371  *
372  * ARGUMENT GROUP ALLOCATION
373  *
374  DATA         DS$       5          *FOR OP-SYS USE
375               DN$       DATA****
376               DN$       DATAPROC
377               DC$       1
378               DC$       16         *ARGGROUP SIZE
379               DC$       0          *BUSY FLAG
380               DC$       0          *EXECUTION COUNT
381               DC$       7          *NUMBER OF ITEMS
382               DC$       1          *WORDS PER ITEM
383               DL$       7938       *..AN
384               DL$       7946       *..ANF
385               DL$       7962       *..ACN
386               DL$       7970       *..ADN
387               DL$       7986       *..PRC
388               DL$       8042       *..PRD
389               DL$       7978       *..WDN
390               DS$       18         *RESERVED
391               DS$       16         *RESERVED
392               DATAEND$
393  *
394  *
395  * DATAPROC EXECUTABLE SEGMENT
396  *
397  * EXECUTIVE: MAIN.     PRIORITY=0
398  MAIN.        BACKEXEC  MAIN.,82
399  S$1          ENTER$    GETDATA.
```

```
400  S$2        RETURN$E MAIN.
401  *
402  * EXECUTIVE: BADADC.  PRIORITY=1
403  BADADC.    FOREEXEC BADADC.,82
404  S$3        TSTXVA$  7938,C     *AN
405             LDM$S1   D0,7938     *B+11
406             HLD$S1   D0
407             LDI$S1   D1,800      *MINAREA,B+11
408             JNEQ$S1  S$5,D1
409  S$4        ADVISE$H 1,C     *ADCMESS1
410             EXIT$    S$6
411  S$5        ADVISE$H 2,C     *ADCMESS2
412  S$6        RETURN$I BADADC.
413  *
414  * TASK: GETDATA.
415             DC$      7938       *XVAR ADDRESS
416             DC$      7946       *XVAR ADDRESS
417             DC$      7962       *XVAR ADDRESS
418             DC$      7970       *XVAR ADDRESS
419             DC$      7986       *XVAR ADDRESS
420             DC$      8042       *XVAR ADDRESS
421             DC$      7978       *XVAR ADDRESS
422             DC$      7          *NUMBER OF XVARS
423             DC$      1          *TASK ENABLE
424             DC$      0          *TASK COMPLETE
425  GETDATA.   DS$      2          *ENTRY OVERHEAD
426  S$7        CALL$    SAMPLE,DATA
427  S$8        ADVISE$R 1,C     *DATA
428  S$9        RETURN$T GETDATA.
429  *
430  *
431  * TARGET LIBRARY PROCEDURES
432             SAMPLE
433             END
434
435
436
437
438
439
440
441             ****RTMPL ERRORS: 0
442             ****RTMPL WARNINGS: 0
```

```
 1  * RTMPL: DUALNOZZ
 2  *        DALE J. ARPASI
 3  * PROGRAM: DATAPROC - PREP
 4  * FUNCTION: RTX
 5  * GENERATED: 11/27/84  10:28:52
 6  *
 7  *
 8  * REQUIRED TARGET MACRO FILE
 9  INITIAL$ MACRO
10  .CMB     EQU     $81C
11  .AVT     EQU     $844
12  .AVX     EQU     $845
13  .AVS     EQU     $846
14  .AVB     EQU     $847
15  .AWC     EQU     $84C
16  .NOFE    EQU     ($918)*2
17  .XA1$P   EQU     ($9B2)*2
18  .XA1$C   EQU     ($9B6)*2
19  .XA2$P   EQU     ($9BA)*2
20  .XV2$P   EQU     ($9BC)*2
21  .XA2$C   EQU     ($9C2)*2
22  .XV2$C   EQU     ($9C4)*2
23  .FE$P    EQU     ($9C8)*2
24  .FE$C    EQU     ($9CC)*2
25  .AK$P    EQU     ($9D4)*2
26  .AK$C    EQU     ($9D6)*2
27  .PB$P    EQU     ($9D8)*2
28  .PB$C    EQU     ($9D9)*2
29  .PI4B    EQU     ($9DA)*2
