NASA CONTRACTOR REPORT

NASA CR-18

NASA CR-1868

SPACEBORNE COMPUTER EXECUTIVE ROUTINE FUNCTIONAL DESIGN SPECIFICATION

Volume II. Computer Executive Design for Space Station/Base

by J. R. Kennedy and W. S. Fitzpatrick

Prepared by
COMPUTER SCIENCES CORPORATION
FIELD SERVICES DIVISION, AEROSPACE SYSTEMS CENTER
Huntsville, Ala. 35802
for George C. Marshall Space Flight Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • OCTOBER 1971

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
NASA CR-1868		

4. TITLE AND SUBTITLE	5. REPORT DATE
Spaceborne Computer Executive Routine Functional Design Specification - Volume II: Computer Executive Design for Space Station/Base	October 1971
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT #
J. R. Kennedy and W. S. Fitzpatrick	
9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. WORK UNIT NO.
Computer Sciences Corporation Field Services Division, Aerospace Systems Center 8300 South Whitesburg Drive Huntsville, Alabama 35802	11. CONTRACT OR GRANT NO. NAS8-24930
	13. TYPE OF REPORT & PERIOD COVERED
12. SPONSORING AGENCY NAME AND ADDRESS	Contractor Final Report
National Aeronautics and Space Administration Washington, D. C. 20546	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

16. ABSTRACT

This report describes computer executive functional system design concepts derived from study of the Space Station/Base. Information Management System hardware configuration as directly influencing the executive design is reviewed. The hardware configuration and generic executive design requirements are considered in detail in a previous report (System Configuration and Executive Requirements Specifications for Reusable Shuttle and Space Station/Base, 9/25/70). This report defines basic system primitives and delineates processes and process control. Supervisor states are considered for describing basic multiprogramming and multiprocessing systems. The report defines a high-level computer executive including control of scheduling, allocation of resources, system interactions, and real-time supervisory functions. The description is oriented to provide a baseline for a functional simulation of the computer executive system.

This report is Volume II of a three-volume report entitled "Spaceborne Computer Executive Routine Functional Design Specification." The other two volumes are:

Volume I: Functional Design of a Flight Computer Executive Program for the Reusable Shuttle

Volume III: Executive Routine Primitives and Process Control

17. KEY WORDS	18. DISTRIBUTION STATEMENT
Space Station Executive Supervisor Multiprocessor Multiprogramming Information Management System Data Management System	Unclassified - Unlimited

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	88	$3.00

*For sale by the National Technical Information Service, Springfield, Virginia 22151

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

SUMMARY

The purpose of this report is to describe the functional design of an Executive computer program for the Space Station. It is based upon a study of the Information Management System (IMS) of the Space Station/Base. The study included a survey of Station-related documents, discussions with NASA personnel and review of predecessor operating systems.

Two multiprocessors, a dual-redundant processor and a simplex processor are included in the IMS. Each computer is composed of identical basic components to reduce the spares, maintenance and training requirements. The computers contain microprogrammed control memory, 32-bit word lengths and perform 200-250 thousand operations per second. Main memory modules range from six specified for the Experiments computer to one for the Biomedical computer. Each module is 16,384 32-bit words, 1-microsecond cycle time, and has a 6-port interface. Specifications for the several required modules of auxiliary memory have not yet been identified.

The Data Bus Controller provides the facility for controlling the transfer of data and furnishing the Data Management System (DMS) computer with activity and status information. It contains programmable control; memory for buffer storage; decoding for modem subcarrier selection; and bit-rate transitions. The Controller interfaces the DMS computer to the Digital Data Bus. The Digital Data Bus is a time-division/frequency-division multiplexed bus.

Digital terminals interface devices through modems to the Digital Data Bus. The Remote Data Acquisition Unit (RDAU) is a standard interface component connecting the terminal to external sensors and receivers. Analog data terminals similarly connect external signals to the Analog Data Bus.

A Process is the series of actions required to perform a unit of work in the IMS. System Primitives are basic functions used to describe Executive actions and states related to Processes. Construction is the collection of subprograms, library functions and system procedures associated with an identifying block (Process Control Block) to form a Process.

The Space Station Executive is the collection of algorithms that exercise computer control over computer resources. These algorithms may be implemented in combinations of software, microcode and hardware. The DMS multiprocessor is the focal point of control, including the data bus, and as initiator for the other computers.

Several methods are available to control intercomputer communication. The objective is maximum speed of interaction but minimum interference. Intracomputer data movements require checks and counterchecks at process interfaces.

Scheduling of computer resources is sensitive to configuration, job stream characteristics, and mission goals. Separate scheduling strategies for each computer include those for the DMS, Experiment and GNC. The Biomedical computer is an integral part of the Biomedical Subsystem and is not controlled from the DMS Executive. A specific scheduling strategy is not outlined for the Biomedical computer.

DMS and Experiment Scheduling are functionally similar since their configurations are identical and the resources to be allocated are similar. The Scheduler updates a queue when a new process requests computer time or when resources are returned to the Executive. Dispatching of processor time on the DMS is on a time-quantum basis which provides time sharing ability.

Experiment Scheduling Strategy is similar to an earthly laboratory environment. The DMS initiates Experiment scheduling from the timeline and crew requests. The Experiments Scheduler is a table generated list of attributes. The Experiments will be time-event driven under pseudo-control of experiment supervisors.

The GNC Scheduler considers unconstrained, periodic and interrupt processes initiated by the DMS Executive. The GNC is envisioned to be a foreground/background operation driven by the time and event table and dispatched by interrupts.

A supervisor call is an interrupt-type means by which a process can request allocation or deallocation of resources or other Executive services. The Input Job Processor is a routine for interpreting job control requirements. The Timeline Control routine maintains a high level language version of the timeline which it decodes and merges into the discrete time-event stream.

Data bus control can be accomplished by oversample, polling or interrupt techniques. The Data Bus Controller is assumed to perform many repetitive I/O tasks. The Data Bus Programmer prepares the Controller to cycle through its functions. The Controller supports device-to-device transfers which may require the mixing of ancillary data and the resolution of conflicts for shared resources.

Input/Output control routines will minimize delays to processors and maximize device utilization. The Process Control Block will contain information required by the I/O handlers. I/O error recovery is planned for each phase of information exchange. Procedures are considered for saving I/O data during equipment down-time.

Interrupts will appear at processor interrupt-interface points. An Executive controlled interrupt mask ignores irrelevant signals but passes

relevant interrupts. Requirements for arming, disarming, preprocessing, acknowledging, stacking and queuing of interrupts are recognized.

Polling is an alternative to interrupts. It is accomplished by querying devices with a polling signal or standard interface word. The list of polled devices receives priority arranged service.

Projected computer components are highly fault-tolerant. Real-time diagnostics perform tests to verify correct operation or to isolate suspected malfunctions. The failure rate is determined for transient failures and measured against acceptable thresholds. Maintenance diagnostics are selected to trace hard failures to lower levels. Processor design can include control and sense lines to enable the execution of an instruction one step at a time, the results of which can be compared to the simulated step value.

If the last processor of a multiprocessor fails, it will attempt to recover itself. A watchdog timer alarm will alert the second multiprocessor of the complete failure of the first. The surviving multiprocessor will become the DMS and take executive control. The survivor will reconfigure the system to make maximum use of available components and to place malfunctioning components off-line for replacement. The configuration changes will be displayed and further changes can be initiated by the crew.

Utility routines perform the routine operating functions for the computer facility. Accounting, library routines and mathematical functions are examples of expected requirements for utility functions.

SECTION I. INTRODUCTION

NASA will accomplish many "firsts" with the realization of a manned laboratory as envisioned for the Space Station. One of these firsts will be the establishment of a general-purpose computing facility for a manned spacecraft similar to earthbound installations. Identification of the similarities and differences between spaceborne and earth-based facilities is not an objective of this report but are referred to, tutorially, for clarification of design tradeoffs. This report will provide a basis for functional simulation of the computer executive system. A functional simulation, combined with a representative jobstream simulation, will permit the development of detailed specifications and performance measurement criteria. The simulation could be constructed as an important validation tool.

The design of a Space Station Computer Executive System (Executive or Exec) is discussed in four parts:

- Configuration,
- System Primitives and Process Control,
- Executive Design, and
- Conclusions and/or Recommendations.

The configuration review is necessary to establish guidelines and major points of departure. For example, the presence of multiprocessors establishes requirements for defining Executive control procedures for multiprogramming. The Data Bus and I/O Switching system place specific requirements and constraints upon the Executive for special data routing capabilities.

Configuration definition is currently being pursued under other NASA contracts. Specific data and control definitions are not generally available. Where definition is required and not available, assumptions are identified based upon the most recent NASA documents (1), a configuration and requirements report (2), prior experience, and discussions with NASA personnel.

---

[1] MSFC-DRL-160, Line Items 8 and 13, July 1970, NAS8-25140.

