A REAL TIME DIGITAL SIMULATION SUPERVISOR

by

Jeff Ira Cleveland II

B.S.E.E. Texas College of Arts and Industries, 1963

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

April 1970

Thesis directed by

Maurice K. Morin

Professorial Lecturer in Engineering

ABSTRACT

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

In an effort to simplify the programing and operation of digital simulation applications, a "real time digital simulation supervisor," hereinafter referred to as "the supervisor," has been developed. Its major function is to perform all real time input/output control, timing synchronization, communication and control, and other system oriented functions peculiar to real time operation.

The major objective of this work is to simplify the programing and planning tasks associated with flight simulation investigations to the level of conventional Fortran programing without sacrificing the interactive flexibility required to perform digital simulation.

A general description of the computer complex with particular emphasis on the digital simulation subsystem is presented. The general requirements and structure of the supervisor are discussed, along with an overview of its implementation. The structure of a simulation

applications program is discussed and the man-machine control interface through the supervisor is described.

Special problem areas and special techniques developed to solve the problems are presented. The areas discussed are real time program control, real time data recording, time critical supervisor operation, and error recovery and diagnostics. This discussion illustrates the manner in which the simulation applications program is able, through a series of simple subroutine calls, to form a unified code structure that will perform the task of real time digital simulation.

ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to Mr. George C. Salley of National Aeronautics and Space Administration, Langley Research Center, who wrote the first version of the real time digital simulation supervisor and without whose assistance this project could not have been completed. The author would also like to thank Mr. Maurice K. Morin for his guidance and patience in the preparation of this work. Last, but certainly not least, a debt of gratitude is owed to the author's wife, Ruth, who patiently waited as this work was completed.

TABLE OF CONTENTS

PAGE

ABSTRACT ii

ACKNOWLEDGEMENTS iv

LIST OF FIGURES vi

LIST OF SYMBOLS viii

TEXT

Chapter

I. INTRODUCTION 1

II. SYSTEM HARDWARE AND SOFTWARE 3

III. REQUIREMENTS OF A REAL TIME DIGITAL SIMULATION SUPERVISOR 12

IV. GENERAL OVERVIEW OF IMPLEMENTATION 18

V. SIMULATION APPLICATION PROGRAM STRUCTURE 21

VI. MODE CONTROLS 26

VII. REAL TIME DATA RECORDING 32

VIII. TIME CRITICAL OPERATIONS 36

IX. ERROR RECOVERY AND DIAGNOSTICS 41

X. CONCLUDING REMARKS 47

APPENDIX

Chapter

I. HARDWARE CONFIGURATION 49

II. OPERATING SYSTEM SOFTWARE 64

BIBLIOGRAPHY 72

LIST OF FIGURES

FIGURE		PAGE
1.	Program Control Panel	5
2.	CDC 6000 Computer Organization	7
3.	Computer Organization with Real Time Digital Simulation	9
4.	Central Processor Time Allocation with Three Real Time Simulation Jobs	11
5.	Real Time Simulation Supervisor Block Diagram	19
6.	Simulation Applications Program Block Structure	22
7.	Simulation Applications Program Initialization Section	24
8.	Typical Program Flow for Real Time Modes	27
9.	Typical Block Sequences during Real Time Operation	29
10.	Linkage Diagram of Block Dispatcher	31
11.	Typical File Environment Table and Data Buffer	33
12.	Use of Masks for Packed Discrete Manipulation	38
13.	Supervisor Special Execution at Lost Time Synchronization Interrupt	42
A1.	Langley Research Center Computer Complex	50
A2.	Batch Processing Subsystem and Decentralized Operation	54
A3.	Auxiliary Storage Subsystem	55
A4.	Real Time Simulation Subsystems	58
A5.	Remote Terminal Subsystems	61
A6.	On-Line Subsystem	63

FIGURE PAGE

A7. Operating System Software 65

A8. 6000 System 68

A9. Operational Environment 70

LIST OF SYMBOLS

ACD	Langley Research Center Analysis and Computation Division
ADC	analog-to-digital converter
ADCON	analog-to-digital controller
ADDIS	analog-to-digital and discrete input system
CDC	Control Data Corporation
CPU	central processing unit
CRT	cathode ray tube
DAC	digital-to-analog converter
DACON	digital-to-analog controller
DADOS	digital-to-analog and discrete output system
FET	file environment table
I/O	input/output
LRC	Langley Research Center
MOPS	million operations per second
PP	peripheral processor
RCT	requested compute time
RTM	real time monitor
RTSS	real time simulation subsystem
SCOPE	simultaneous computing and operation of peripheral equipment
SKED	real time scheduling program

CHAPTER I

INTRODUCTION

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

In an effort to simplify the programing and operation of digital simulation applications, a "real time digital simulation supervisor," hereinafter referred to as "the supervisor," has been developed. Its major function is to perform all real time input/output control, timing synchronization, communication and control, and other system oriented functions peculiar to real time operation.

The major objective of this work is to simplify the programing and planning tasks associated with flight simulation investigations to the level of conventional Fortran programing without sacrificing the interactive flexibility required to perform digital simulation.

A general description of the computer complex with particular emphasis on the digital simulation subsystem is presented. The general requirements and structure of the supervisor are discussed, along with

an overview of its implementation. Areas which required special techniques are discussed along with a description of the techniques employed.

CHAPTER II

SYSTEM HARDWARE AND SOFTWARE

The National Aeronautics and Space Administration Langley Research Center Computer Complex consists of four CDC 6000 series digital computers with associated subsystems of peripheral equipment. As seen in the Appendix, several different types of computer subsystems are integrated into the complex. It is in this environment that real time digital simulation must be performed. Time synchronization and problem solution rates must be guaranteed while allowing other functions of the complex, such as batch, conversational CRT, and remote terminal processing, to be performed on the same computer at the same time.

To perform digital simulation, two special Real Time Simulation Hardware Subsystems (RTSS) are employed to perform the real time input/output and timing that are necessary. Each RTSS consists of a real time clock, an analog-to-digital input subsystem (ADDIS), and a digital-to-analog output analog system (DADOS). Each RTSS has a complement of analog-to-digital (ADC) and digital-to-analog (DAC) converters and a set of discrete input and output channels for event sensing, control, and status indications.

An integral part of each RTSS is a set of program control stations. Each station consists of a simulation control console, a CRT interactive display console, a typewriter for printing short messages, and associated analog recording devices. Figure 1 illustrates the control panel of a simulation control console. Each panel has a set of

switches which are connected to discrete input channels. The set of function sense switches are used for programer inputs to the simulation program while the set of mode control switches are for specific flow control and functional requests. The data entry keyboard with the digital decimal display comprise a means of data entry into the computer and a means of displaying values to the programer. At the left of the figure, the white and red indicators are used to denote logical status and event occurrence in the simulation program. In the lower right are potentiometers that are connected to ADC input channels to give a "twiddle" capability. With this complement of switches and indicators, the programer is able to control the simulation interactively.

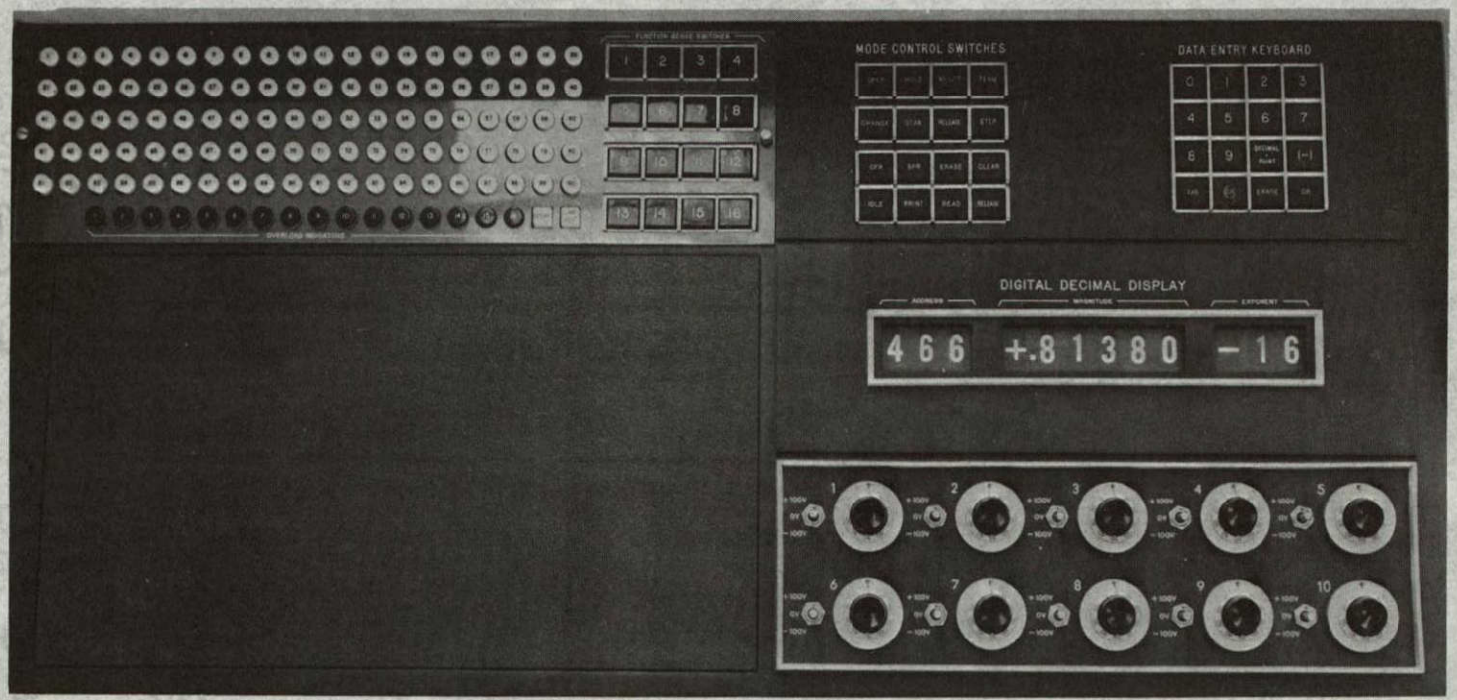


Figure 1. - Program control panel.

CDC 6000 Computer Organization

Figure 2 illustrates typical CDC 6000 computer organization with the SCOPE (Simultaneous Computing and Operation of Peripheral Equipment) operating system. Central memory is partitioned by software into eight areas called control points. Each control point can function independently of the others and one job occupies one control point which gives a multiprograming capability. Control point 0 is reserved for the use of the operating system and contains tables and pointers for the SCOPE system.

Associated with central memory and the central processor is a bank of ten peripheral processors (PP) which run independently of the central processor. PP 0 contains the PP monitor which coordinates and controls activities in the computer system. PP 9 contains a display and communication program which supports console operation. Twelve half-duplex data channels are controlled by the PPs and all channels may be active at once.

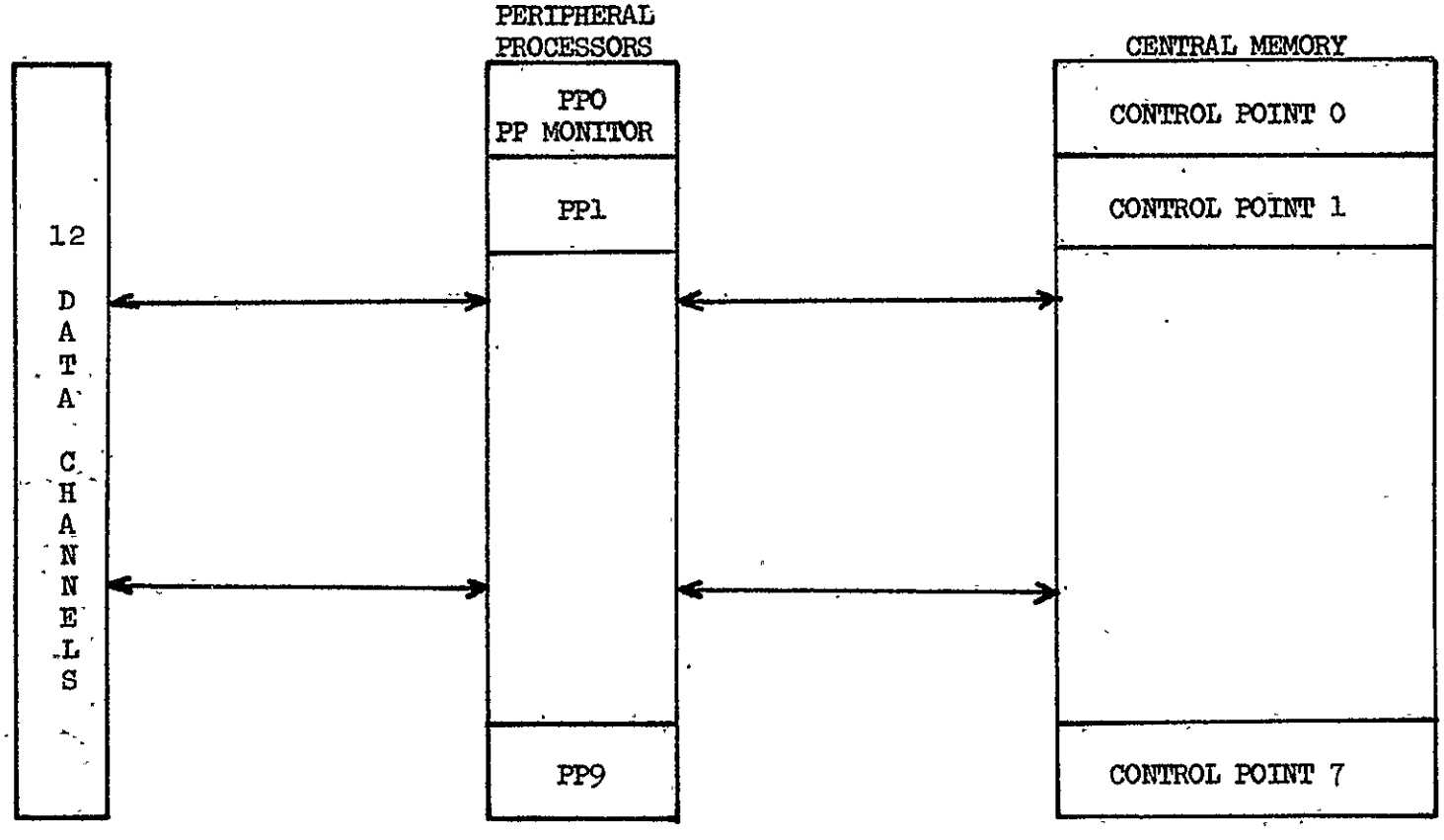


Figure 2. - CDC 6000 computer organization.