30  .EAD     EQU     $1400
31  .XVM     EQU     $1F00
32  .XVC     EQU     $1F01
33           ENDM
34  ORG$     MACRO
35           ORG     \1
36           ENDM
37  DC$      MACRO
38           DC      \1
39           ENDM
40  DS$      MACRO
41           DS      \1
42           ENDM
43  DL$      MACRO
44           DC.L    \1
45           ENDM
46  DN$      MACRO
47           DC.L    '\1'
48           ENDM
49  DATASEG$ MACRO
50           ENDM
51  DATAEND$ MACRO
52           ENDM
53  STV$S1   MACRO
54           MOVEP   \2,((\1-.XVM)*2)(A4)
55           MOVE.B  D7,((\1-.XVC)*2)(A4)
56           MOVEA   \1+4,A0
57           TST     (A0)
```

```
 58                 BMI.S    SV2\@+2
 59  SV1\@          MOVEA    (A0)+,A1
 60                 ADDA     #.CHB,A1
 61                 TRAPV
 62                 MOVEA.L  (A1),A1
 63                 ADDA     #\1,A1
 64                 TRAPV
 65                 MOVE     \2,(A1)
 66                 MOVE     D7,-(A1)
 67                 TST      (A0)
 68  SV2\@          BPL.S    SV1\@
 69                 ENDM
 70  TSTXVL$        MACRO
 71                 CMP.B    ((\1-.XVC)*2)(A4),D7
 72                 BEQ.S    TL3\@+4
 73                 TST.B    ((\1-.XVC)*2)(A4)
 74                 BEQ.S    TL3\@+4
 75                 MOVEQ    #0,D5
 76  TL1\@          ADDQ     #1,D5
 77                 TRAPV
 78                 CMPA     #0,A4
 79                 CMP.B    (\1-1),D7
 80                 NOP
 81                 BNE.S    TL1\@
 82                 CMP      (\1+$1D100),D5
 83                 BLE.S    TL2\@
 84                 MOVE     D5,\1+$1D100
 85  TL2\@          MOVE.L   \1,D5
 86                 MOVEP.L  D5,((\1-.XVM)*2)(A4)
 87  TL3\@          MOVE.B   D7,((\1-.XVC)*2)(A4)
 88                 ENDM
 89  ACTIVAT$       MACRO
 90  ACT\@          TST.B    .NOFE(A4)
 91                 NOP
 92                 BNE.S    ACT\@
 93                 MOVE     #\1,D0
 94                 IFC      '\2','P'
 95                 MOVEP    D0,.FE$C(A4)
 96  AC\@           TST.B    .PB$C(A4)
 97                 NOP
 98                 BNE.S    AC\@
 99                 MOVE.B   #1,.PB$C(A4)
100                 OR.B     #3,D7
101                 MOVE.B   D7,.AK$P(A4)
102                 ANDI.B   #$FB,D7
103                 MOVE.B   D7,$1800(A4)
104                 ORI.B    #4,D7
105 ACA\@           TST.B    .AK$P(A4)
106                 NOP
107                 BNE.S    ACA\@
108                 MOVE.B   D7,$1800(A4)
109                 ENDC
110                 IFC      '\2','C'
111                 MOVEP    D0,.FE$P(A4)
112 AC\@           TST.B    .PB$P(A4)
113                 NOP
114                 BNE.S    AC\@
```

```
115                MOVE.B  #1,.PB$P(A4)
116                MOVE.B  #3,.AK$C(A4)
117                AND.B   #$FB,D7
118                MOVE.B  D7,$1800(A4)
119                ORI.B   #4,D7
120   ACA\@        TST.B   .AK$C(A4)
121                NOP
122                BNE.S   ACA\@
123                MOVE.B  D7,$1800(A4)
124                ENDC
125                ENDM
126   ADVISE$H     MACRO
127                MOVE.B  #\1,D5
128                TRAP    #9
129                ENDM
130   DISPATC$     MACRO
131                MOVE    #\2,NT\@