[2] Kennedy, et al, System Configuration and Executive Requirements Specifications, NAS8-24930, Computer Sciences Corporation, September 1970.

System primitives and process control concepts define the basis for the Executive design discussion. Primitives and the associated control are defined in some detail. Volume III of this report discusses primitives and process control in greater detail and relates them to hardware implementation techniques. The discussion demonstrates the simplicity of design when performed in these terms and provides a vehicle for machine independent discussions.

The Executive functional design is presented as a high-level description of a spaceborne computing facility's computer control program (Exec). It is a functional representation of the hardware/software and software/software interactions in a multicomputer, multiprocessor, and multiprogramming environment. The extra-terrestrial constraints are particularly evident in the provisions for error detection and recovery and reconfigurability.

Trade studies and/or simulations are recommended at points where potentially critical assumptions are made with insufficient data. Several areas for additional study have also been identified. Some of these studies, such as job stream, are being conducted under NASA contracts and should readily lend themselves to simulation. Several areas are primarily treated with examples since the generality would not be made more clear by exhaustive but vague descriptions.

SECTION II. CONFIGURATION

The equipment presently proposed for the Space Station Information Management System (IMS) is listed in table 1. Some of this equipment has an indirect influence on the Executive. Image processing, for example, could be accomplished by a specially designed processor that would take advantage of hardware implementations of well-defined data analysis procedures. However, such a feature has not been specified and the application programs and concomitant requirements must be considered.

The entertainment equipment is not envisioned to be within the sphere of DMS control functions. On the other hand, a library of available tapes and microfilms could possibly be maintained by an application program. This would place a requirement upon the Executive for cataloging, storing, and retrieving entertainment information. In the absence of better definition, it will be necessary to make assumptions concerning requirements based on the presence of equipment and/or capabilities provided.

The remaining equipment listed in table 1, Computation, Data Acquisition, Data Distribution, and Bulk Data Storage, has a basic driving impact upon the Executive design and receives more explicit treatment. This equipment is discussed under two topics: (1) computers, and (2) data distribution. Thus, the configuration can be viewed from the computer Executive's point of view for the functional design to follow (Section IV. Space Station Executive).

A. Computer

The meaning of the word "computer" has become vague due to the proliferation of designs and the terminology and acronyms that have been loosely used to describe various hardware assemblies which might contain more or less of the capabilities connotated. Therefore, a list of definitions is presented, the expected characteristics of major Space Station assemblies is reviewed, and major unique components are described in order to establish an unambiguous foundation for discussion.

1. Definitions.

- Main Memory. The addressable storage from which software code (instructions and data) are extracted and operated upon.

- Auxiliary Memory. The storage outside the range of the Arithmetic Logic Units (ALU). Thus, information must be transferred to main memory to be operated upon by the ALU.

TABLE 1. EQUIPMENT LIST

Equipment Name	Quantity	Equipment Name	Quantity
Computation		Bulk Data Storage	
DMS Processors	2	Buffer and Control	1
Main Memory	5	Transport Controllers	2
Auxiliary Memory	4	Transports	2
		R/W Electronics	1
Experiment Processors	2	Switching Matrices	2
Main Memory	6		
Auxiliary Memory	4	Image Processing	
		Film Viewer	1
GNC Processors	2	Film Reader	1
Main Memory	2	Image Processing Control Station	1
Biomed Processor	1	Adjustable Multichannel Filter	1
Main Memory	1	Working Image Storage	1
		Video Storage	1
Printers	2	Permanent Storage	1
		Time Reference Unit	1
Data Acquisition		Plotter	1
Analog Terminals	6	Microfilm System	1
TV Control Units	10		
Signal Conditioners	10		
Remote Acquisition Units	48	Entertainment	
Sensors	2400	TV Monitors	14
		Film Library	1
Data Distribution		Audio/Video Tape Storage	2
Digital Terminals	15	Portable Audio Tape Unit	2
Decoders	16	Video Tape Unit w/Vidicon	2
Command Decoder	1	Portable Microfilm Viewers	2
Coax	1150 ft		
Twisted Pair	1050 ft		

- Arithmetic and Logic Unit (ALU). The arithmetic function (fixed and floating point), shifting, logic functions (decisions), and control functions are performed here. The portion of the ALU that controls sequencing, contains the decode matrix, and sends signals to circuits as decoded from the instruction may be referred to, at times, as the control segment of the ALU.

- Microcode. The replacement of the logic gate matrix by formatted instructions either similar to conventional machine language code or isomorphic bit to data path control. The outputs of microcoding, then, become the control signals for the system.

- Firmware. Software or microcode stored in read-only memory and thus not subject to dynamic change. There may be associated Local Store.

- Multiprogramming. The interleaving of program executions so as to produce the appearance of simultaneous execution.

- Multiprocessing. The operation of more than one cooperating ALU in a single computer (ALUs plus memory and switching).

2. Computer Characteristics. Each computer is envisioned as being composed of identical basic units to enhance the spares, maintenance, training, and cost aspects through broad commonality. A computer is thus composed of one (simplex) or more (multiprocessor) arithmetic and logic units (ALU), and one or more memory module units. Recently specified characteristics of the ALU include:

- 200-250 KOPS
- Floating point arithmetic
- Shared memory bus (in multiprocessor configuration)
- 8-channel, high-speed I/O interface
- Microprogrammed LSI control memory
- 32-bit plus check bits word length manipulation.

Similarly, the following characteristics for main memory modules have been specified:

- Monolithic
- 16,384 32-bit plus check bit words

- 1 microsecond cycle time
- 0.5 microsecond access time
- 6-port interface
- Random addressability.

The characteristics of auxiliary memory are assumed to be similar to main memory (possibly identical). The auxiliary memory could be different; for example, it could exhibit a comparatively slower access time or even a different organization. Variations in configuration of memories, addressing schemes, procedures for information coding and manipulation, and effects upon the overall system constitute a valuable trade analysis area that should be supported by simulation.

The definitions and characteristics outlined permit a generic description of the computers aboard the Space Station:

- A multiprocessor (DMS) for the control of scheduling and the data bus, and general computing facility use.
- A multiprocessor for the control of experiments and the compression/recording of experiment data.
- A dual redundant (two completely separate and independent simplex computers) system for Guidance and Navigation (GNC).
- A simplex computer which is an integral part of the Biomedical subsystem. This computer is envisioned to be dedicated to the collection and processing of experimental medical information.

a. DMS Computer. A multiprocessor has been identified as the focal point for control of the complete Space Station onboard Information Management System. To the multiprocessor previously described as formed by the combination of ALU elements and memory modules is added switching interfacing mechanisms and the Executive software control capability. This totality forms the DMS computer. The DMS Multiprocessor is designated as commander of data bus traffic and the center of nonexperiment computing power. (This prime unit is sometimes referred to interchangeably as the Data Management System (DMS) computer and the Information Management System (IMS) computer. In this discussion and the discussions to follow IMS will refer to the total Information Management System. DMS will refer to the prime computer performing the central data management functions.) One general arrangement of a DMS Multiprocessor is shown in figure 1. This simplified diagram illustrates only the most basic building block connections.