Real Time Control

Figure 3 shows the internal computer organization employed to support digital simulation. Illustrated are three digital simulations running concurrently. An additional central memory monitor is added to control point 0 to give faster monitor response to time critical demands for service. When real time simulations are running, a scheduling program (SKED) is always in control point 1. SKED allocates real time equipment and schedules the timing cycles for real time operations.

A dedicated PP with a dedicated data channel connected to ADCON (Analog-to-Digital Controller) and the real time clock comprise ADDIS. ADDIS places the outputs of ADCs and discrete inputs into the control points indicated by SKED. A second dedicated PP with a dedicated data channel connected to DACON (Digital-to-Analog Controller) and the real time clock comprise DADOS. DADOS removes data from the control points as indicated by SKED for output to the real time world.

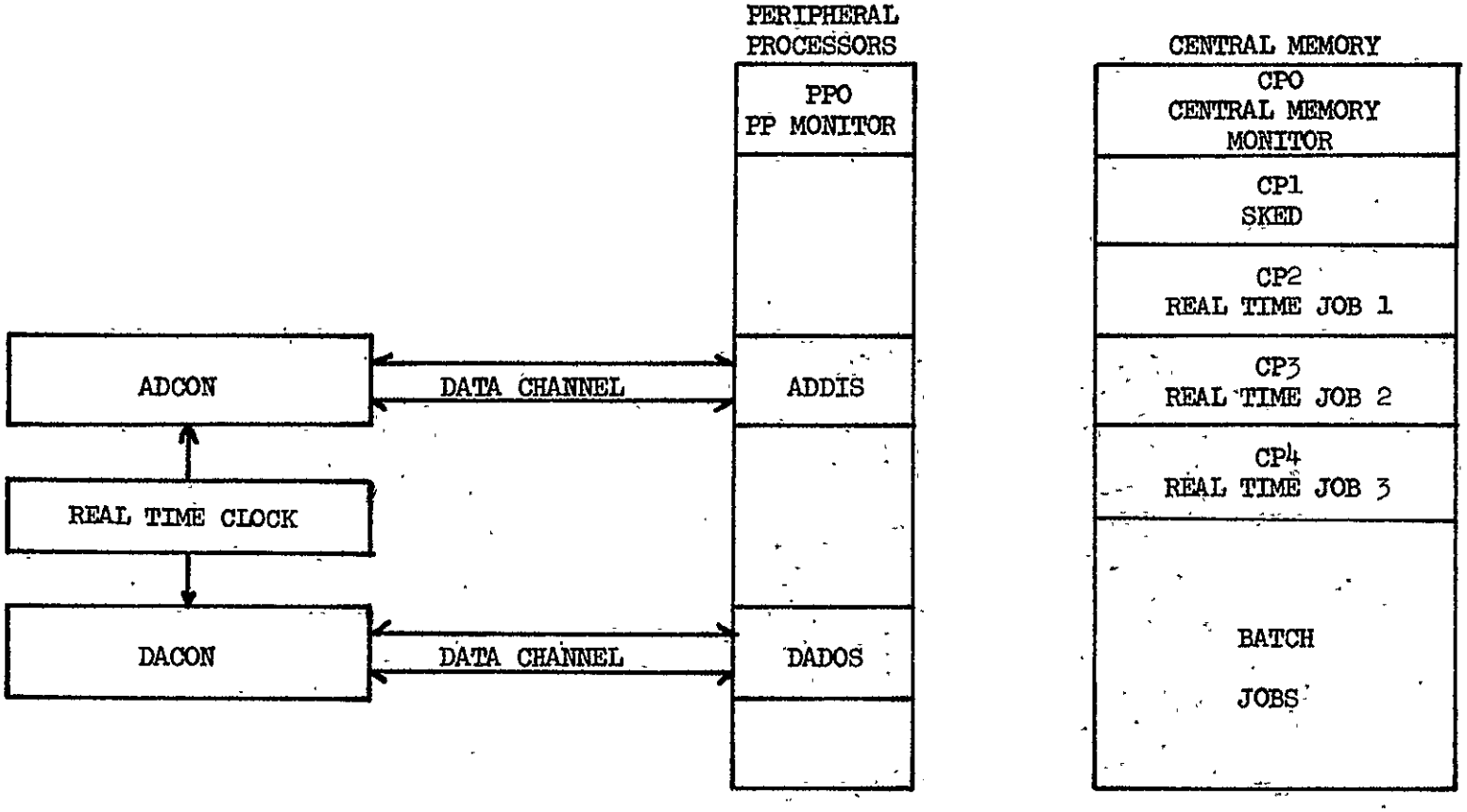


Figure 3. - Computer organization with real time digital simulation.

Figure 4 illustrates the allocation of central processor time with three simulation jobs running at different sampling rates. Note that a job does not necessarily compute the response of one frame continuously, but may be interrupted to allow another job to finish. The scheduling of compute and idle time is done by SKED according to timing information supplied by the simulation program.

This, then, illustrates the hardware and software environment in which real time digital simulations are performed and in this same environment the supervisor must perform its functions.

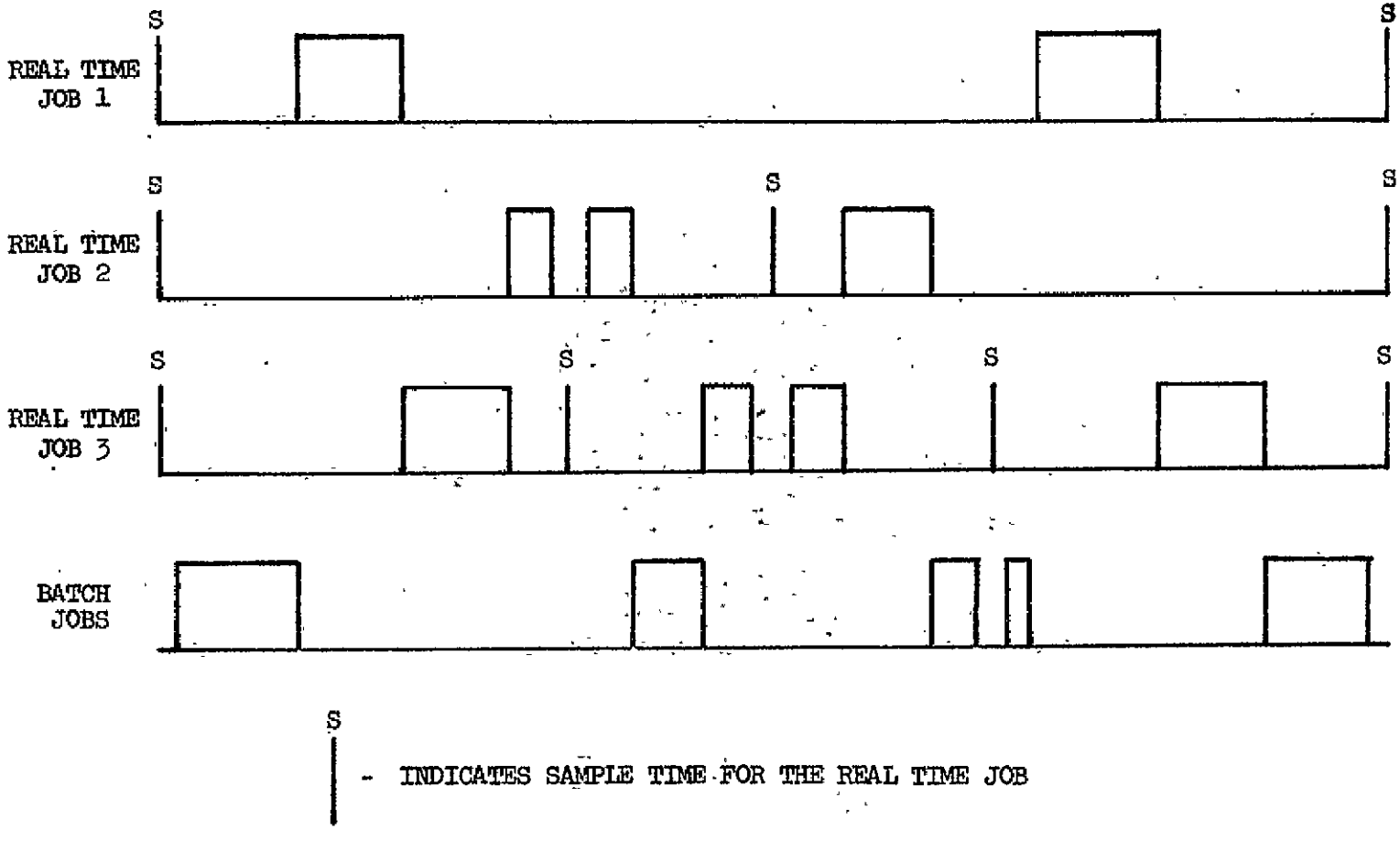


Figure 4. - Central processor time allocation with three real time simulation jobs.

CHAPTER III

REQUIREMENTS OF A REAL TIME DIGITAL SIMULATION SUPERVISOR

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

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

A. Real Time System Initialization

B. Real Time Timing Control

C. Real Time Central Memory Input/Output Control

D. Control after Lost Time Synchronization Interrupt

E. Mode Control

F. Real Time Data Storage and Retrieval

G. Print Output Control

H. Error Recovery and Diagnostics

I. Batch Job Compatibility

A. Real Time System Initialization

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

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

B. Real Time Timing Control

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

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

C. Real Time Central Memory Input/Output

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

D. Control after Lost Time Synchronization Interrupt

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

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

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

E. Mode Control

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

F. Real Time Data Storage and Retrieval

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

G. Print Output Control

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

H. Error Recovery and Diagnostics

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

I. Batch Job Compatability

Real time digital simulation is expensive in terms of machine resources and execution time. Therefore, it is undesirable to do

computations in real time when it is not necessary, such as during early coding checkout and purely analytic studies where real time input and control is not needed. An additional requirement of the supervisor is the capability of operating the simulation job as a real time job or as a normal batch job with minimum change necessary for the program.

CHAPTER IV

GENERAL OVERVIEW OF IMPLEMENTATION

The supervisor was designed and implemented with the following objectives: (1) to provide a high-level dialect of the Fortran language that would be used as a digital simulation language, (2) to make this dialect easy to use but still maintaining the complete capabilities of the real time digital simulation subsystem, and (3) to make the operation of the supervisor as efficient as possible commensurate with the time available for development. To this end, the following sections discuss some of the processes and techniques used to develop the supervisor. A general description of the structure of a real time simulation program is presented with emphasis on the impact of supervisor on programing. The implementation of mode controls and real time data recording is discussed with descriptions of special techniques developed. A discussion of techniques used in mode controls, time critical operations, and error recovery are presented to illustrate the varied programing procedures used to implement this real time digital simulation supervisor.