132                BRA.S   DP1\@
133   NT\@         DC      0
134   SA0\@        DC.L    0
135   DP1\@        LEA     (\1+(4*(\2-1))),A0
136   DP2\@        MOVEA.L (A0),A1
137                TST     -(A1)
138                BNE.S   DP4\@
139                TST     -(A1)
140                BEQ.S   DP6\@
141   DP3\@        MOVEA   -(A1),A2
142                BEQ.S   DP5\@
143                IFC     '\3','C'
144                BTST.L  #31,D7
145                BNE.S   DP3\@
146                CMP.B   -1(A2),D7
147                BEQ.S   DP3\@
148                MOVE.L  A2,D5
149                SUB     #.XVM,D5
150                TRAPV
151                LSL.L   #1,D5
152                TRAPV
153                MOVEA.L D5,A5
154                ADDA.L  A4,A5
155                TRAPV
156                CMP.B   -2(A5)),D7
157                BEQ.S   DPY\@
158                MOVE    A2,D5
159                MOVEP   D5,.XA1$P(A4)
160   DPZ\@        TST.B   .PB$P(A4)
161                NOP
162                BNE.S   DPZ\@
163                MOVE.B  #1,.PB$P(A4)
164                MOVE.B  #1,.AK$C(A4)
165                ANDI.B  #$FE,D7
166                MOVE.B  D7,$1800(A4)
167                ORI.B   #1,D7
168   DPI\@        TST.B   .AK$C(A4)
169                NOP
170                BNE.S   DPI\@
171                MOVE.B  D7,$1800(A4)
```

```
172                 BRA.S   DP4\@
173  DPY\@          MOVEP.L 0(A5),D5
174                 MOVE.L  D5,(A2)
175                 MOVE.B  D7,-1(A2)
176                 ENDC
177                 IFC     '\3','P'
178                 BTST.L  #31,D7
179                 BNE.S   DP3\@
180                 CMP.B   -1(A2),D7
181                 BNE.S   DP4\@
182                 MOVE.L  A2,D5
183                 SUB     #.XVM,D5
184                 TRAPV
185                 LSL.L   #1,D5
186                 TRAPV
187                 MOVEA.L D5,A5
188                 ADDA.L  A4,A5
189                 TRAPV
190                 MOVEP.L 0(A5),D5
191                 MOVE.L  D5,(A2)
192                 MOVE.B  D7,-1(A2)
193                 ENDC
194                 BRA.S   DP3\@
195  DP4\@          LEA     \1,A2
196                 CMPA.L  A2,A0
197                 BEQ.S   DP1\@
198                 SUBA    #4,A0
199                 TRAPV
200                 BRA.S   DP2\@
201  DP5\@          MOVE.L  A0,SA0\@
202                 JSR     4(A0)
203                 MOVEA.L SA0\@,A0
204  DP6\@          MOVEA.L (A0),A1
205                 MOVE    #1,-2(A1)
206                 SUB     #1,NT\@
207                 TRAPV
208                 BGT.S   DP4\@
209                 LEA     \1,A2
210                 LEA     (\1+(4*(\2-1))),A0
211  DP7\@          MOVEA.L (A0),A1
212                 SUBA    #4,A0
213                 TRAPV
214                 MOVE    #0,-2(A1)
215                 CMPA.L  A2,A0
216                 BPL.S   DP7\@
217                 ENDM
218  RETURN$E       MACRO
219                 RTS
220                 ENDM
221  RETURN$T       MACRO
222                 RTS
223                 ENDM
224  BACKEXEC       MACRO
225                 LEA     \1-\2,A3
226                 ENDM
227  ENTER$         MACRO
228                 TST     \1-4