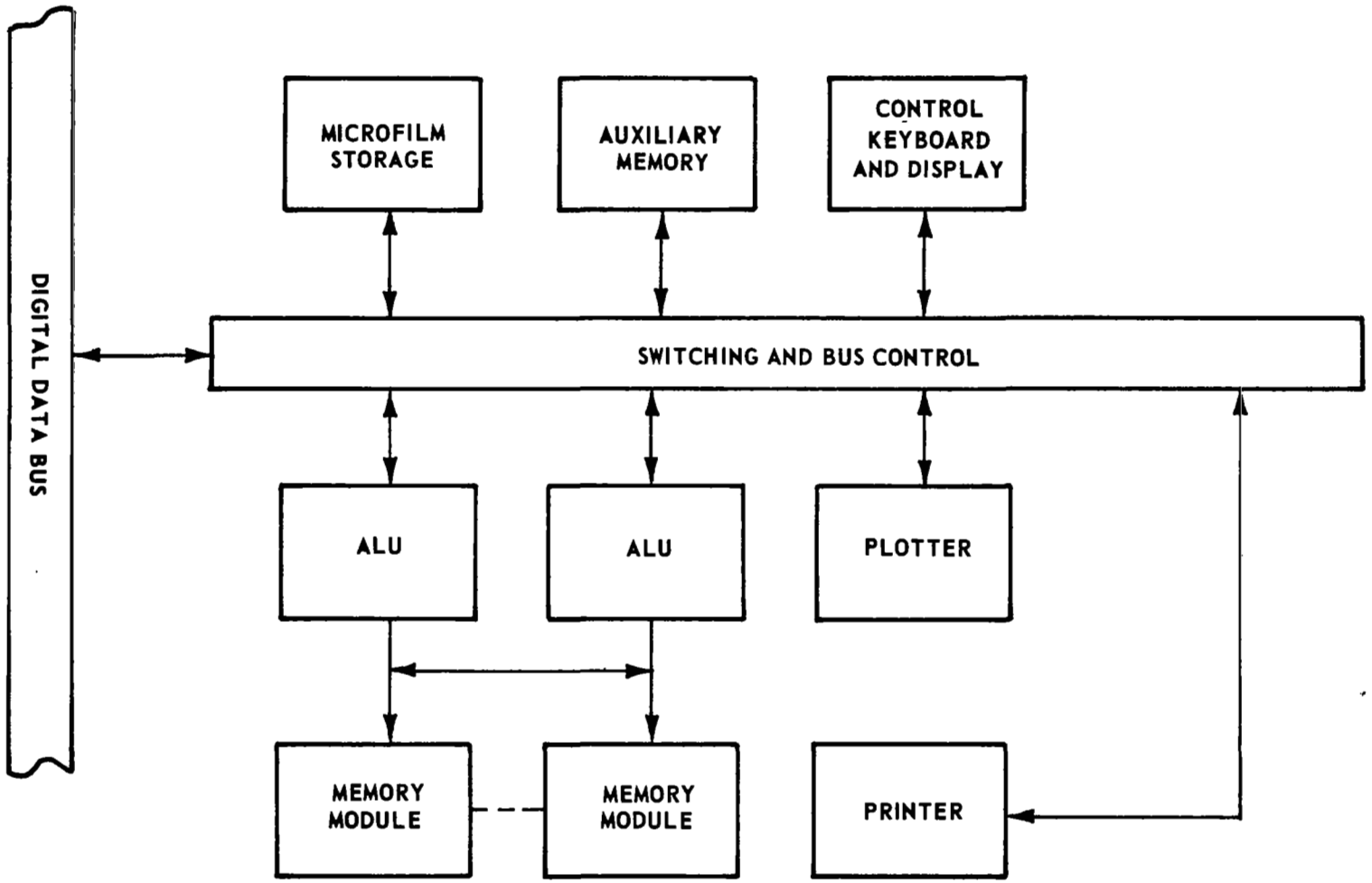

FIGURE 1. MULTIPROCESSOR BLOCK DIAGRAM

Each ALU, connected to memory modules, forms a simplex computer. The simple arrangement shown in the diagram reinforces the indication of isolation of computing elements. However, more interactive capabilities are implied and, therefore, considered in the Executive design. Several features that are commercially available for control of multiprocessor interaction should receive trade study attention, supported by simulation. The interactive capabilities cannot be exploited safely without extensive simulation to prove that there are not diminishing returns or potential degradation in performance and/or reliability.

b. Experiments Computer. The second multiprocessor is specified to control experiments from the dynamic schedule initiated by the DMS computer and to direct the processing, reduction and storage of acquired experiment data. It is further required to provide backup to the DMS multiprocessor in the event of catastrophic failure of both DMS system ALUs. Figure 1 also represents the Experiments Multiprocessor since, according to the assumed baseline, the computer systems (DMS and Experiment) are identical and thus physically interchangeable. The full extent of the interaction between the multiprocessors will require additional definition.

c. GNC Computer. A dual redundant computer has been specified for guidance/navigation aboard the Space Station. The best information to date indicates that these are two identical but unconnected computers formed from the same basic modules described earlier. The operational posture of this equipment has been defined as follows. One simplex computer will be operating under the simplest Executive form. In the event of failure, a manual switchover would be made by a monitoring astronaut to the backup computer. Intuitively, this imples a cold start design of considerable flexibility.

It seems illogical to have a redundant, idle system that must be brought up and manually started when the number one system fails. The system could be expanded to include synchronized operation of the dual simplex computer systems. The outputs of the two systems would then be compared in a voting-box manner. However, with two votes, the only conclusion that can be drawn from disagreement is that one (or two) computers have failed or that the voting mechanism is in error. It will not indicate if, or which, output is useful; however, the second system would be up and ready. Further investigation is indicated in this area.

A solution might be proposed for the present. The Space Station GNC computer must be reliable and accurate when in use, but it is under-utilized. Its routine function is to provide position information and control data for the IMS. These are important functions, but other less frequent tasks (i.e., docking maneuvers) are critical. Perhaps the spare ALUs and memory units can be stowed with switching to the GNC. During the critical periods, at least three ALUs would operate synchronously and vote to insure correct computations, identification of failure, and continuity of operation.

When the GNC is not in critical-function use, it will provide spares for the rest of the system. Spares can be maintained in a powered-down condition. However, the Executive system should have knowledge of every unit, its condition, and the means (via the data bus) to checkout spare units after power is applied. The spares stowing procedure could include design features that would allow the DMS computer to apply power, warm-up, checkout, and switch modules.

It is beyond the scope of this report to establish an operations philosophy or significant Space Station design alteration requirements. The procedure described above supports the contention that a viable, dynamic system is predicted for GNC and that the Executive will support the GNC functions.

d. Biomedical Computer. The data collection and computations associated with biomedical experiments will be performed on a dedicated simplex computer formed from the previously defined modules (ALU, main memory, and auxiliary memory). The computer will be interfaced to the Data Bus. Its computations will not be scheduled nor will it be controlled by the DMS computer, with the exception of access to the Data Bus.

The biomedical subsystem will be an integration of a subject-astronaut, sensors, data acquisition and signal conditioning, biomedical experiment equipment, computer, and investigator-astronaut. The investigators are normally specified to be medical doctors. The subjects are scheduled as available (skills plus free time). The computer programs will provide the flexibility required for mixing subjects, skills, and experiments. This flexibility requires good man/machine interaction and well-managed data files.

The man/machine interface will be accomplished by interactive CRTs. A high-level language will permit the investigator to acquire data, perform prespecified analysis, and direct disposition of results. This indicates that a bulk storage is available directly, that analysis hardware may be required (special function processors), and that the difficulties associated with manipulation of the data be made completely transparent to the user. If the subsystem includes complex chemical analysis (e.g., blood-gas analysis, lung volume and diffusion, etc.), some preprocessing hardware will be a system requirement. EEG and ECG measurements and analysis will require preconditioning and filtering arrangements.

The computer will provide a laboratory data management tool. It will provide standard formats for data presentation, automatic logging, retention of test results, statistical comparisons, calibration summaries for equipment in use, and monitor (via the high-level language) methodology to warn of nonuniform procedures. It would not be unreasonable to expect automatic diagnostic procedures and "learning machine" type interactive computer programs to reduce training time and to permit operation by various other skill specialists.

B. Data Distribution

The Data Bus will form the primary connective route (physical channel) for transmission of information and control data within the computer complex. The acquisition and display of data is implemented with Digital Terminals, Command Decoders, Printers, Visual Displays, Lights and Audio Signals. Some information, such as test and checkout stimuli, responses, control signals, and other signals defined as internal to various major subsystems will be routed through separate local data distribution networks. Control of these data is beyond the scope of this investigation since it does not directly affect computer loadings. The built-in test and checkout incorporated into major subsystems will provide passive backup to the active fault detection and warning system. Built-in monitoring and testing circuits will be especially important during periods of inactivity and the subsequent reactivations for system readiness verification.

Data distribution is performed under computer control through the data bus controller, the data bus, digital terminals, and remote data acquisition units. Several additional assumptions are made regarding the data bus controller in Section IV.

1. Data Bus Controller. The DMS computer interfaces to the Digital Data Bus via the Data Bus Controller. The Controller will provide buffer storage and decoding capabilities for modem subcarrier selection and bit-rate transitions. It will provide the facility for enabling the transfer of data and furnish the computer with status information. The computer will establish the control frequency selection and addressing of codes and message priorities. The Data Bus Controller will provide formatting, message assembly, status analysis, and external requests/acknowledgements. Data Bus Controller functions are further discussed in the software design sections.

2. Data Bus. There are both Analog and Digital Data Buses. The Analog Bus will accept signals from modems that have received their instructions from the computer via the Data Bus Controller. All frequencies will be preassigned according to the interfacing modem. Signals captured from the Analog Bus will be recorded on magnetic tape or displayed on CRTs.

The Digital Data Bus is controlled by the Data Bus Controller. Assumptions are made about the Controller when required to simplify the design discussion. The Controller will route control signals to peripheral units via a standard I/O interface word. The process could be represented by a two-wire serial bit stream Data Bus System. The single data path could be time-shared by the I/O devices. There are assumed to be several carrier frequencies, which is logically equivalent to having several wires. Thus, the Data Bus is a time division/frequency division multiplexed bus. The computer programs will maintain lists of I/O devices, their carrier frequencies, and the frequencies in use/available. The bus has not been specified as synchronous or asynchronous with respect to time.

Probably, the Data Bus Controller will format the standard I/O interface word and broadcast it to all Data Bus interfaces on that frequency. The terminal designated at that frequency by that identification code will further decode the interface word for additional instructions.