Figure 5 shows the supervisor in functional block diagram form. The supervisor is basically comprised of two sections, initialization and real time control. During initialization, the supervisor obtains certain information from the simulation program through declarative subroutine calls. From these calls, the supervisor initializes certain internal tables for use during real time

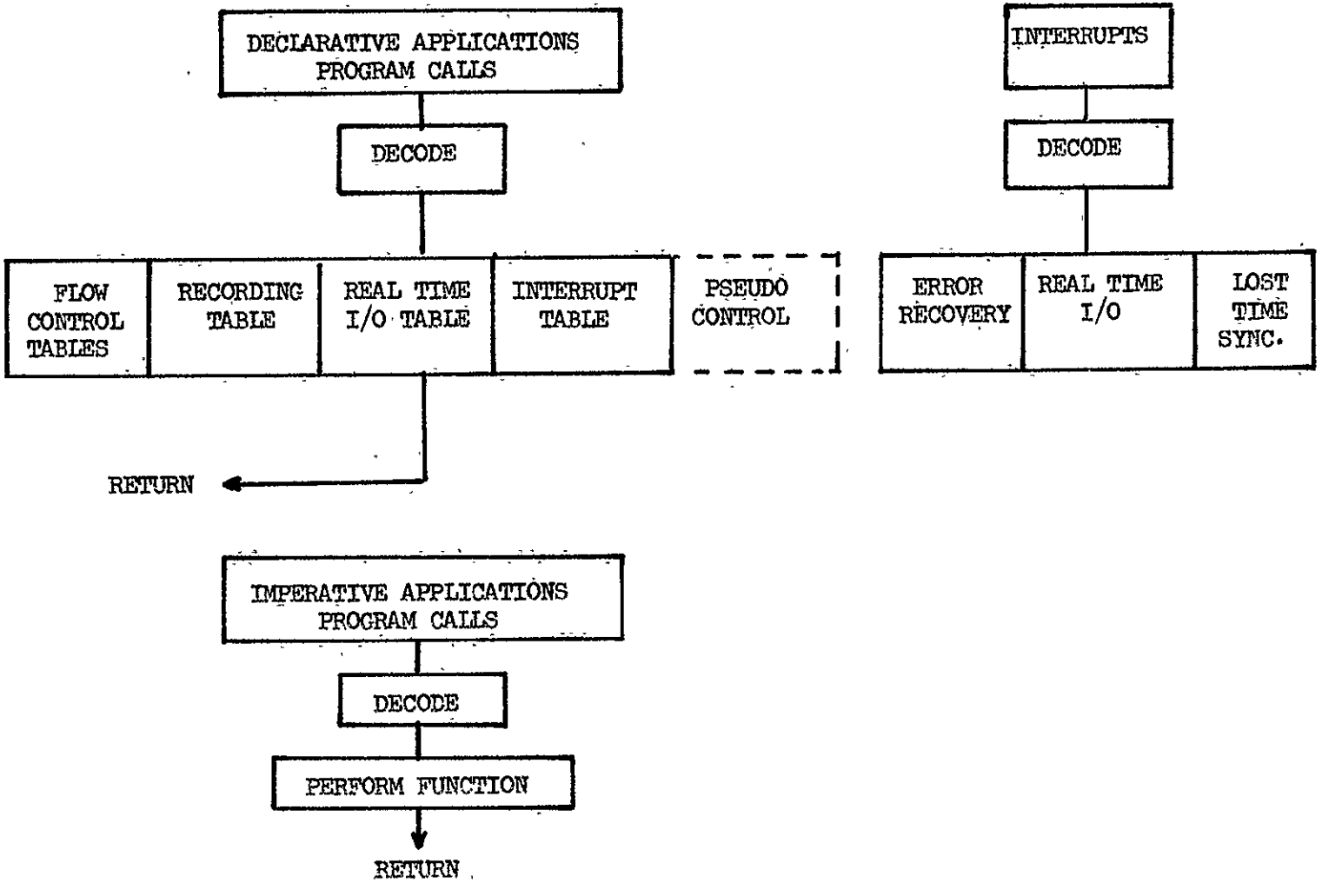


Figure 5. - Real time simulation supervisor block diagram.

execution. These tables include (1) addresses for the flow control of the simulation program, (2) variable table for recording of data, (3) real time I/O table for communication with the real time subsystem, (4) interrupt table for control of execution when lost time synchronization occurs, and (5) a set of pseudo control inputs for controlling the simulation if it is a normal batch job.

During real time operation, the supervisor must respond to imperative calls from the simulation program as well as respond to interrupts. The imperative calls from the simulation program cause the supervisor to process the requested function. There are three types of interrupts that affect the supervisor during real time operation. The real time I/O interrupt causes the supervisor to start execution after the sample time for the simulation occurs. The error interrupts may come from the hardware, if an arithmetic error occurs, or from the software, if any other errors occur. Lost time synchronization interrupt is issued by the monitor for action by the supervisor.

CHAPTER V

SIMULATION APPLICATION PROGRAM STRUCTURE

To meet the requirements set forth earlier, a real time digital simulation application program is written entirely in Fortran. All real time functions are provided by the supervisor and are activated by the use of the subroutine call. Figure 6 shows the block structure of a simulation program. The unique feature of the simulation program is that the various blocks are not linked--the interblock linkage is done by the supervisor under the control of the programer at his simulation control console. Except for the initialization section, there may be more than one each of the other blocks. The following paragraphs describe the function of each section.

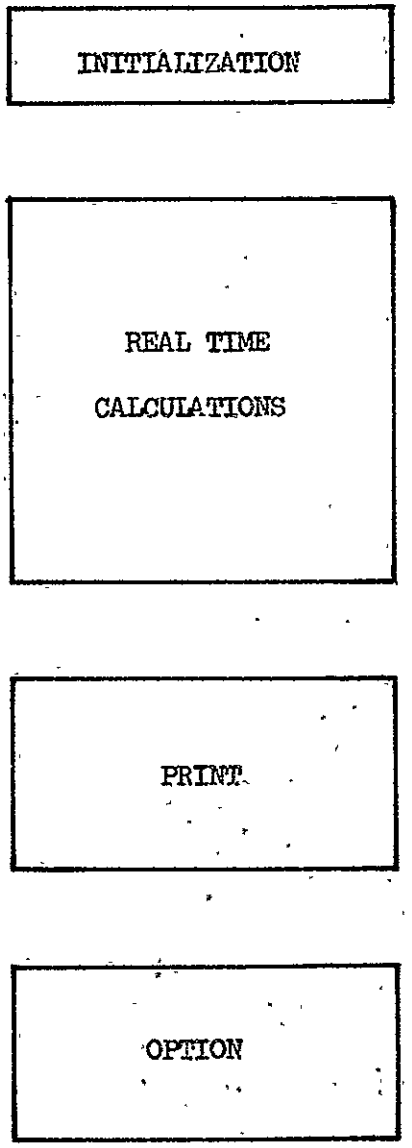


Figure 6. - Simulation applications program block structure.

Initialization Section

Figure 7 shows the structure of the initialization section of the simulation program. The normal non-executable statements are first, followed by initialization of the supervisor. The supervisor is initialized by a series of Fortran calls that pass the necessary addresses for communication and control of the simulation program. The simulation program then initializes program variables and parameters by reads, equalities, and/or subroutines.

At the end of the initialization section, the simulation program executes a CALL READY. This gives control to the supervisor and signals that the program is ready for real time operation. At this time, supervisor completes the initialization of certain tables and sets up error recovery code. The supervisor then indicates to the system that real time initialization is necessary. After the system has initialized, the supervisor requests real time operation. From this point forward, control of the program is done from the simulation control console.

Real Time Section

In this section, the equations of motion are solved in synchronization with time, so that real time responses to input signals are seen in the outside world. During this time, the supervisor is maintaining the time synchronization as well as performing functional tasks requested by the program and by the simulation programer at the simulation console. A more detailed description of the flow control done by

```
Fortran Non-Executables

    COMMON
    DATA
    EQUIVALENCE
    DIMENSION
    TYPE
      .
      .
      .

Supervisor Initialization

    CALL OPERATE  (        )
    CALL HOLD     (        )
    CALL RESET    (        )
    CALL INOUT    (        )
      .
      .
      .

Initialization of Program Variables

    READ ALPHA, BETA, . . .
    A = 0.0
    B = 4.5
      .
      .
      .

    CALL READY
```

Figure 7. - Simulation applications program initialization section.

supervisor is found in the next chapter.

Print Section

When the programer at the simulation console requests the print function, the supervisor returns control in non-real time to the simulation program in the Print Section. Here, the program may "play back" the data that was previously recorded in real time. Data may be analyzed, plotted on a plotting device, or printed at the control of the program. When the necessary processing of the Print Section is complete, control is returned to the supervisor for further real time processing.

Option Section

When the programer at the simulation console requests the option function, the supervisor returns control in non-real time to the simulation program. The simulation program then may make any analysis or initialization that is required. Perhaps the conditions for the next run might be read from a card deck or the conditions may be calculated from a complex set of error equations. There is no specific function of this section--it just gives the programer a section that can be executed in non-real time at the programer's request. At the end of the section, control is returned to the supervisor for further real time processing.

CHAPTER VI

MODE CONTROLS

Mode control for real time digital simulation is defined as the sensing of change of status of the mode control keys by the supervisor and the supervisor performing the requested flow control and/or function. Figure 8 illustrates the flow of a typical simulation indicating the flow control done in response to mode control.

In RESET mode, the initial conditions are applied to the state variables to initialize for a new run. In OPERATE mode, the complete dynamic equations are solved and the simulation is computing real time responses. In HOLD mode, the values of the state variables are held at their last calculated value, thus giving a static description of the simulation at the point the HOLD mode was entered. These three basic modes, RESET, HOLD, and OPERATE, form the primary mode control system that the supervisor must provide. In addition to these three primary modes, there is a set of secondary modes which are functions rather than flow controls. These functions include idle, erase real time file, process printing, clear output channels, continuous frame record and single frame record. These secondary modes are subordinate to the primary modes and there is a definite hierarchy of operation of the modes and this is employed in the supervisor.

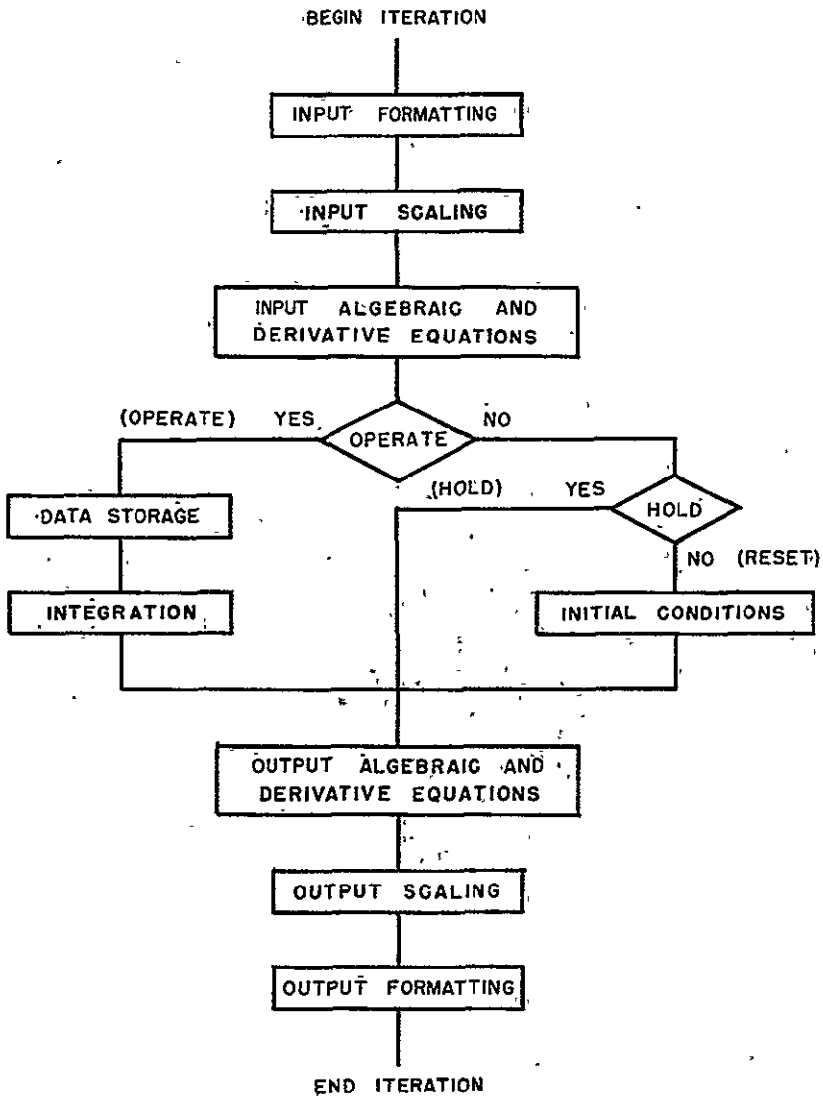


Figure 8. - Typical program flow for real time modes.