```

```
229             BEQ.S   *+4
230             JSR     \1+4
231             ENDM
232 REDO$       MACRO
233             JMP     \1
234             ENDM
235 EXIT$       MACRO
236             JMP     \1
237             ENDM
238 CALL$       MACRO
239             IFNC    '','\2'
240             LEA     \2,A0
241             MOVE.L  A0,-(A7)
242             ENDC
243             IFNC    '','\3'
244             LEA     \3,A0
245             MOVE.L  A0,-(A7)
246             ENDC
247             IFNC    '','\4'
248             LEA     \4,A0
249             MOVE.L  A0,-(A7)
250             ENDC
251             IFNC    '','\5'
252             LEA     \5,A0
253             MOVE.L  A0,-(A7)
254             ENDC
255             JSR     \1
256             ENDM
257 JTR$        MACRO
258             TST     \2
259             BNE     \1
260             ENDM
261 HLD$S1      MACRO
262             MOVE    \1,-(A7)
263             ENDM
264 JNGT$S1     MACRO
265             CMP     (A7)+,\2
266             BGE     \1
267             ENDM
268 JNLT$S1     MACRO
269             CMP     (A7)+,\2
270             BLE     \1
271             ENDM
272 ADD$S1RI    MACRO
273             ADDI    #\3,\2
274             TRAPV
275             ENDM
276 MUL$S2IR    MACRO
277             MULS    #\2,\3
278             ASL.L   #1,\3
279             TRAPV
280             ENDM
281 NOT$BVR     MACRO
282             NOT     \1
283             ENDM
284 SUB$S1IR    MACRO
285             SUBI    #\2,\3
```

```
286                 TRAPV
287                 NEG       \3
288                 TRAPV
289                 ENDM
290   CVP$S21       MACRO
291                 SWAP      \2
292                 ENDM
293   LDI$S1        MACRO
294                 MOVE      #\2,\1
295                 ENDM
296   LDM$S1        MACRO
297                 MOVE      \2,\1
298                 ENDM
299   LDM$BV        MACRO
300                 MOVE      \2,\1
301                 ENDM
302   SCL$S1R       MACRO
303                 IFLE      \2-8
304                 ASR       #\2,\1
305                 ENDC
306                 IFGT      \2-8
307                 MOVE      #\2,D5
308                 ASR       D5,\1
309                 ENDC
310                 ENDM
311   SLV$S1        MACRO
312                 TST       D7
313                 BPL.S     SL1\@
314                 TST       \1-2
315                 BEQ.S     SL1\@
316                 MOVE      \1,\2
317                 BRA.S     SL1\@+4
318   SL1\@         MOVE      \2,\1
319                 ENDM
320   READ          MOVEA.L     (A7)+,A0
321                 MOVEA.L     (A7)+,A1
322                 MOVEA.L     (A7)+,A1
323                 MOVE.L      A0,-(A7)
324                 RTS
325   DAC1          MOVEA.L     (A7)+,A0
326                 MOVEA.L     (A7)+,A1
327                 MOVE.L      A0,-(A7)
328                 RTS
329   DAC2          MOVEA.L     (A7)+,A0
330                 MOVEA.L     (A7)+,A1
331                 MOVE.L      A0,-(A7)
332                 RTS
333   *
334   *
335   * CONTROL AND INITIALIZATION
336                 INITIAL$
337                 DATASEG$
338   *
339   *
340   * FOREGROUND EXECUTIVE MAPS
341   * ...NONE REQUIRED
342   *
```