3. Digital Terminals. The Digital Data Bus, a multi-channel device for coded digital data transfer, is interfaced to the Remote Data Acquisition Units (RDAU) and thus to digital input sources via digital terminals. Clock frequencies (from the time reference) and the terminal addresses (unit designation) are intercepted by the digital terminals. The terminals decode the address and thus each responds only to its uniquely designated code. The terminal activated by the designation code transmits an acknowledge code to the computer by placing it on the Data Bus at the return carrier frequency. The frequency, which is comparable to channels on many ground-based computers, utilizes subcarriers to segregate logically independent digital bit streams. This reduces individual bit-rate requirements by providing more logical facilities. Operational visibility is enhanced by providing separation among data transfers.

Buffer storage is provided in conjunction with the Digital Terminal. The devices interfaced to the terminal may or may not require separate buffering. Examples can be postulated such that data must be held by one or more devices for transfer to the terminal buffer, so that both buffered and unbuffered devices are expected to be interfaced to Digital Terminals.

4. Analog Terminals. An Analog Terminal intercepts and decodes the unique device designation code associated with it. Each Terminal can input eight (8) signals. Any or all of the eight signals may be selected for output onto the Analog Data Bus. The control of the terminals is the same as for Digital Terminals.

5. Remote Data Acquisition Units. Remote Data Acquisition Units (RDAUs) are interfaced to the digital terminals. Each RDAU will accommodate up to 64 analog (converted A/D) and/or 8-bit signals. Thus, a Digital Terminal may ultimately interface up to 512 inputs to the data bus interface.

To reach a particular signal or bit, as discussed earlier, a standard interface word is broadcast under computer control. The terminal identified in the standard code word will then further decode the standard interface word to determine the mode. The transmission will include the address of an RDAU for digital terminals and instruction codes to reference the RDAU connections and RDAU procedures. The standard word will include the instruction code and reference frequency for analog terminals.

The RDAU will signal the Bus Controller for the completion of command transfers. The Controller will then transmit any required signal conditioning

commands, amplifier range settings, signal conversions, multiplexing, or calibration instructions, as provided by the controlling computer.

The RDAU is highly self-contained and process control-oriented. To relieve the controlling computer of excessive transmissions of commands/requests, the RDAU has extensive capabilities to continue independent action. It contains memory, self-test features, and is programmable for three modes of operation:

- Compare mode. The RDAU will sequentially scan its inputs and test each for out-of-limit conditions. The limit thresholds are set by the control computer via the standard interface word.

- Sequential output. The RDAU will sequentially scan each of its inputs and transmit the conditioned and/or digitized data onto the data bus.

- Single channel. The standard interface word can cause the RDAU to sample a single input. It can direct a single sample or command that the single input be repetitively sampled.

In simplified fashion, the RDAU's path to the computer is illustrated by figure 2. The Analog Bus, Backup Digital Bus, and Intra-system Buses are not depicted since they will not clarify the present computer executive study.

C. Configuration Summary

Once the Space Station is operative, it will assume many of the attributes of a ground-based processing facility. The essential difference is in the special precautions required where life support and station safety are involved due to the hostile space environment. The normal working environment, however, is recognized to be similar to a ground-based laboratory which can be maintained by more ordinary procedures. Less stringent requirements will permit use of automatic testing/backup, load sharing and less equipment to provide station computing continuity and the ability to recover from powered-down periods.

The Space Station apparently will not have periods when 100% performance of all computer systems is required. If critical functions are identified and planned for by partitioning, a lower level of reliability (with recovery) can be tolerated in noncritical areas. Recovery procedures can be adjusted to acceptable levels to minimize experiment data loss or interruption of processing. However, for the GNC, an increased reliability and a functional use for spares was proposed.

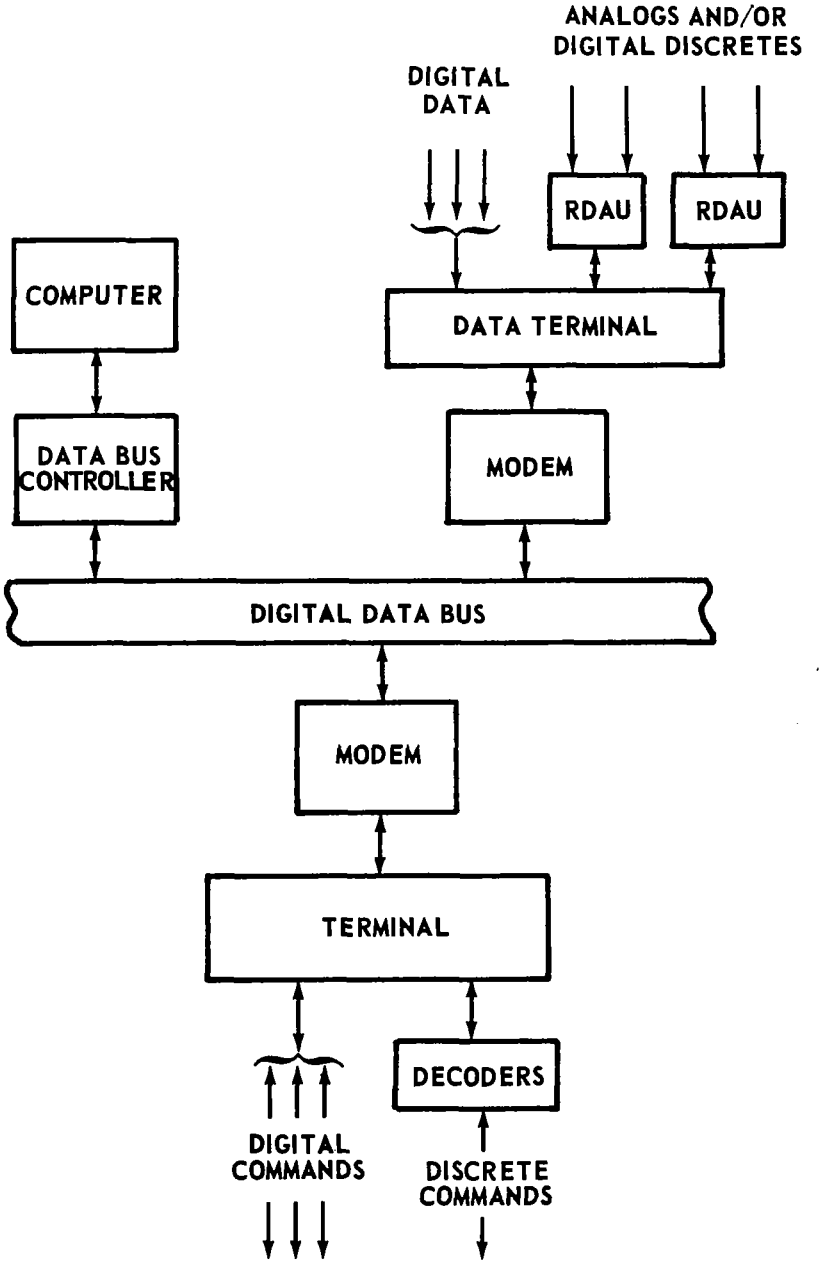

FIGURE 2. DATA DISTRIBUTION SYSTEM

A functional simulation of the Space Station configuration is recommended to ascertain more detailed requirements and, thus, establish a baseline for preparation of detailed specifications for the Executive software. The driver for such a simulation would be the representative job stream. A large part of job stream definition can be derived from the experiments load (3) and the expected time-line activities.

---

[3]Blue Book Update, Reference Earth Orbital Research and Applications Investigations, NAS8-25051.

SECTION III. PROCESS CONTROL AND SYSTEM PRIMITIVES

The supervisory aspects of controlling program execution within a multipurpose digital computer environment have become so complex that not only is it difficult to implement scheduling procedures that have predictable effects, but it has also become difficult to establish design requirements and describe a control method that exhibits primarily desirable features. This complexity is an inevitable result of demanding more out of computing systems in terms of the total number of tasks to be completed in a given time interval with highly automated self-determination. Since faster program development is also desirable, systems have been further complicated by the addition of functional responsibility in the area of development support through program debug, text edit, language translation, and file maintenance capabilities. Design approaches can be simplified by recognizing that many functional responsibilities can be isolated through partitioning or segmentation into well defined and easily managed subfunctions. The purpose of this section is to define certain basic system functions that are independent of configuration.

A. Process Control

The following discussion outlines the concept of a "process" as it relates to the computer executive function. Based on this concept, the important features of process "construction," "primitives," "control," and "termination" are discussed. Volume III (4) of this report discusses these features and details of possible implementations.

1. Concept of a Process. The usual quantity of work referred to in discussions regarding computer systems is a "task." "User tasks" and "system tasks" have been accepted as terms for describing the entities that an executive routine deals with in its supervisory capacity. A task will be similarly regarded in this discussion as a specific quantity of work to be accomplished.