Block Programing

A simulation program may be easily broken up into logical program blocks as shown in Figure 9. These blocks are executed in different sequences for the different primary modes. It was required that the supervisor provide the necessary linking and delinking in order to perform the proper block execution sequence. The major technique developed for primary mode control was the block dispatching code generator.

Block Dispatching Code Generator

The block dispatching code generator is initialized by a series of calls that indicate the blocks that are to be executed for a given mode and the order in which they are to be executed. For example,

CALL OPERATE (10S, 15S, 20S, 25S, 10S, 15S)

indicates a sequence of blocks to be executed in OPERATE mode. (In CDC 6000 Fortran, an "S" following a number in a subroutine parameter list indicates that the number is a Fortran statement number and the location of that statement number is passed to the called subroutine.) In the example above, program blocks are to be executed in the following order: Block 10-15, Block 20-25, Block 10-15.
This illustrates the manner in which the starting and ending addresses of code blocks and their sequences are transmitted to the block dispatching code generator.

When a primary mode (OPERATE, HOLD, or RESET) is entered, the code generator must delink the program blocks from the previous mode

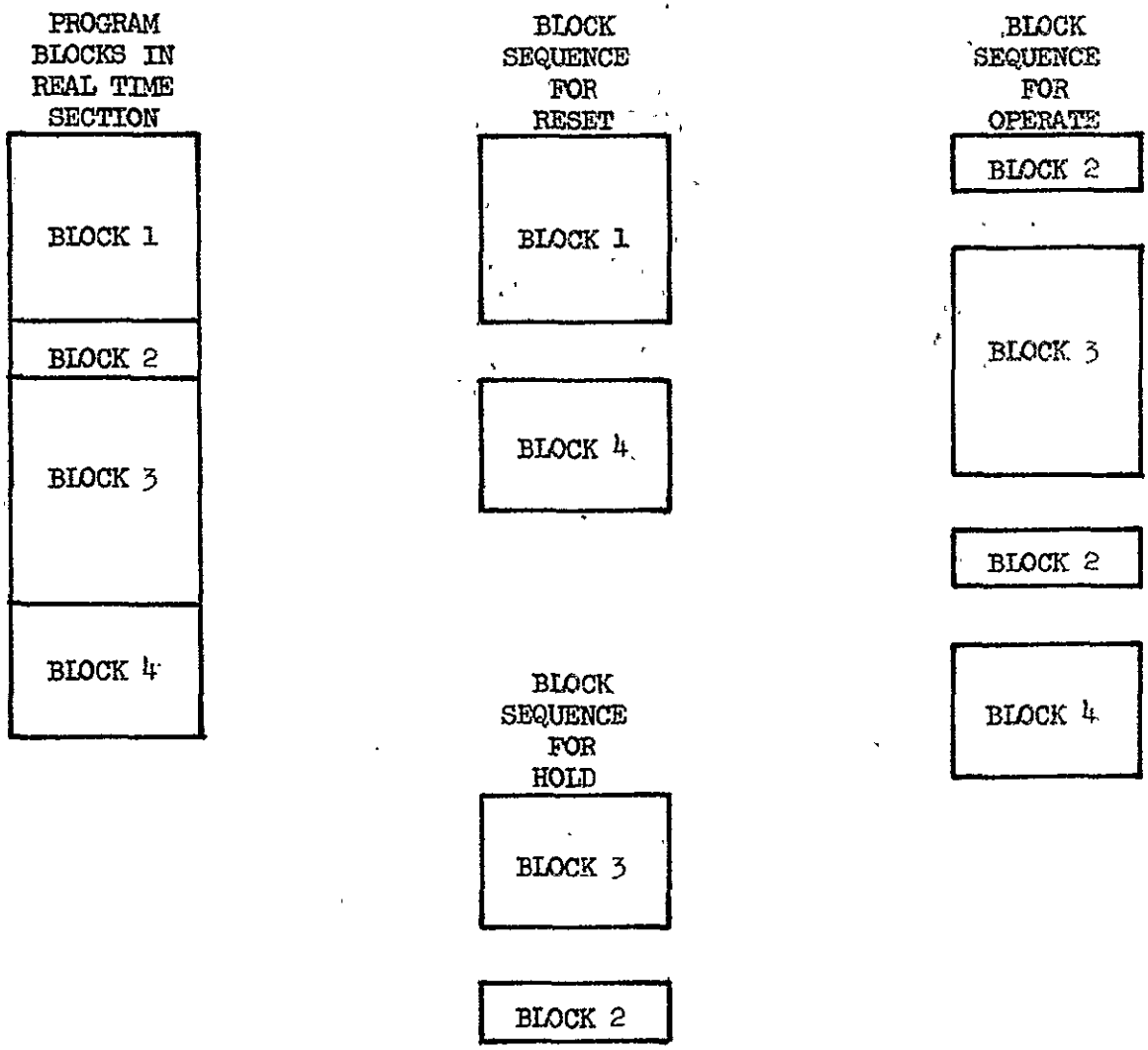
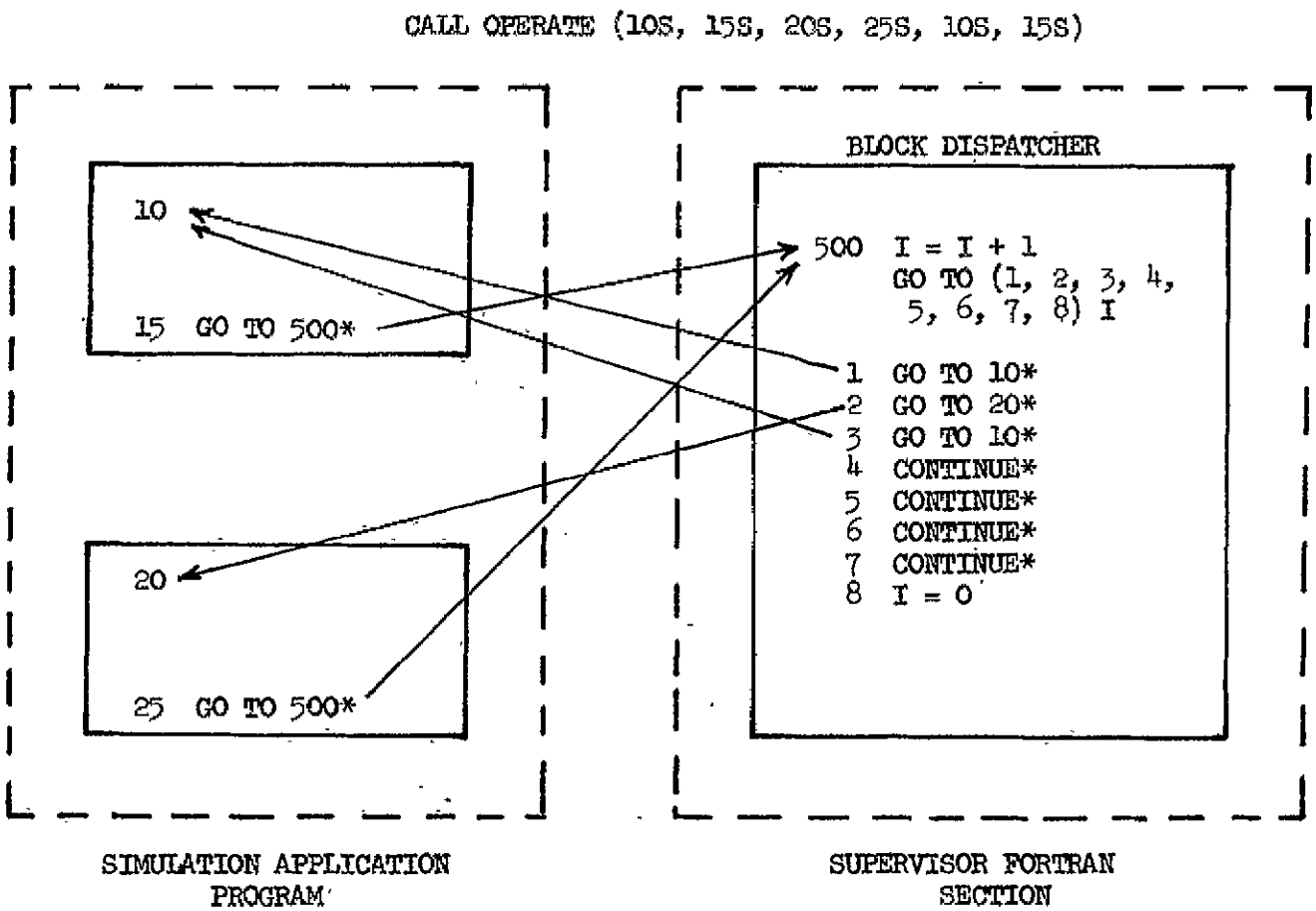


Figure 9. - Typical block sequences during real time operation.

and then must link the block together for the new mode. Figure 10 illustrates the manner in which the application program blocks and the Fortran portion of the supervisor are linked. The code generator must insert code at the end of each code block so that the block dispatcher in the supervisor can direct the block execution sequence. The code generator inserts code into the block dispatcher so that the proper sequence is executed. This results in a unified code structure that will rapidly execute in the requested order.

It should be noted that a maximum of seven distinct code blocks are allowed for each primary mode giving the programer a great deal of latitude in constructing a simulation application program. New initialization calls are allowed at any time so that block program flow can be modified "on the fly." Thus, by using this block dispatching code generator, the supervisor is able to provide a block link-delink capability that results in a unified code structure.



*INDICATES CODE INSERTED BY THE BLOCK DISPATCHING CODE GENERATOR

Figure 10. - Linkage diagram of block dispatcher.

CHAPTER VII

REAL TIME DATA RECORDING

One of the major capabilities of the supervisor is the storage of data generated during real time operation. Normally, Fortran input/output (READ, PRINT, WRITE) is performed by Fortran library subroutines that interface with the operating system. Unfortunately, each time a file action request is performed the central processor is relinquished until the file action request has been completed. Thus, if Fortran input/output is attempted during real time operation, time synchronization will be lost, compromising the validity of the simulation. Therefore, a specialized output procedure was required to perform this function.

The objectives in the implementation of this capability were threefold: (1) to provide the functional capability, (2) to implement the function in a manner that would give rapid execution (this function is done in real time), and (3) to provide a simple means of programing the function in the simulation application program.

To examine the implementation of the first objective, the method of central memory output in CDC 6000 computers must be examined. With each file that a program uses, there are two areas in core storage used, as shown in Figure 11. The file environment table (FET) contains the necessary information for communication to the external input/output (I/O) peripheral processor. The buffer area contains the data to be transmitted in the I/O operation. The first word of the

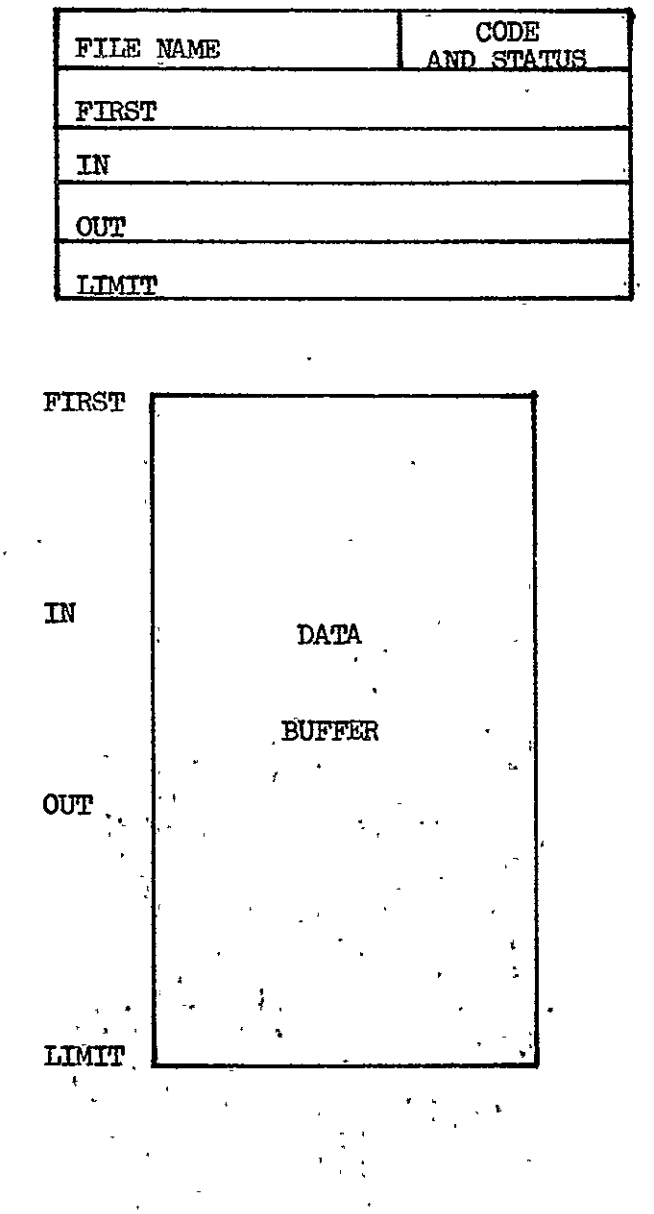


Figure 11. - Typical file environment table and data buffer.