```
343  *
344  * EXECUTABLE SEGMENT ENTRY ADDRESSES
345              ORG$      5376
346              DL$       GETAN.
347              DL$       WRTACN.
348              DL$       WRTADN.
349              DL$       JOBDONE.
350  *
351  *
352  * SIMULATION TRANSFER MAPS
353              ORG$      5888
354  *   5888                          FROM DATAPROC.P.AN
355              DC$       -2           *...RESERVED
356              DC$       -1           *...END OF MAP
357  *   5892                          FROM DATAPROC.P.ANF
358              DC$       4            *..TO CORESIM RT-BUS
359              DC$       -2           *...RESERVED
360              DC$       -1           *...END OF MAP
361  *   5898                          FROM CORESIM.C.JOBDONE
362              DC$       0            *..TO DATAPROC RT-BUS
363              DC$       -2           *...RESERVED
364              DC$       -1           *...END OF MAP
365  *   5904                          FROM CORESIM.C.ACN
366              DC$       0            *..TO DATAPROC RT-BUS
367              DC$       -2           *...RESERVED
368              DC$       -1           *...END OF MAP
369  *   5910                          FROM CORESIM.C.ADN
370              DC$       0            *..TO DATAPROC RT-BUS
371              DC$       -2           *...RESERVED
372              DC$       -1           *...END OF MAP
373  *   5916                          FROM CORESIM.C.WDN
374              DC$       0            *..TO DATAPROC RT-BUS
375              DC$       -2           *...RESERVED
376              DC$       -1           *...END OF MAP
377  *   5922                          FROM CORESIM.C.PRC
378              DC$       0            *..TO DATAPROC I-BUS
379              DC$       -2           *...RESERVED
380              DC$       -1           *...END OF MAP
381  *   5928                          FROM CORESIM.P.FFNA
382              DC$       -2           *...RESERVED
383              DC$       -1           *...END OF MAP
384  *   5932                          FROM CORESIM.P.FLC
385              DC$       -2           *...RESERVED
386              DC$       -1           *...END OF MAP
387  *   5936                          FROM CORESIM.P.WDNC
388              DC$       -2           *...RESERVED
389              DC$       -1           *...END OF MAP
390  *   5940                          FROM CORESIM.P.PREPDONE
391              DC$       -2           *...RESERVED
392              DC$       -1           *...END OF MAP
393  *   5944                          FROM CORESIM.P.DUCTDONE
394              DC$       -2           *...RESERVED
395              DC$       -1           *...END OF MAP
396  *   5948                          FROM DUCTSIM.C.WDNB
397              DC$       4            *..TO CORESIM RT-BUS
398              DC$       -2           *...RESERVED
399              DC$       -1           *...END OF MAP
```

```
400  *   5954                        FROM DUCTSIM.C.PRD
401              DC$       0          *..TO DATAPROC I-BUS
402              DC$       -2         *...RESERVED
403              DC$       -1         *...END OF MAP
404  *   5960                        FROM DUCTSIM.P.FFNA
405              DC$       -2         *...RESERVED
406              DC$       -1         *...END OF MAP
407  *   5964                        FROM DUCTSIM.P.FLC
408              DC$       -2         *...RESERVED
409              DC$       -1         *...END OF MAP
410  *   5968                        FROM DUCTSIM.P.PREPDONE
411              DC$       -2         *...RESERVED
412              DC$       -1         *...END OF MAP
413  *
414  *
415  * CHANNEL TRANSFER MEMORY ALLOCATION
416              ORG$      7936
417  *   7936                        XFER IMAGE OF DATAPROC.P.AN
418              DC$       0          *..CALC FLAG
419              DC$       25600      *  ..S1 B+11
420              DC$       0          *..FILLER WORD
421              DC$       5888       *..XFER MAP ADDR
422  *   7944                        XFER IMAGE OF DATAPROC.P.ANF
423              DC$       0          *..CALC FLAG
424              DC$       20210      *  ..S1 B+11
425              DC$       0          *..FILLER WORD
426              DC$       5892       *..XFER MAP ADDR
427  *   7952                        XFER IMAGE OF CORESIM.C.JOBDONE
428              DC$       0          *..CALC FLAG
429              DC$       0          *  ..BOOLEAN