Most of the discussion, however, will be concerned with the sequence of actions performed in order to complete a task. This sequence is referred to as a "process." It is important to note that a process may execute code from

---

[4] Kennedy, Sr., J. R., Spaceborne Computer Executive Routine Functional Design Specification, Volume III: Executive Routine Primitives and Process Control, Computer Sciences Corporation, March 1971.

either system or user (application) programs, or both, in a more or less arbitrary order. Since the control devices of process-oriented systems may include those for stopping process execution, code that is common to several processes must be reentrant. To place the concept of a process in perspective, we define the following heirarchy:

- Process
- Routine
- Primitive

The Process is composed of Routines (procedures) and Primitives with connective coding. Similarly, Routines are made up of Primitives, possibly other Routines, Coding, and the associated linkages. Primitives are the basic cells from which Processes may be constructed, although a Primitive may consist of a collection of Primitives and linkages. Primitives are immediate candidates for microcode implementation.

2. Construction. In its broadest sense, process construction consists of program design and coding followed by translation to machine-executable code, collection into a module that is mapped onto main (instruction) memory, copying into main memory, and creation and initialization of a block of main memory that constitutes a set of state variables for control of the process.

For purposes of this discussion, a simple definition of construction will be sufficient. It includes:

- Collection and linking,
- Mapping to main memory, and
- Process control block formations.

Collection and linking is the gathering together of the various routines constituting a set of process code. This may include relocatable object modules, system routines, library functions, and machine code routines. After the code is gathered, it is mapped onto a contiguous set of instruction memory locations plus the prescribed locations for machine code routines. Once collection and mapping have been accomplished, a Process Control Block (PCB) can be constructed. Figure 3 shows a possible structure for a typical PCB and conceptual access mechanism. The contents of this block are for the most part self-explanatory; however, table 2 defines the entries.

It should be pointed out that no specific memory word size or processor register set is assumed; the organization of the PCB is therefore subject to

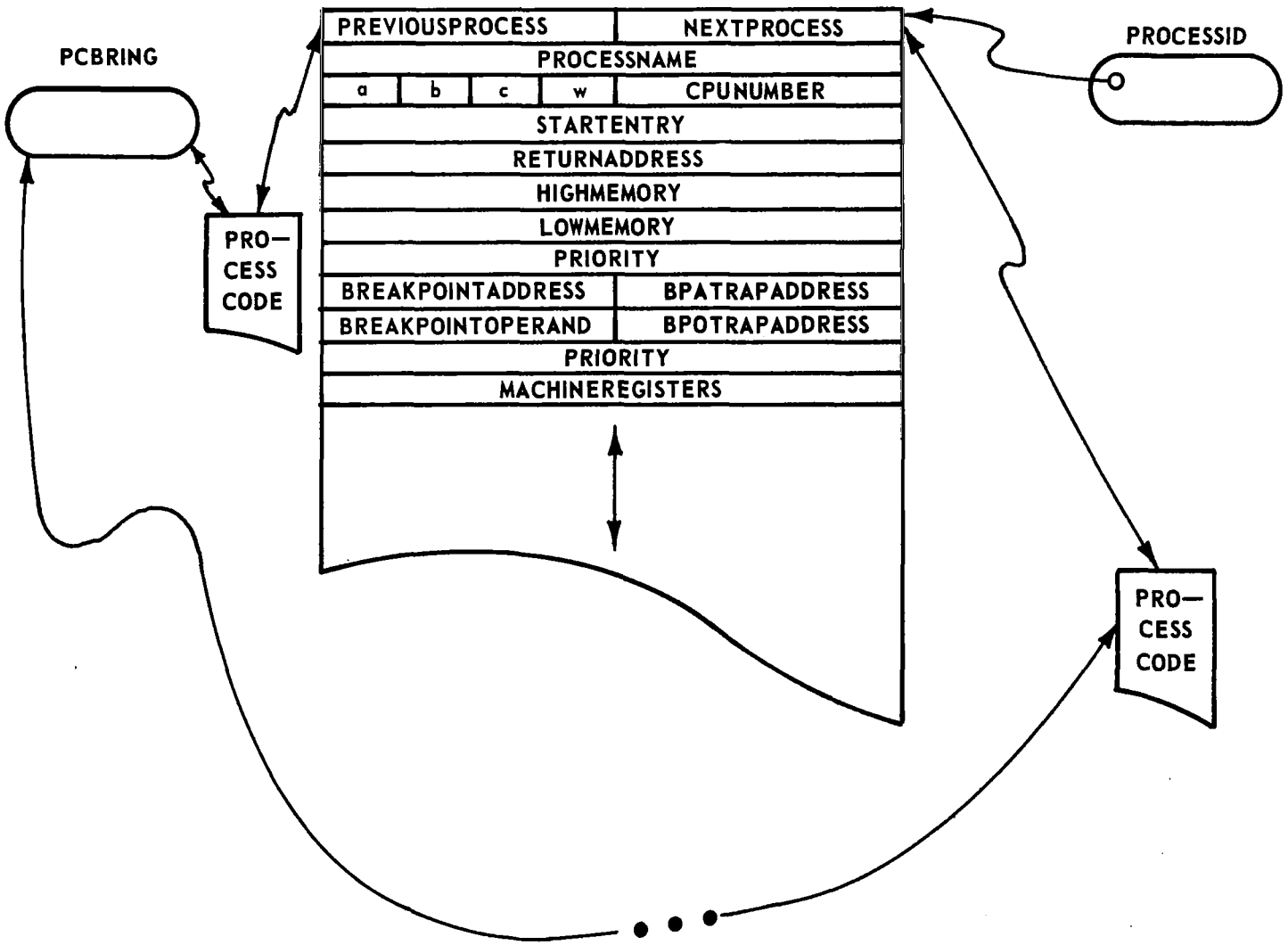

FIGURE 3. PROCESS CONTROL BLOCK

TABLE 2. PROCESS CONTROL BLOCK ENTRY DESCRIPTIONS

ENTRY	DESCRIPTION
Previousprocess	Pointer to the predecessor PCB on the ring.
Nextprocess	Pointer to the successor PCB on the ring.
Processname	Unique name for this process.
a b c	Three bit process state indicator.
w	Counter showing the number of unserviced WAKE primitives invoked for this process.
CPUnumber*	Hardware address of the processor unit (ALU) associated, during execution, with this process.
Priority	Relative process priority.
Startentry	Instruction memory address of the first instruction.**
Returnaddress	Memory address of the next instruction when process activity is interrupted; execution will be resumed at this location.
Highmemory	Largest instruction memory address associated, for protection and access purposes, with this process.
Breakpointaddress	Instruction memory address which, if it becomes the argument of an instruction fetch cycle, will cause an internal processor trap to a predetermined instruction memory address specified by BPAtrapaddress.***
Breakpointoperand	Instruction memory address which, if it becomes the argument of a datum fetch cycle, will cause an internal processor trap to a predetermined instruction memory address specified by BPOtrapaddress.***
Machineregisters	A block of words reserved for saving all programmable processor registers when process activity is stopped. Must include all registers depicting process state information.

* Entries a, b, c, w, and CPUnumber constitute the Processor Status Word.

** The exact meaning of all main memory addresses is dependent on the details of hardware addressing. The preliminary organization shown here is merely representative. For instance, if data and instructions are separated a high and low data memory address would be required; if a paged memory is used, the page file map would be saved.

*** These values would have meaning only when the associated processor is operating in a debug mode.

optimization in a specific architectural case. Also, the storage area reserved for PCBs is assumed to be protected from applications processes through definition of a set of privileged instructions.

B. System Primitives

A set of basic operators (primitives) is defined in table 3 to permit the functional discussion of controlled software. A primitive may implement a simple function or concatenate a complex relationship of functions, including the use of other primitives. System primitives will be especially useful in future specification and detailed executive software design. Their definitions are specific and unambiguous. Furthermore, the implementation can be performed in software, microcode, or hardware. Primitives can be analytically simulated by software implementation with measurements made to determine the increased effectiveness and efficiency realizeable by converting to microcode or hardware.

System supporting procedures, macro calls for supervisor functions, make use of system primitives. The immediate difference is that supporting procedures are visible to the application programmer. Supporting procedures fall into two classes: (1) User, and (2) System-restricted. User permitted procedures are the supervisor functions made available to the application programmer via simple mnemonics (e.g., EXITI) to perform several complicated executive functions. The coding of such procedures will not appear in-line with the application, but rather be implemented via trap or call execution. Process parameters required will appear in-line and in the Process Control Block. System-restricted procedures, of course, are called only within the executive software and are not generally available to the application programmer.

A sample support procedure is illustrated in figure 4. An exit or abort process is initiated by a trapped EXITI or ABORTI mnemonic. These commands request the removal of a specified process from further ALU resource competition and the release of associated allocated-resources. The ABORTI, during debug mode, will cause additional processes such as postmortem dumps, error testing, and status recording to be made available.