FET contains the file name and the code and status of the file. File action requests are placed in the code and status position and status is returned by the I/O processor. FIRST and LIMIT indicate the beginning and ending addresses of the buffer and IN and OUT indicate where data can be put in and taken out of the buffer.

To implement the storage of real time data, a Fortran callable subroutine was written that creates an FET and a data buffer in the Fortran section of the supervisor. This enables the Fortran section to directly make file action requests in the file name portion of the FET, to interrogate status of the I/O operation, and to manipulate the data buffer and the buffer pointers. This satisfies the second objective, in that simple Fortran replacement statements and simple conditional transfer statements can be used to contol the I/O.

In addition to the above, the data buffer was partitioned into two segments that operate as seperate buffers by the control of FIRST and LIMIT. This allows the supervisor to fill one buffer while the I/O system empties the other buffer, enabling the supervisor to continue the storage process while output to the physical file is taking place.

There is an inherent problem in this scheme. If the supervisor fills the buffers too fast or if the output system response time becomes large, the supervisor may need to fill a buffer before the output system has emptied it. If this occurs, the supervisor must discard data until the output is complete. The supervisor stores two extra code words each time a frame is recorded and this allows

missing data to be detected when the file is "played back." In order to provide faster response to real time data recording demands, the operating system has provided a means for requesting a high priority read or write. The use of this capability is limited to the supervisor and the supervisor uses this function only when necessary, so that the response of other real time jobs is not unduly affected.

Program Interface

The simulation program interface has been designed so that the program, through a series of subroutine calls, specifies a list of variables to be recorded. This list is normally specified during program initialization but may be respecified during any non real time processing. The supervisor sets up an address table to direct the storage of data. This allows the recording to be made without the passing of an extensive parameter list or forcing the simulation program to use a fixed array for recorded data variables. This program interface accomplishes the third objective; a simple means of programing has been provided.

CHAPTER VIII

TIME CRITICAL OPERATIONS

The central processor time used by the supervisor during real time operation is strictly overhead which restricts the time that is available for the simulation program to calculate real time results. An objective of the supervisor was to shorten this overhead time as much as possible. The following sections describe some of the techniques and methods used to shorten execution during time critical operations.

Discrete Handling

Discrete inputs and outputs are placed in and taken out of central memory in packed form; that is, sixty discrete channels per word. The first version of the supervisor unpacked inputs and packed outputs so that discretes were available to the simulation program and the supervisor as logical variables, one discrete channel per word. This unpack/pack operation consumes a considerable amount of time each frame. A technique was developed to provide discrete manipulation in packed form within the supervisor as well as the simulation program.

This technique for handling discretes was developed to allow in-line Fortran testing which is fast and easy to use. As shown in Figure 12, two arrays are set up in COMMON/MASKS/. The TMASK array has only one bit set in each word, the particular bit being set depending on the array element. The FMASK array has all but one bit

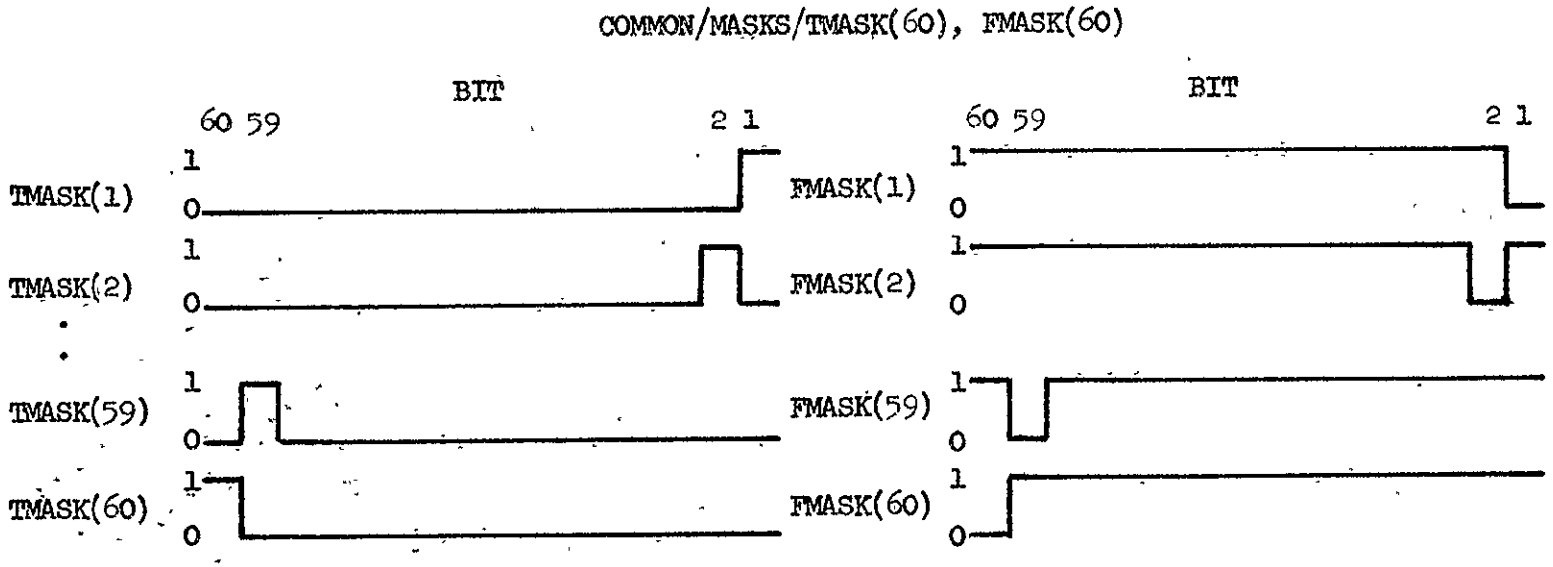
set in each word and again the bit that is not set is dependent on the array element. The use of these masks for discrete manipulation is illustrated in the figure. By using other masks, a simulation program can set or reset a number of discretes with one Fortran statement.

Fortran Usage

Special usages were developed to speed up the Fortran section and to reduce the overall core storage requirements. Compilations were made and the object code was analyzed to find the most efficient Fortran coding techniques for the real time sections. Some of the techniques found and used are described below.

Extensive use is made of the "GO TO NAME" statement where NAME is variable name normally defined by an assign statement. However, CDC 6000 Fortran will allow the variable to be defined in any manner. By communication through parameter lists and COMMON, these variables can be used by any routine. This allows this type of GO TO statement to interconnect various routines by direct transfers rather than indirect subroutine linkage. Arithmetic operations are allowed since the content of the variable is simply an address. This allows the supervisor to compute addresses where necessary without having to use an assembly language routine.

It was found that some forms of IF statements are more efficient than others and it was also found that single replacement was more efficient than multiple replacement. (Multiple replacement: A = B = C = D = 0.0)




```
TESTING DISCRETES
   IF (IDIS(1).AND.TMASK(30)) GO TO 100
     TESTS THIRTIETH BIT IN FIRST WORD OF DISCRETE INPUT ARRAY

SETTING DISCRETES
   ODIS(2) = ODIS(2).OR.TMASK(43)
      SETS BIT 43 TRUE IN SECOND WORD OF DISCRETE OUTPUT ARRAY
   ODIS(1) = ODIS(1).AND.FMASK(22)
      SETS BIT 22 FALSE IN FIRST WORD OF DISCRETE OUTPUT ARRAY
```

Figure 12. - Use of masks for packed discrete manipulation.

The techniques above result in faster executing code. In investigating subroutine linkage, a CALL SUB (0,0,0,0) would cause references to four different words of zero which is redundant and wasteful of central memory. This same call would be implemented as CALL SUB (IZ,IZ,IZ,IZ) where IZ is a data zero. Wherever practical, in-line Fortran code is used instead of using subroutines to conserve execution time. Such special techniques give small gains in execution time and core storage, but when added together provide significant overall gain.

Subroutine Linkage

The supervisor consists of two lengthy subroutines, one in Fortran and one in assembly language. The investigation of object code from Fortran compilations showed that the use of a parameter list and multiple entry points was inefficient, both in terms of execution time and memory usage. Therefore, all alternate entry points and the parameter list were removed from the Fortran subroutine. Entry points were added to the assembly language routine and parameters were then passed to the Fortran routine through COMMON. About 1000_8 locations were saved in a Fortran routine whose executable code was about 4500_8 locations. The resulting savings in memory and execution time were significant.

Deferred Execution

Time remaining is defined as the scheduled compute time minus the actual compute time. The value of time remaining can vary over a wide range depending upon demand of the application program. It is required

that supervisor make effective use of time remaining so that the maximum actual compute time is not increased by supervisory functions. In order to accomplish this, a technique for deferred execution was developed. During frames when the simulation program has heavy demands for computation, certain supervisor flags are set and execution of some supervisor functions are not done. When the computation demand of the simulation program becomes light, then these functions are executed.

In the same view, the supervisor performs any lengthy operations in non-real time that do not require real time operation. Care is taken so response to programer inputs is not unduly affected.

These various techniques have been used to make the supervisor more efficient in the use of central memory and the central processor. This allows the simulation program to have a bigger slice of time and allows either more or longer jobs to reside in central memory.

CHAPTER IX

ERROR RECOVERY AND DIAGNOSTICS

Recovery from error is one of the important features of a simulation supervisor. In a normal batch processing mode, when an execution error is detected, the program being executed is aborted. This mode of operation is undesirable for real time digital simulation since this can lead to loss of expensive data and machine execution time as well as lost manhours. One of the objectives of the supervisor was to capture certain classes of execution errors in order to enable the program to fix the simulation and to continue real time operation. If the program could not be fixed on-line, then at least the current data could be recovered for later analysis. Areas to be discussed are: (1) recovery from lost time synchronization interrupt, (2) non-real time execution attempted in real time, (3) Fortran execution errors, (4) arithmetic errors, and (5) supervisor dialect errors.

Lost Time Synchronization Interrupt

When the simulation program attempts to exceed the time allotted for a real time response to be calculated, the monitor issues a lost time synchronization interrupt. At this point, the central processor is taken away from the simulation program and given to another job as shown in Figure 13. On the next frame, real time control of the simulation job is returned to the supervisor as specified by the supervisor during program initialization. The supervisor then uses

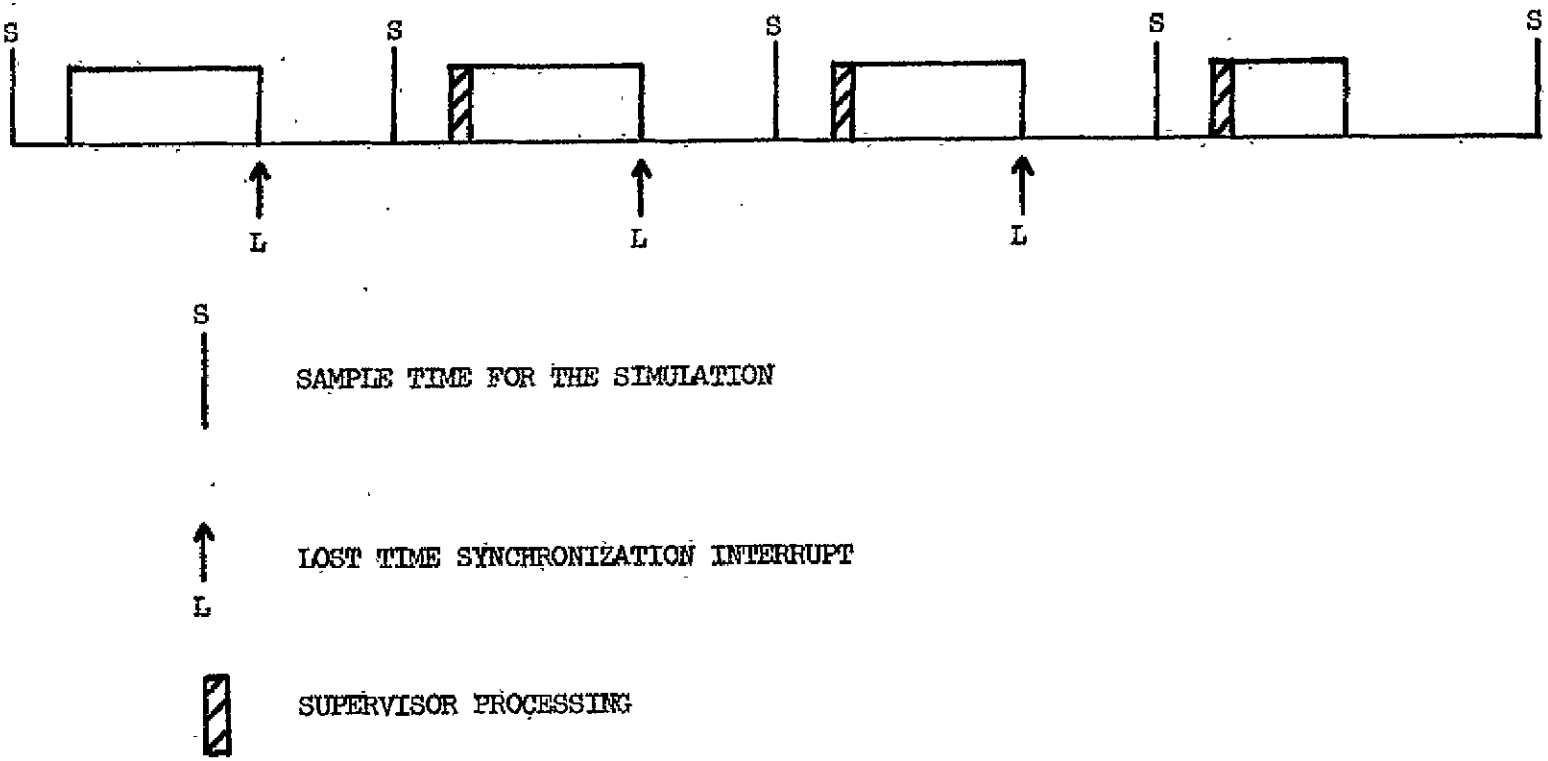


Figure 13. - Supervisor special execution at lost time synchronization interrupt.

a special routine to save the operating registers so that the program may be restarted at the exact point that interrupt occurred.