430              DC$       0          *..FILLER WORD
431              DC$       5898       *..XFER MAP ADDR
432  *   7960                        XFER IMAGE OF CORESIM.C.ACN
433              DC$       0          *..CALC FLAG
434              DC$       0          *  ..S1 B+10
435              DC$       0          *..FILLER WORD
436              DC$       5904       *..XFER MAP ADDR
437  *   7968                        XFER IMAGE OF CORESIM.C.ADN
438              DC$       0          *..CALC FLAG
439              DC$       0          *  ..S1 B+10
440              DC$       0          *..FILLER WORD
441              DC$       5910       *..XFER MAP ADDR
442  *   7976                        XFER IMAGE OF CORESIM.C.WDN
443              DC$       0          *..CALC FLAG
444              DC$       0          *  ..S1 B+9
445              DC$       0          *..FILLER WORD
446              DC$       5916       *..XFER MAP ADDR
447  *   7984                        XFER IMAGE OF CORESIM.C.PRC
448              DC$       0          *..CALC FLAG
449              DC$       16384      *  ..S1 B+1
450              DC$       0          *..FILLER WORD
451              DC$       5922       *..XFER MAP ADDR
452  *   7992                        XFER IMAGE OF CORESIM.P.FFNA
453              DC$       0          *..CALC FLAG
454              DC$       0          *  ..S1 B+1
455              DC$       0          *..FILLER WORD
456              DC$       5928       *..XFER MAP ADDR
```

```
457  *    8000                        XFER IMAGE OF CORESIM.P.FLC
458               DC$       0          *..CALC FLAG
459               DC$       13516      * ..S1 B+1
460               DC$       0          *..FILLER WORD
461               DC$       5932       *..XFER MAP ADDR
462  *    8008                        XFER IMAGE OF CORESIM.P.WDNC
463               DC$       0          *..CALC FLAG
464               DC$       0          * ..S1 B+2
465               DC$       0          *..FILLER WORD
466               DC$       5936       *..XFER MAP ADDR
467  *    8016                        XFER IMAGE OF CORESIM.P.PREPDONE
468               DC$       0          *..CALC FLAG
469               DC$       0          * ..BOOLEAN
470               DC$       0          *..FILLER WORD
471               DC$       5940       *..XFER MAP ADDR
472  *    8024                        XFER IMAGE OF CORESIM.P.DUCTDONE
473               DC$       0          *..CALC FLAG
474               DC$       0          * ..BOOLEAN
475               DC$       0          *..FILLER WORD
476               DC$       5944       *..XFER MAP ADDR
477  *    8032                        XFER IMAGE OF DUCTSIM.C.WDNB
478               DC$       0          *..CALC FLAG
479               DC$       0          * ..S1 B+5
480               DC$       0          *..FILLER WORD
481               DC$       5948       *..XFER MAP ADDR
482  *    8040                        XFER IMAGE OF DUCTSIM.C.FRD
483               DC$       0          *..CALC FLAG
484               DC$       16384      * ..S1 B+1
485               DC$       0          *..FILLER WORD
486               DC$       5954       *..XFER MAP ADDR
487  *    8048                        XFER IMAGE OF DUCTSIM.P.FFNA
488               DC$       0          *..CALC FLAG
489               DC$       0          * ..S1 B+1
490               DC$       0          *..FILLER WORD
491               DC$       5960       *..XFER MAP ADDR
492  *    8056                        XFER IMAGE OF DUCTSIM.P.FLC
493               DC$       0          *..CALC FLAG
494               DC$       13516      * ..S1 B+1
495               DC$       0          *..FILLER WORD
496               DC$       5964       *..XFER MAP ADDR
497  *    8064                        XFER IMAGE OF DUCTSIM.P.PREPDONE
498               DC$       0          *..CALC FLAG
499               DC$       0          * ..BOOLEAN
500               DC$       0          *..FILLER WORD
501               DC$       5968       *..XFER MAP ADDR
502  *
503  *
504  * LOCAL VARIABLE ALLOCATION
505               ORG$      10000
506               DC$       0          *AN H-LTCH
507  AN           DC$       25600      * ..S1 B+11
508               DC$       0          *ANF H-LTCH
509  ANF          DC$       20210      * ..S1 B+11
510  *
511  *
512  * PROGRAM CONSTANT ALLOCATION
513  K1           DC$       1          * I1
```