A process has been defined as a sequence of actions performed to complete a task. A task can be either a system or application unit of work. The process is constructed by assembling the code, routines and library functions in addressable storage. System primitives are a basic set of operators through which the Executive system will control the processes. Process control information is contained in a vector of state variables called the Process Control Block. Primitives are concatenated to form procedures which are "called" by processes.

TABLE 3. SYSTEM PRIMITIVES

NAME	OPERATIONAL FUNCTION
STOP	Stop the executing process until a WAKE occurs (one execution of its compute cycle has been completed).
WAKE	Prepare the specified process for one (additional) execution of its compute cycle.
WAIT	Discontinue the executing process until some prespecified event has occurred.
CONTINUE	The prespecified event has occurred; continue execution of the specified waiting process at the point of discontinuance.
DISPATCH	Start a specified processor in the execution of a specified process (associate a processor and process).
PREEMPT	Stop a specified processor in the execution of its associated process.
SUSPEND	Stop the specified processor in the execution of its associated process. Stop the specified process if it is executing; prevent execution until a corresponding RELEASE occurs. Do not disturb any existing processor/process logical association.
RELEASE	Return the specified process to the state it was in at the time of the most recent corresponding SUSPEND.
EXIT	Terminate the executing process.
ABORT	A specified process has failed. Terminate the process and initiate remedial action.

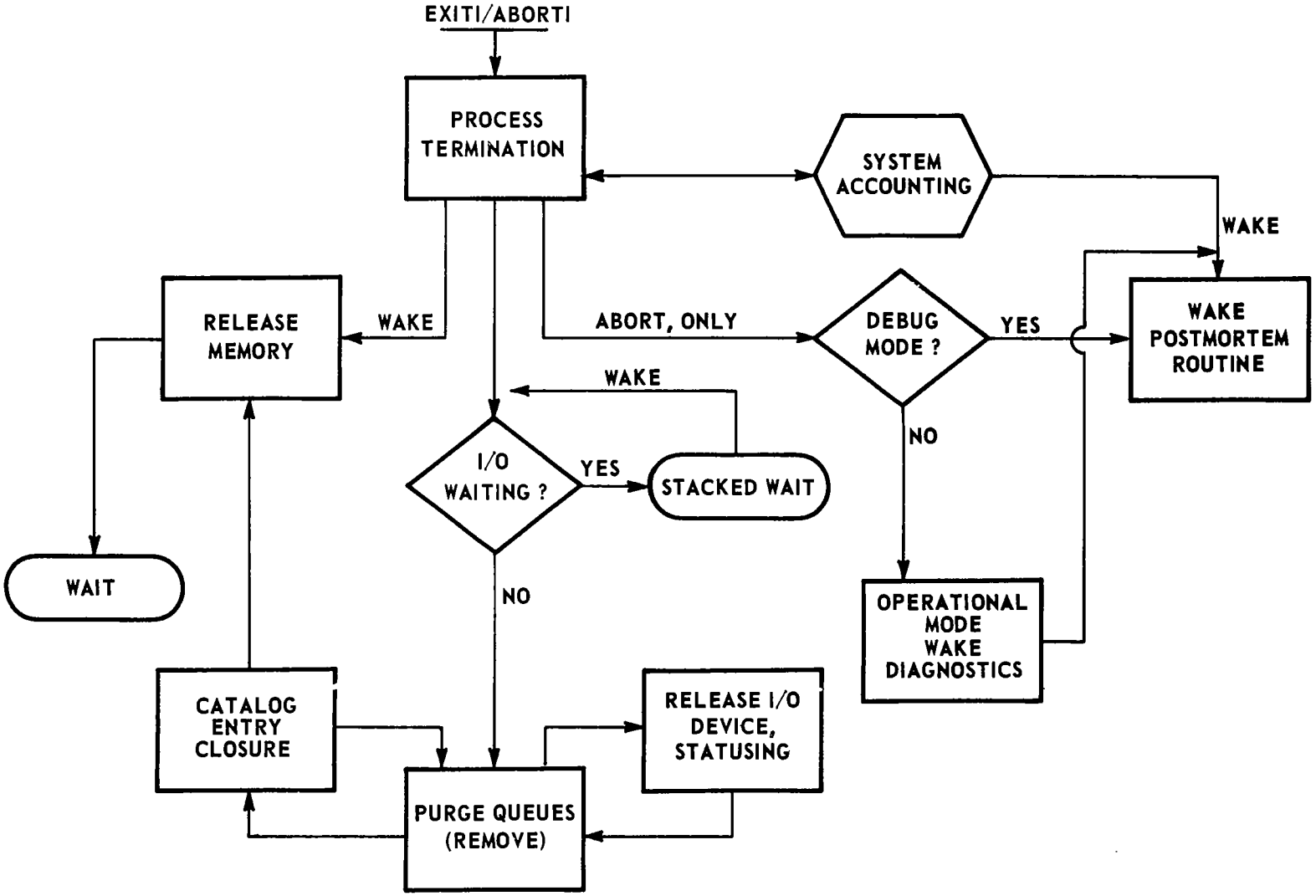

FIGURE 4. EXIT/ABORT PROCEDURES

SECTION IV. SPACE STATION EXECUTIVE

The Space Station Executive is that collection of algorithms that exercise computer control over computer resources. The "Executive" will be defined as a generic term referring to the Space Station computer control regardless of implementation in software, microcode, firmware, or hardware.

Description of the Executive as employing the advantages of a combination of software, microcode, firmware, and hardware requires a more precise definition of terms. It is readily apparent that tradeoffs must be conducted, in some predetailed-design stage, to identify and locate each Executive routine. The nucleus routines should receive extensive treatment (the nucleus is composed of all Executive routines that are not removed from main memory and especially those that provide interfacing, software/hardware or software/software). The design of the transient area for executing of routines infrequently executed should be investigated to determine the relative merits of an optimized-fixed area versus a dynamic area of memory.

There are advantages to implementation in hardware, software or hardware. For example, an algorithm implemented in software would be flexible (easily altered, visible). The same algorithm implemented through microcode or firmware would have different attributes. Microcoding operates faster, is more efficient, and its internal operations are transparent to the application programmer. However, microcode is less flexible than software since it requires higher skills for preparation, maintenance, and verification. Firmware is protected from change, but that protection incurs time and considerable expense when alterations are required. The same algorithm implemented in hardware could be more reliable and, certainly, invisible to all users. However, hardware implementations are inflexible and would require even longer time periods for changes. Hardware changes will ripple verification requirements through the entire hardware/software system. Microcoded and hardware implementations would produce, to the application programmer, a virtual "computation machine" with additional attributes above the basic system.

A study should be performed to identify the nucleus components and to fix their relative values for implementation in software/microcode/firmware/hardware. For example, complexity of development, probability of alteration, and reliability/validation requirements of each module would be traded against the attributes of software, hardware, read only store, and control store implementation. Frequency of utilization will be a driving determinant, and representative models of these demands can be utilized in several areas of investigation.

Within the present constraints of Space Station definition for the Information Management System (IMS) and the changes the experiments definition book is undergoing, the Executive will be approached from the long term view. A high-level, functional discussion relatively free from computer architecture specifics will be undertaken.

A high-level functional design will be described for Executive control of the computer complex. A natural focal point of interest is the Multiprocessors since they form the basis of the computing capacity, control and capabilities of the system. A thorough consideration must be given to intercomputer communication and data bus control since intuitively one feels that the additional distance of transmissions (timing and possibly noise) and complexity (more equipment-sharing resources) can produce additional problem areas.

A. Intercomputer Communications

Figure 5 illustrates the general flow of information among the primary computer systems. Each computer will have hardware and software (or microcode) to enhance communications among the computers. Modularity within software entities is an important factor in achieving cooperation and defining isolation among the separate computer systems conducting specific missions. The interface between modules can be a verification point prior to each transaction.

Several methods of intercomputer communication can be employed. Some basic methods are:

- Shared memory
- Mailbox
- Semaphore
- Sense lines and interrupts
- Timers
- Data bus messages.

Figure 5 illustrates that there are communication dedicated features of each computer. The configuration is not sufficiently firm to attempt to designate procedures between the system computers. Any of the above listed methods (and others) could be employed. Some of the more viable procedures will be discussed in detail. Executive hardware/software design will enhance the interactive capability of the system. It will also add to the protection, from failures or other spurious communications, of the individual computers.