The supervisor must now determine what the operation of the simulation program will be. There are several options available to the programer.

A. The supervisor will stop execution, print the program address at which interrupt occurred and wait for programer action at the control console. Programer action may be
   (1) Selection of a different mode
   (2) Advance frame command--one more frame of computation will be allowed at which time the program will again stop.
   (3) Any functional request (print, idle, erase, etc.) followed by (1) or (2)

B. The supervisor will restart the program in real time and allow a specified number of cycles of lost time operation to occur before stopping the program.

C. The supervisor will restart the program in real time and allow it to finish the current iteration only, stop the program and (1) allow any option under A, above, or (2) transfer to a specified point in the simulation program.

D. The supervisor will restart the simulation program indicating that lost time is occurring and continue real time operation.

E. The supervisor will stop the execution, print the program address at which interrupt occurred and return control to a

specified point in the simulation program.

In the event that the simulation program specifies D as the lost time option to be performed, supervisor performs a short register save-restore--only enough registers are saved to test a flag and return to real time.

Figure 13 illustrates operation of a program using option D where lost time synchronization interrupts have occurred three times during one real time solution. The program has taken four frames to calculate one response so that real time synchronization has been delayed and the solution can no longer be guaranteed.

It should also be emphasized that when supervisor returns control to the program, the program may request real time operation at the point at which interrupt occurred.

Real Time Errors

There are certain functions such as Fortran input/output that cannot be attempted in real time with the guarantee that time synchronization will be maintained. Therefore, it was required that supervisor detect this type of error and provide the necessary recovery capability. Code modification techniques were used to insure detection of these errors. Code is inserted at the entry point of each potential error generating routine. When the routine is called, control is transferred to the supervisor. The supervisor, then, is able to check for real time operation each time one of these routines is called and is able to capture the error before it occurs.

Fortran Execution Errors

A central subroutine called SYSTEM is called by any system Fortran subroutine that detects an error (like attempting to take the square root of a negative number). The subroutine issues an error message to the print file and aborts the program if the error is fatal. It is required, therefore, that supervisor detect such errors and insure that the program does not abort.

The supervisor performs three modifications to SYSTEM: (1) the error table internal to SYSTEM is modified so that all errors are processed as non-fatal so that no aborts occur, (2) code at the entry point is modified so that control is returned to the supervisor rather than the error detecting routine, and (3) the output section of SYSTEM is modified so that diagnostic messages are relayed to the typewriter rather than the standard line printer.

The Fortran functions of STOP, END, EXIT are serviced by entry points in SYSTEM. By modification of code, these entry points are "locked out" and entry is made to the supervisor when one of these functions is attempted. Program termination is initiated by the supervisor only and requires a positive request from the program control console.

Arithmetic Errors

An important function of the supervisor is the recovery from arithmetic errors. Arithmetic errors occur when the hardware detects an operation that it cannot perform, that is, address range error,

division by zero, etc. When an arithmetic error is detected, transfer is made to the operating system for abort. However, a special routine was put in the operating system that will return control to supervisor when an arithmetic error is detected. The supervisor then prints out the contents of the operating registers and appropriate memory locations for analysis by the programer. The programer still has full control of his simulation program after an arithmetic error has occurred.

Supervisor Dialect Errors

Since subroutine calls to the supervisor form a simulation language, there can be sequences of calls that would comprise "syntax" errors. These errors, as well as illegal parameter errors, are detected by the supervisor. If an error occurs during initialization, the program is aborted, since real time execution had not been initiated. If, however, any of these errors occur after real time operation has begun, the supervisor will capture control and give the programer a trace back of the error causing procedure.

CHAPTER X

CONCLUDING REMARKS

The objective of this work has been to develop a supervisory system to support "man in the loop" real time digital flight simulations. To this end, a set of routines called "the supervisor" was developed that enable a simulation applications programer to program very large simulations entirely in Fortran without sacrificing the flexibility, control, and interactive features required to perform digital simulation. A basic simulation dialect consisting of simple Fortran subroutine calls has been developed and will provide a base for later extensions of the supervisor.

Continued research and development work will be undertaken as the need arises. One of the areas of future development is the capability of coordinated real time inputs from computer-based storage systems. This would allow coordinated inputting of information such as recorded wind gusts and other predetermined measured variables. This and continued development work will keep the supervisor abreast of the needs of the real time digital simulation community.

APPENDIX

The following discussion is an abridged version of Volume I, Section 2.1, of the Langley Research Center Programing Manual. This section was written by Mr. Maurice K. Morin, Head, Programing Techniques Branch, and edited by the thesis author. The hardware and software resources of the Langley Research Center Computer Complex are described as a reference for the paper.

CHAPTER I

HARDWARE CONFIGURATION

The system configuration is represented schematically in Figure A1. It can be divided into three main subdivisions: the communication and control section, consisting of a large shared core memory; the processing section consisting of four independent computers; and the input/output section containing all peripheral equipment organized into a shared peripheral pool.

Communication Section

The shaded block at the center of Figure A1 represents the shared core memory. It will consist of one million 60-bit words of core storage equally accessible to all computers in the processing section. This component will serve as the focal point of the configuration by providing for all inter-system communication. It is scheduled for delivery in 1970. The remaining elements of the hardware configuration are currently installed and are in productive use.

Processing Section

The processing section, surrounding the shared core memory on Figure A1 consists of four independent computer systems. The first, a 6400A, has a unified central processor capable of executing .5 million operations per second (MOPS), which is approximately 1.4 times the speed of our previously installed IBM 7094II. A unified

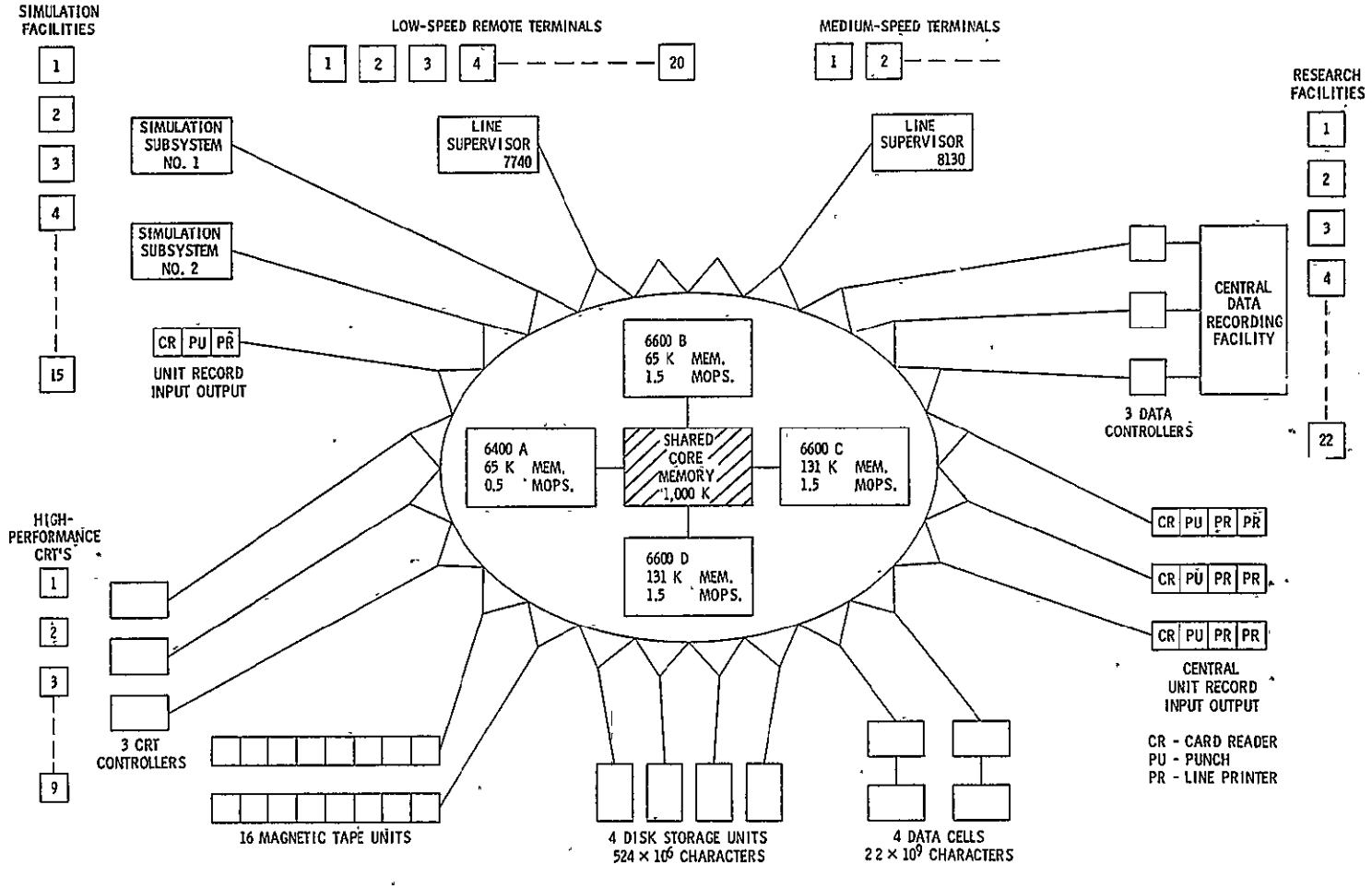


Figure A1. - Langley Research Center computer complex.

central processor means that instructions are executed one at a time in the sequence dictated by the algorithm being solved. Its memory consists of 65,000 60-bit words. The memory cycle time is 1 microsecond; however, through the use of inter-leaving, an effective access time of 100 nanoseconds can be achieved.

The next computer, the 6600B, has a multiple functional unit central processor capable of executing 1.5 million operations per second, which is approximately five times faster than our previously installed IBM 7094II. The multiple functional unit central processor provides the ability to execute numerous instructions simultaneously without altering the intent of the algorithm being solved. This feature is the main difference between the 6400 and 6600 series of computers. The 6600B, 6600C, and 6600D each have a central memory capacity of 131,000 words.

The A, B, C, and D designators have been established by local convention, and do not represent manufacturer's models. They are used simply to distinguish among the machines.

Each of these 6000 series computers has ten peripheral processors, each of which, in effect, is a stored program computer with a processor and 4096 12-bit words of memory. These ten peripheral processors, together with twelve half duplex channels, serve as the link for input/output communication and control, and overall system operation control for each 6000 series computer.

I/O Section

The third subdivision of the computer configuration, surrounding the central processors in Figure A1, is the shared peripheral pool. Communication between the central computers and peripheral equipment is accomplished through a battery of multi-access switches, which allow the computers to communicate with any peripheral device. Thus, the peripheral equipment is shared among the computers. These switches operate under program control. A central display with manual lockout controls is also available to accommodate system partitioning. With this feature, a computer and selected peripheral equipment can be physically partitioned off from the complex and used for development work or maintenance without endangering the continued reliable use of the remaining portion of the computer complex.

Shared Peripheral Pool Subsystems

The functional capability of the equipment comprising the shared peripheral pool will be described beginning with the equipment shown at the bottom right-hand corner of Figure A1 and proceeding clockwise around the chart.