```
514  *
515  *
516  * DISPATCH TASKLIST ALLOCATION
517  TL$1        DL$       WRTACN.
518              DL$       WRTADN.
519  *
520  *
521  * ARGUMENT GROUP ALLOCATION
522  *
523  ACNG        DS$       5          *FOR OP-SYS USE
524              DN$       ACNG****
525              DN$       DATAPROC
526              DC$       2
527              DC$       1          *ARGGROUP SIZE
528              DC$       0          *BUSY FLAG
529              DC$       0          *EXECUTION COUNT
530              DC$       1          *NUMBER OF ITEMS
531              DC$       1          *WORDS PER ITEM
532              DL$       7962       *..ACN
533              DS$       1          *RESERVED
534  *
535  ADNG        DS$       5          *FOR OP-SYS USE
536              DN$       ADNG****
537              DN$       DATAPROC
538              DC$       2
539              DC$       1          *ARGGROUP SIZE
540              DC$       0          *BUSY FLAG
541              DC$       0          *EXECUTION COUNT
542              DC$       1          *NUMBER OF ITEMS
543              DC$       1          *WORDS PER ITEM
544              DL$       7970       *..ADN
545              DS$       1          *RESERVED
546  *
547  ADCVAR      DS$       5          *FOR OP-SYS USE
548              DN$       ADCVAR**
549              DN$       DATAPROC
550              DC$       2
551              DC$       32         *ARGGROUP SIZE
552              DC$       0          *BUSY FLAG
553              DC$       0          *EXECUTION COUNT
554              DC$       1          *NUMBER OF ITEMS
555              DC$       1          *WORDS PER ITEM
556              DL$       10002      *..AN
557              DS$       62         *RESERVED
558              DS$       32         *RESERVED
559  *
560  ADCCHN      DS$       5          *FOR OP-SYS USE
561              DN$       ADCCHN**
562              DN$       DATAPROC
563              DC$       2
564              DC$       32         *ARGGROUP SIZE
565              DC$       0          *BUSY FLAG
566              DC$       0          *EXECUTION COUNT
567              DC$       1          *NUMBER OF ITEMS
568              DC$       1          *WORDS PER ITEM
569              DL$       10008      *..K1
570              DS$       62         *RESERVED
```