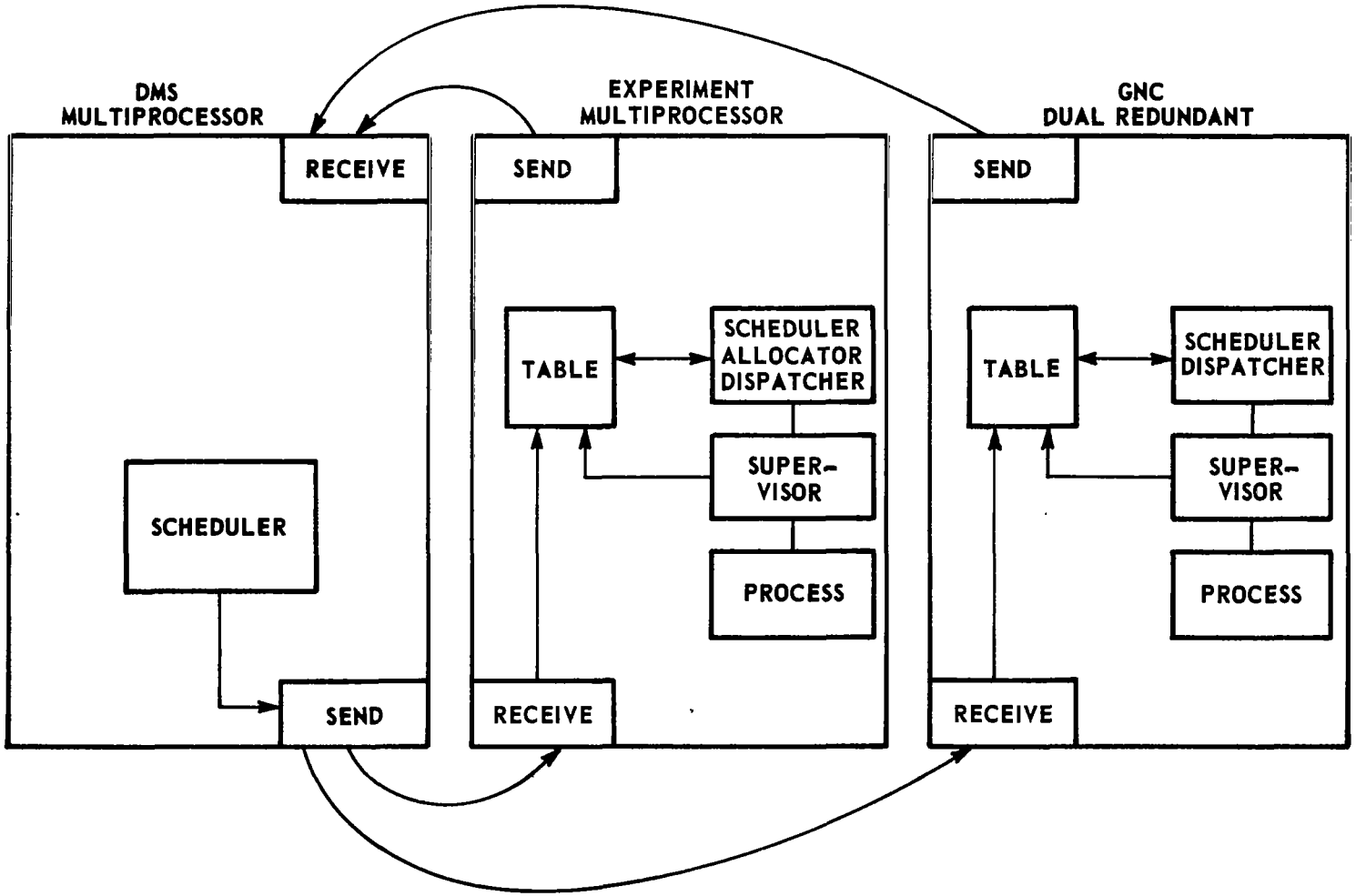

FIGURE 5. INTER-COMPUTER COMMUNICATION

B. Intracomputer Data Control

The data flow within the DMS multiprocessor will be complex. The flow among the Scheduler, Supervisor, and Data Bus Controller is illustrated in figure 6. There are major software interfaces at each juncture of data flow and each command and control point. These system units will be defined and discussed in some detail, but the overall relationship, illustrated here, is indicative of the information management system under control of the Executive.

The system is interactive. There is feedforward and feedback analogous to well understood industrial process control systems. There is sharing of information through the system data base. The Supervisor controls access to data and provides routines for manipulation of the data base. Each process makes use of the Supervisor's capabilities and thus has access to the data base, and, indirectly, to the Scheduler. The objective is to share the available resources among active processes, as defined by the Scheduler, and controlled by the Supervisor.

Can such an interactive system cause objectionable interaction or interference among processes? Just as in a large process control system, checks and counterchecks will be established at interfaces to prevent interference. For example, the Supervisor will maintain knowledge of every process, its status, requirements, and the resources available to fill those requirements. This is primarily accomplished through the Process Control Block described earlier and the system data base.

Procedures can be set up between modules to assure that data transmissions are accurate. Hardware trap instructions enable the detailed inspection of data to determine that the sending module has the proper credentials, that the data transmitted is complete and reasonable, and that the receiving module is prepared to accept the data. Additional verification checks can be made; however, many conventional tests are time-consuming and require additional resources -- developmental and operational. Procedures of this type cannot be recommended without an extensive analysis of need versus cost.

Further precautions can be taken through the implementation of requirements/utilization prediction and supervisory algorithms. Procedures can be proved to guarantee that individual problems (e.g., thrashing and deadlock) will not arise through scheduling. However, the value of such implementations must also be traded against their cost in overhead time, space, and side effects (such as restriction of throughput). It is possible that a thorough study and functional simulation would produce detailed design procedures for scheduling, data management, and resource allocation that will take advantage of the knowledge gained and eliminate the need for real-time implementation of the actual algorithms. These algorithms are basic to a detailed system design and not considered further here.

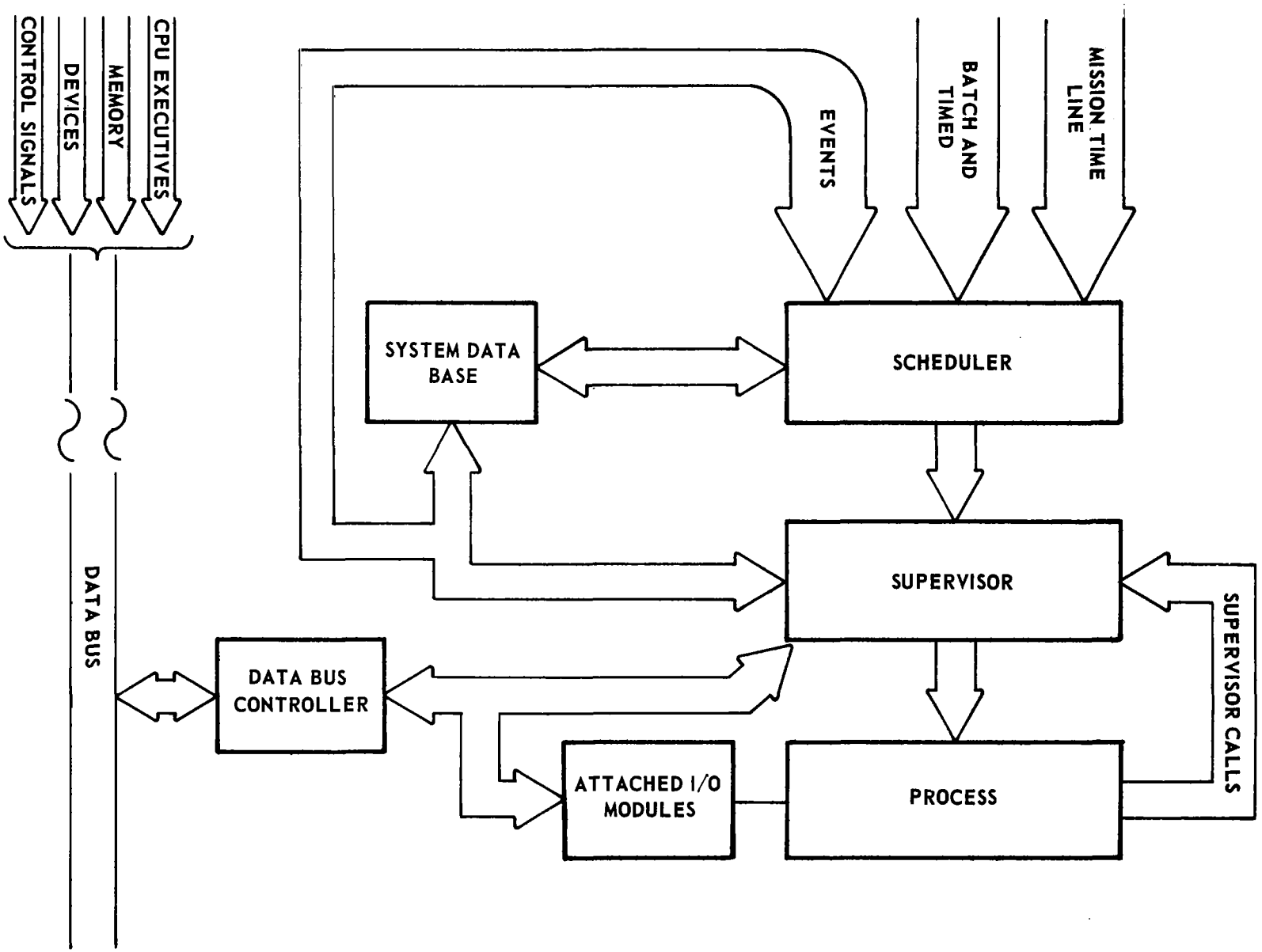

FIGURE 6. GENERAL SYSTEM INFORMATION FLOW

C. Scheduler

An algorithm for the scheduling of computer resources (figure 7) is dependent upon (1) the hardware configuration, (2) attributes of the input process stream, and (3) the goals to be achieved. We have described a multiprocessor-multicomputer configuration environment. Detailed assumptions such as memory organization, paging drums, and addressing algorithms have not been made. To make such assumptions would permit a greater level of design detail, but would negate the long term value of the design. However, a functional scheduler design can be pursued, bringing out some of the salient characteristics of schedulers in general, and the Space Station in particular.