Batch processing subsystem

The first major subsystem of the shared peripheral pool is the batch processing subsystem. It consists of a battery of conventional types of unit record equipment, located in the computer complex. The major elements comprising this subsystem and their performance are listed in Figure A2. Of particular interest here is the

decentralization of operation of this equipment. This is accomplished through a message switching and response polling software system, using six low performance CRT/keyboard inquiry type devices, each of which is associated with small groupings of peripheral equipment. For example, one CRT/keyboard is associated with a reader, punch, and printer. Its function is to provide the operator of this equipment with all the information and control necessary to effectively operate these particular pieces of equipment. This concept has been implemented throughout the peripheral pool, and has proved extremely effective in decentralizing operations into smaller self-manageable groups. As a matter of fact, the major portion of our peripheral equipment is located in different rooms from the main systems' operation consoles.

Auxiliary storage subsystem

The auxiliary storage subsystem consists of three main elements: data cell drives for permanent storage of programs; disk storage drives for high speed transient storage; and magnetic tape drives for data storage and communication with off-line devices. Figure A3 lists the major characteristics of the equipment in this subsystem The total capacity of the four disk drives is 524 million 6-bit characters. The average access or head positioning time is 60 milliseconds; the average rotational delay is 25 milliseconds. These performance features, coupled with the fact that as many as 128 files of 32,000 characters each are available at each access position

4 CARD READERS	1000 CARDS/MIN.
3 CARD PUNCHES	250 CARDS/MIN.
6 LINE PRINTERS	1000 LINES/MIN.
1 PAPER TAPE PUNCH	150 CHAR./SECOND
6 CRT/KEYBOARD (PERIPHERAL CONTROL STATIONS)	12 INCH 1000 CHAR. SCREEN

EXAMPLE OF DECENTRALIZED OPERATION

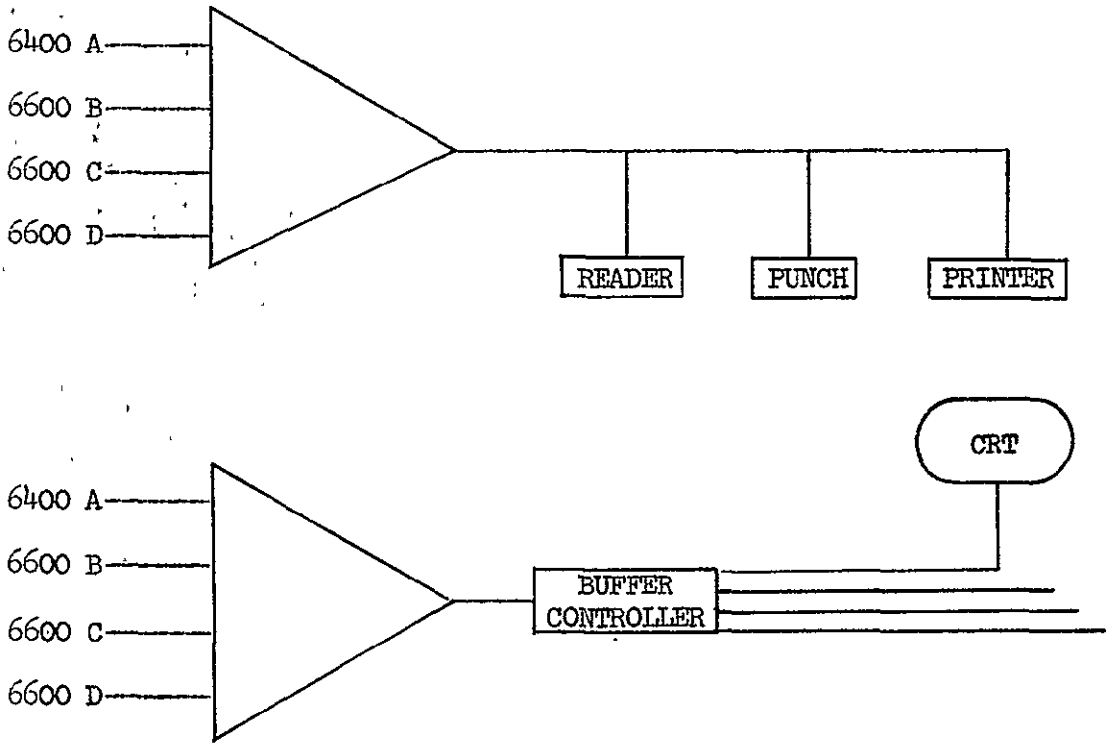


Figure A2. - Batch processing subsystem and decentralized operation.

AUXILIARY STORAGE SUBSYSTEM

4 DISK STORAGE DRIVES (TRANSIENT STORAGE)

524 M CHARACTERS

60 MS AVG. ACCESS TIME

25 MS AVG. ROTATIONAL DELAY

128 FILES AVAILABLE AT 1 ACCESS POSITION

(FILE = 32,000 CH)

4 DATA CELL DRIVES (PERMANENT STORAGE)

2.2 BILLION CHARACTERS

40 REMOVABLE WEDGES

DATA MANAGEMENT SOFTWARE FOR PROGRAM AND STORAGE STASH, FETCH AND MODIFY, REPLACE

16 MAGNETIC TAPE DRIVES

MAX. DATA TRANSFER SPEED 120,000 CHAR./SEC. BURST RATE AT 800 BPI (BITS/INCH)

TRANSPORT SPEED 150 INCHES/SECOND

LONGITUDINAL DENSITY 200,556 AND 800 BPI

DATA CODE 7 TRACK - EVEN PARITY BINARY CODED DECIMAL
OR 7 TRACK - ODD PARITY BINARY

Figure A3. - Auxiliary storage subsystem.

of each disk drive, has greatly improved system throughput in comparison with previously available disk systems.

The software controlling disk operation utilizes a centralized stack processor. In processing disk requests, the stack processor first selects all requests associated with real-time applications; all other requests are then selected, primarily, on the basis of minimizing head movement. The system provides for both sequential and random processing for files stored on disk.

The data cell drives provide for a total on-line storage volume of 2.2 billion 6-bit characters of alphanumeric or binary information. This information volume is equally divided over forty removable wedges, ten wedges per data cell drive. It is estimated that this storage subsystem will provide adequate storage for 1000 application programs on-line to the computer systems.

A data management software subsystem rigidly controls the cataloging, storage, and retrieval of all programs stored in data cells. Usage statistics such as date of entry, number of times acc ssed, date of last use, etc. and a descriptive label for each file are maintained by the system. The unit of information on the data cells is a file consisting of a source program and its compiled binary object program. Because of the need for extreme care in maintaining the continued reliability and integrity of the entire permanent storage information base, the user is not permitted to gain direct access to the content of his files in the data cells during his program's execution. Instead, upon command, an entire file will be

accessed and transcribed from data cell to disk in one continuous operation. Once the file is transcribed to disk, the user can manipulate it in any way he chooses. The converse is also true. Before a new file will be cataloged into permanent storage, it must exist in its entirety on disk. As an example, assume a user has a program cataloged in the data cells. With a simple set of control cards, he can FETCH his program, MODIFY selected statements, compile and execute, and, if desired, REPLACE the original program with the new modified version.

Real-time simulation subsystem

One of the major requirements of the LRC Computer Complex is the ability to perform multiple real-time digital flight simulations in a single 6000 computer (see Figure A4). To perform "man in the loop" digital simulation requires that the computer operate as part of a closed loop, time critical system where precise problem solution rates must be guaranteed in order to maintain the integrity of the simulation. These requirements have necessitated the design and implementation of a hardware and software subsystem with unusually high performance characteristics. Figure A4 lists the major elements that comprise the hardware subsystems.

A real-time clock, accurate to 100 nanoseconds provides timing control for both the input/output subsystems and computing algorithm. The input subsystem consists of eighty analog channels and 960 discretes for external event sensing. Each analog channel converts to

REAL-TIME SIMULATION SUBSYSTEMS

EACH OF THE TWO SIMULATION SUBSYSTEMS CONTAIN THE FOLLOWING

REAL-TIME CLOCK

INPUT SUBSYSTEM

80 A/D CHANNELS	15 BIT 1.25 US/CHANNEL
960 DISCRETES	

OUTPUT SUBSYSTEM

192 D/A CHANNELS	15 BIT 1.25 US/CHANNEL
960 DISCRETES	

CRT SUBSYSTEM

6 CRT CONSOLES	HIGH PERFORMANCE
1 HARD COPY RECORDER	

Figure A4. - Real time simulation subsystems.

fifteen bits, fourteen data bits plus sign, and through a series of multiplexors can achieve a conversion rate of 1.25 microseconds per channel. The output subsystem contains 192 digital output channels and 960 discretes for external event control. Each output channel has fifteen bits of precision and operates at an effective rate of 1.25 microseconds per digital-to-analog conversion.

In order to maintain sufficient communication and control during the progress of a simulation, a high performance CRT (cathode ray tube), with keyboard, function switches, etc. and a central hardcopy recorder will be available to the test conductor. The CRT will be used to provide a dynamic display of selectable key parameters in graphic and/or tabular form for monitoring purposes. It will also provide real-time control features which will allow the test conductor to START, STOP, or HOLD the simulation, perform detailed analysis of historical information collected during the course of the simulation and resume or reinitialize the simulation at various selectable times.

Development activity is currently underway in ACD which will provide a more general purpose use of CRT's. Our objectives are to provide, through software development, the necessary communication, control, and graphic tools to support interactive problem solving and analysis via the CRT.

Remote terminal subsystem

Remote batch processing services are provided through two independent remote terminal systems (see Figure A5). The first, a low-

speed system, supports twenty terminals, each terminal consisting of a fifteen character-per-second card reader and typewriter printer, operating over voice grade telephone lines. The IBM 7740 teleprocessing computer provides for the collection of jobs from the remote terminals and/or dissemination of results back to the terminals. All job processing is performed in the 6000 computer, to which the terminal system is connected. We are currently processing approximately 120 jobs a day through these terminals.

A medium-speed remote terminal system called EXPORT/IMPORT, consisting of four terminals, comprises the second part of the remote terminal subsystem. Each terminal consists of a 300 card-per-minute reader and 300 line-per-minute printer, operating over voice grade telephone lines. Each remote terminal has a stored program buffer controller, which communicates with the central computer for the transmission of a job or the receipt of computer results, in blocks of approximately 600 characters per transmission.

Jobs submitted from remote terminals process in the central computer in the same fashion as jobs submitted at the center. In order to more effectively expedite output, a route feature is available to the remote terminal user. This capability, invoked through the use of a control card, allows the user to selectively direct output files by name to his remote terminal or to high-speed printers at the computing center.

REMOTE TERMINAL SUBSYSTEM

LOW SPEED

IBM 7740 TELEPROCESSING COMPUTER

20 - IBM 1050 TERMINALS

CARD READER	15 CH/SEC.
TYPEWRITER PRINTER	15 CH/SEC.

MEDIUM SPEED

CDC EXPORT/IMPORT SYSTEM

4 REMOTE TERMINALS

CARD READER	300 CARDS/MIN.
LINE PRINTER	300 LINES/MIN.

Figure A5. - Remote terminal subsystem.

On-line subsystem

Input paths for on-line data reduction applications are provided through direct interfaces between the computers and five digital recorders in our Central Data Recording System (see Figure A6). The central recorders are connected through patchboard switching and underground cables to twenty-two test sites at the Langley Research Center. In normal operation, the digitized recorder outputs from 100 analog inputs are recorded on magnetic tape. When on-line services are required, data from as many as five test sites, in addition to being recorded on magnetic tapes, will be transmitted from the central recorders directly into a 6000 computer, where they will be decommutated, labeled as to origin and time, and stored into the disk subsystem. At the same time, operating in a multi-program environment at a lower priority level than the storage and decommutation program, reduction programs will perform whatever calculations are required upon the data and will route the output back to the remote research facility using the remote terminal subsystem. This subsystem thus far has supported on-line computation requirements for three research facilities at Langley Research Center.

ON-LINE SUBSYSTEM

INPUTS FROM:

5 RECORDING SYSTEM INTERFACES 15KC EACH

OUTPUTS USE:

LOW SPEED REMOTE TERMINALS

REAL-TIME DATA DECOM AND STORAGE
QUASI REAL-TIME DATA REDUCTION

Figure A6. - On-line subsystem.

CHAPTER II

OPERATING SYSTEM SOFTWARE

General Description

Figure A7 provides an overview of the operating system software which supports the use and operation of the Langley Research Center Digital Computer Complex. It is called the LRC SCOPE operating system, SCOPE being an acronym for Simultaneous Computing and Operation of Peripheral Equipment.

The Application Programer's Tools, indicated at the top of Figure A7, represent the software elements which should be most familiar to the user or application programer. They provide the languages and procedures by which the user communicates or interfaces to the Computer Complex. The capabilities and use of these programing tools are documented in the "LRC Computer Programing Manual" and related reference material.