```
571                DS$       32          *RESERVED
572                DATAEND$
573   *
574   *
575   * DATAPROC EXECUTABLE SEGMENT
576   *
577   * EXECUTIVE: GETAN.    PRIORITY=0
578   GETAN.       BACKEXEC  GETAN.,74
579   S$1          CALL$     READ,ADCVAR,ADCCHN
580   S$2          LDM$S1    D0,AN       *B+11
581                ADD$S1RI  D0,D0,12800      *K2,B+11
582                SLV$S1    AN,D0       *B+11
583                STV$S1    7938,D0       *LBUSXFER
584   S$3          HLD$S1    D0
585                LDI$S1    D1,800       *MINAREA,B+11
586                JNLT$S1   S$6,D1
587   S$4          LDI$S1    D2,800       *MINAREA,B+11
588                SLV$S1    AN,D2       *B+11
589                STV$S1    7938,D2       *LBUSXFER
590   S$5          ACTIVAT$  5124,P       *BADADC.
591                EXIT$     S$9
592   S$6          HLD$S1    D0
593                LDI$S1    D3,25600       *MAXAREA,B+11
594                JNGT$S1   S$9,D3
595   S$7          LDI$S1    D4,25600       *MAXAREA,B+11
596                SLV$S1    AN,D4       *B+11
597                STV$S1    7938,D4       *LBUSXFER
598   S$8          ACTIVAT$  5124,P       *BADADC.
599   S$9          LDM$S1    D0,AN       *B+11
600                MUL$S2IR  D0,21770,D0      *K1P622M4,B-1
601                CVF$S21   D0,D0       *B-1
602                SCL$S1R   D0,12       *B+11
603                SUB$S1IR  D0,16,D0       *SC$1,B+11
604                SLV$S1    ANF,D0      *B+11
605                STV$S1    7946,D0       *LBUSXFER
606   S$10         ENTER$    JOBDONE.
607   S$11         DISPATC$  TL$1,2,P
608   S$12         RETURN$E  GETAN.
609   *
610   * TASK: WRTACN.
611                DC$       7962        *XVAR ADDRESS
612                DC$       1           *NUMBER OF XVARS
613                DC$       1           *TASK ENABLE
614                DC$       0           *TASK COMPLETE
615   WRTACN.      DS$       2           *ENTRY OVERHEAD
616   S$13         CALL$     DAC1,ACNG
617   S$14         RETURN$T  WRTACN.
618   *
619   * TASK: WRTADN.
620                DC$       7970        *XVAR ADDRESS
621                DC$       1           *NUMBER OF XVARS
622                DC$       1           *TASK ENABLE
623                DC$       0           *TASK COMPLETE
624   WRTADN.      DS$       2           *ENTRY OVERHEAD
625   S$15         CALL$     DAC2,ADNG
626   S$16         RETURN$T  WRTADN.
627   *
```

```
628  * TASK: JOBDONE.
629              DC$       7954        *XVAR ADDRESS
630              DC$       1           *NUMBER OF XVARS
631              DC$       1           *TASK ENABLE
632              DC$       0           *TASK COMPLETE
633  JOBDONE.    DS$       2           *ENTRY OVERHEAD
634  TEST        TSTXVL$   7954,P       *JOBDONE
635              LDM$BV    D0,7954
636              NOT$BVR   D0,D0
637              JTR$      S$19,D0
638  S$18        REDO$     TEST
639              EXIT$     S$20
640  S$19        ADVISE$H  8,P      *DPMESS
641  S$20        RETURN$T  JOBDONE.
642  *
643  *
644  * TARGET LIBRARY PROCEDURES
645              READ
646              DAC1
647              DAC2
648              END
649
650
651
652
653
654
655
656              ****RTMPL ERRORS: 0
657              ****RTMPL WARNINGS: 0
```

References

1. Blech, Richard A.; and Arpasi, Dale J.: Hardware for a Real-Time Multiprocessor Simulator. NASA TM-83805, 1984.
2. Cole, Gary L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator. NASA TM-83605, 1984.
3. Cole, Gary L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator—Users Manual. NASA TP-2426, 1984.
4. Arpasi, Dale J.: RTMPL—A Structured Programming and Documentation Utility for Real-Time Multiprocessor Simulations. NASA TM-83606, 1984.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TP-2422		

4. Title and Subtitle	5. Report Date
Real-Time Multiprocessor Programming Language (RTMPL) - Users Manual	June 1985
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Dale J. Arpasi	E-1999
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Paper
National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

A real-time multiprocessor programming language (RTMPL) has been developed to provide for high-order programming of real-time simulations on systems of distributed computers. RTMPL is a structured, engineering-oriented language. The RTMPL utility supports a variety of multiprocessor configurations and types by generating assembly language programs according to user-specified targeting information. Many programming functions are assumed by the utility (e.g., data transfer and scaling) to reduce the programming chore. This manual describes RTMPL from a user's viewpoint. Source generation, applications, utility operation, and utility output are detailed. An example simulation is generated to illustrate many RTMPL features.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Multiprocessor; Simulation	Unclassified - Unlimited STAR Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	112	A06

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1985

National Aeronautics and
Space Administration

Washington, D.C.
20546

Official Business
Penalty for Private Use, $300

THIRD-CLASS BULK RATE

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

NASA

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return