The Operation and Control and Special System control software comprise a large quantity of modular system elements which perform numerous functions associated with controlling the orderly flow of jobs through the computer, and operating the various devices contained in the peripheral pool. Most of this software is not directly invoked by the user in the development of his Fortran program, but is implied or called at various levels of communication and control by the operating system as it processes jobs through their various stages.

The Real-Time Monitor, at the center of Figure A7, is the heart

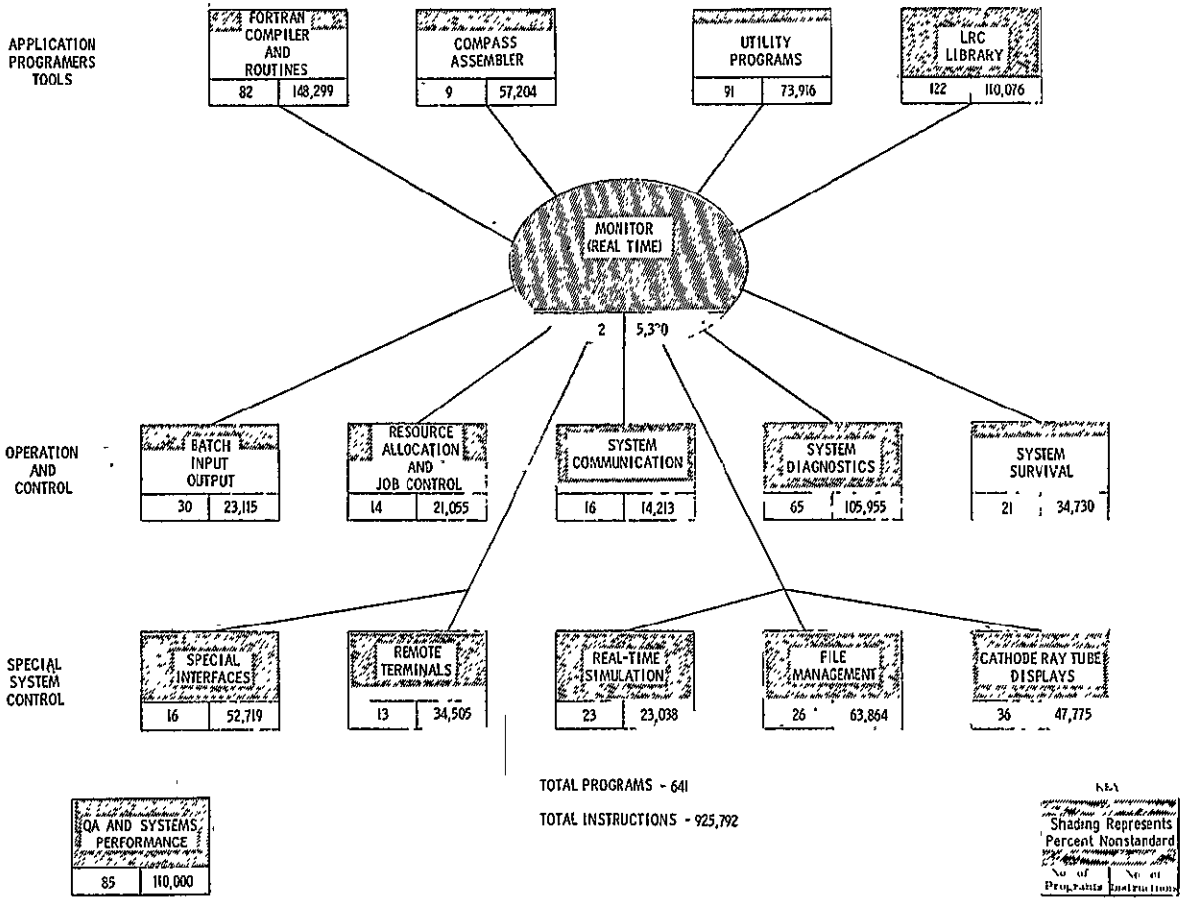


Figure A7. - Operating system software.

of the entire operating system. Its primary function is to provide orderly and responsive communication, control, and resource allocation functions for the entire operating system. Its function is somewhat analogous to that of a dispatcher, where the frequency of occurrence of events to be processed is in the range of thousands per second. The most critical requirement of real-time monitor is the ability to maintain a response time which is minimal, nearly constant, and definable under all possible conditions of system operation. This is the cornerstone upon which digital simulation is based. We have recently completed the implementation of a third version of the monitor portion of the operating system, and with it have achieved a response to time critical events which at no time exceeds 130 miscroseconds. Considering the size and complexity of the IRC Operating System, the current monitor represents a significant achievement in system design and performance.

Overall Design

The IRC SCOPE Operating System has been built around two key design techniques; Multiprograming, and Multiprocessing. Multiprograming can be defined as a mode of computer operation whereby the computational capability of a computer is directed to intermittently support one of a number of different applications which reside in the memory of the computer simultaneously. The distributive allocation of computational power among the various applications is controlled in a variety of ways depending on the nature of the individual jobs

currently being processed. In general, a priority number determines the preferred sequence.

Multiprocessing in the case of the LRC Operating System, can be defined as a mode of computer operation whereby all input/output operations as well as the logic and manipulation associated with them are performed in parallel with the multiprograming mentioned above. In addition, numerous (10-15) input/output operations can be proceeding in parallel with one another.

The purpose of these features, which have been incorporated throughout the entire operating system software, is to dynamically allocate the numerous resources of the computer hardware to a continuously changing workload requirement in an attempt to optimize the performance of the entire computer. The objective, of course, is not to just keep the machine busy, but to provide the best possible service for all users of the computer complex.

The operation of multiprograming and multiprocessing can best be explained by diagrammatic example. Figure A8 depicts a typical 6000 system. On the right, we have the central processing unit, which operates only on programs stored in the central memory located to its left. The ten peripheral processors (PPU's), each of which in effect is a separate computer with 4096 words of memory, communicate through half-duplex channels with all input/output equipment in the peripheral pool and with central memory.

The operational environment of LRC SCOPE is as follows (see Figure A9). As many as seven different application programs, resident

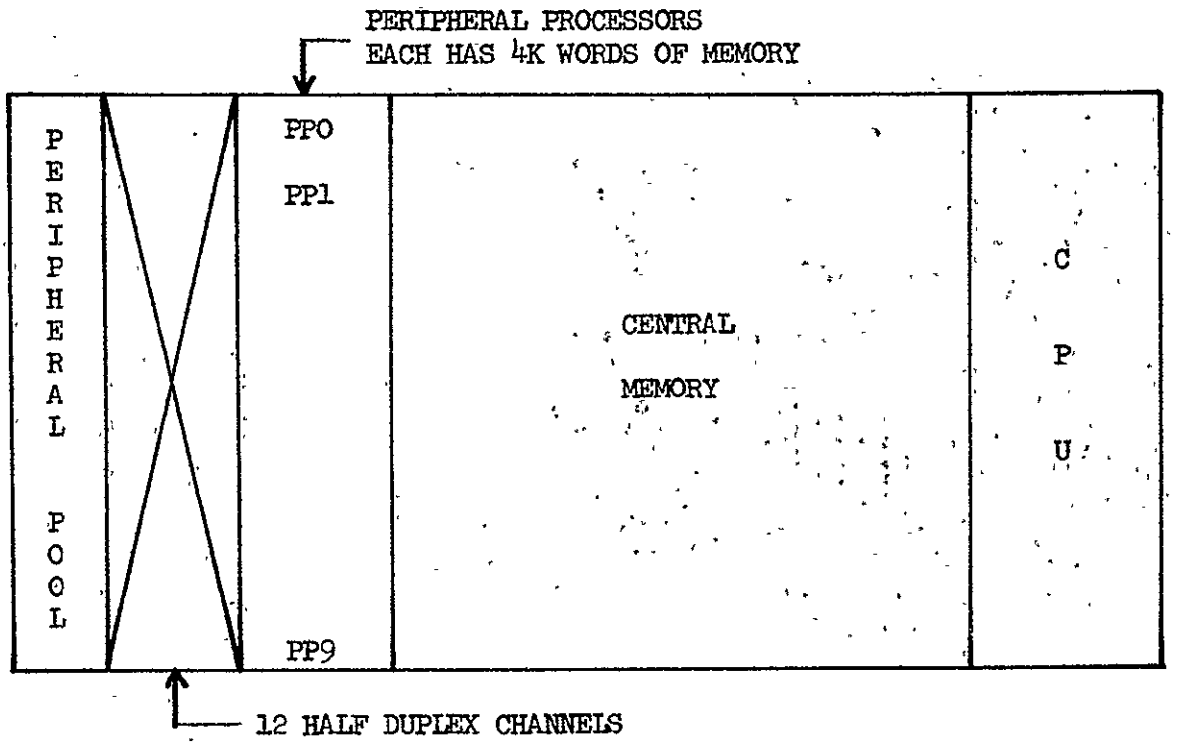


Figure A8. - 6000 system.

in central memory, can operate in a true multiprograming mode, depicted here by Program 1, Program 2, etc. The real-time monitor (RTM), permanently resides in a peripheral processor and a small portion of central memory. Also, a display driver permanently resides in a second peripheral processor to provide continual dynamic display and control for operator communication. The remaining peripheral processors constitute a pool of capability which, upon command from monitor, will perform various input/output and control tasks necessary to support the execution of application programs in central memory, as well as numerous system functions such as reading in jobs, printing results, communicating with remote terminals, etc. The operating system uses a small portion of central memory for job queues, tables, and other passive support and communication requirements.

Let us assume that Program 1 is in execution and that it is the highest priority job in central memory. It will remain in execution as long as its priority remains highest, until it requires external input/output activity. At this point, it will request that input/output be performed. When the request is recognized, monitor will immediately redirect the CPU to the next highest priority job in central memory that is ready to execute and will command a pool peripheral processor to perform the appropriate input/output operation required by Program 1. When the outstanding input/output operation is complete, monitor will place Program 1 in a "waiting for central processor" status which, in effect, places the program in contention

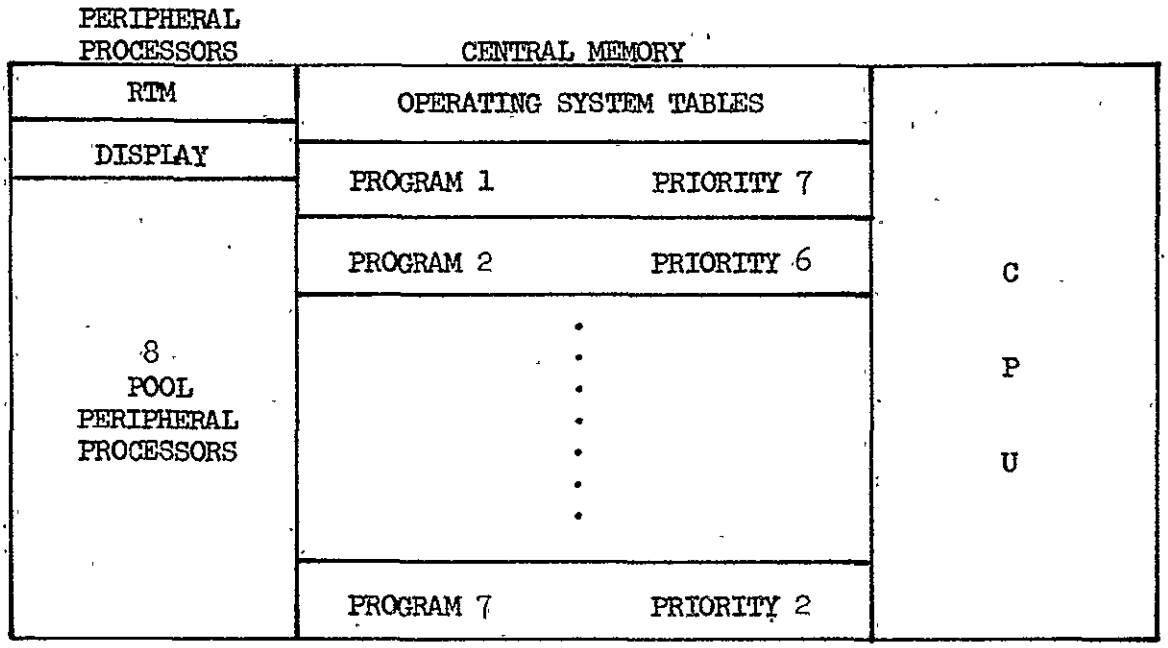


Figure A9. - Operational environment.

for subsequent use of the central processor. This allocation of the central processor among the various programs in central memory (multiprograming) occurs hundreds of times per second. Each occurrence is created by the need of the application program for support activity from the operating system, such as input/output operations. The handling of these support activities by peripheral processors is accomplished in parallel with the central processor and in parallel with one another (multiprocessing).

BIBLIOGRAPHY

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

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

3. Computer Programing Manual. (Internal manual for National Aeronautics and Space Administration, Langley Research Center Digital Computer Complex.)