The attributes of the process stream is the subject of study not yet available. Intelligent estimates of the job mix is a driving basic design requirement. Some specific attributes will be speculated upon with relative assurance. For example, certain time-sharing characteristics will be required to service interactive consoles. The time-sharing load should have relatively minor impact on overall design, compared to the functional requirement for the time-sharing capability itself. There will be considerable real-time servicing of experiment equipment. There will be stationkeeping, time corrections, and other periodic calculations. Large amounts of data are expected to be handled at random intervals requiring correlation and other large core requirement application processes. The process stream should be simulated and evaluated in order to determine an adequate means of scheduling IMS resources. The validation of the IMS begins here with the proper support of detailed design.

The goals of the system are to service real-time requirements and to provide an environment in which the application processes, as illustrated in figure 6, can perform to their specifications as required. It appears that a study of the process stream will show that a large number of jobs requiring a small to moderate amount of ALU to actual time ratio will be related to several large but less frequent jobs requiring large amounts of ALU to actual time ratio. The assumption of a particular mix of ratios is not necessary at the functional level, but the extremes are recognized and acknowledged.

Scheduling of processes undertakes the division of available ALU time and other IMS resources among the tasks that are to be accomplished. To be considered for ALU time, a process must be "ready" (as defined earlier). The code has been loaded and the process control block shows all conditions for "run state" satisfied. Memory configuration (segmentation, paging, ports, phasing, addressing algorithms, etc.) plays an important role in determining how often processes should be loaded which defines swapping constraints. It also helps to clarify a set of potential problems, including thrashing and deadlock, which will be avoided. The protection algorithms mentioned earlier are examples of methods for addressing these classes of problems.

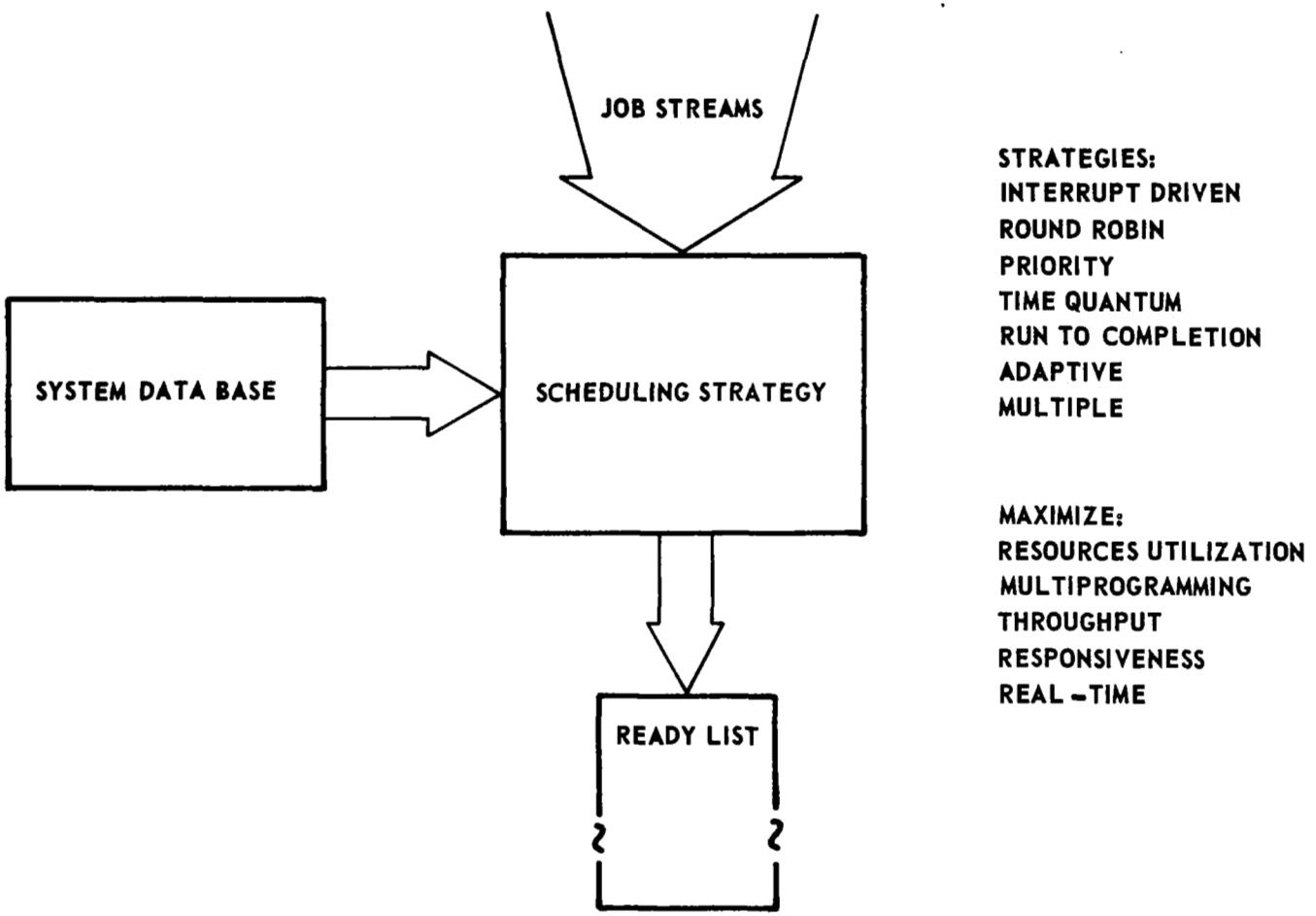

FIGURE 7. PERSPECTIVE OF GENERAL SCHEDULING STRATEGIES

A scheduling procedure could be followed such that "run state" eligible programs generate the scheduling algorithm (adaptive scheduling) based on expected running time, original priority, deadline, waiting time, number of stacked wakes, memory management, and device utilization. Switching of ALUs from process to process will require bookkeeping and data-save functions (much of which is retained in the Process Control Block). This will consume considerable time if ALUs are switched on every change that produces a higher priority. Therefore, once a process has begun execution, it should receive a minimum amount of ALU time. This minimum time may satisfy many requirements and thus reduce scheduling overhead. A minimum or optimum time quantum can be determined through experimentation utilizing simulation.

The ready list contains processes in various states. Those processes that are eligible for ALU time (competing for the "run state") form a switch list. That is to say that the available ALU time is divided, or switched, among them according to some predefined procedure. An indivisible amount of ALU time is referred to as a time quantum, and for the DMS is a basic resource.

The switch list queue control routine will be implemented to serve each process, according to an algorithm with one or more time quantums. Here again, simulation can be very valuable following definition of the process data stream. For example, processes with large I/O requirements can be given larger time slices since they will relinquish the ALU during I/O operations. It would be unfortunate to provide a small time slice to such a process and have it surrendering the ALU frequently and thus increasing overhead. By the same token, the amount of list processing time taken referencing procedures that have to be loaded or that require other time-consuming actions could have a negative effect on the time quantum. Much of this environment can be improved by controlling the production of application software to perform at maximum interactive efficiency with the derived scheduling procedures.

The scheduling procedure could be adaptive (a single algorithm with many parameters) or, as illustrated in figure 8, a specific algorithmic design for each of the computer subsystems to be scheduled. This is directly applicable since the scheduling or preparation of ready lists has been specified for the DMS multiprocessor but will be dispatched and executed on computers that are unique and separate entities.

A separate scheduling strategy for each computer can be implemented on either the DMS or from DMS commands to individual computers. Separate strategies are modular, which enhances flexibility. An individual algorithm can be changed as required without seriously impacting the rest of the system. The same holds true for the dispatching functions since they are resident for each respective computer system. A different dispatching procedure might be

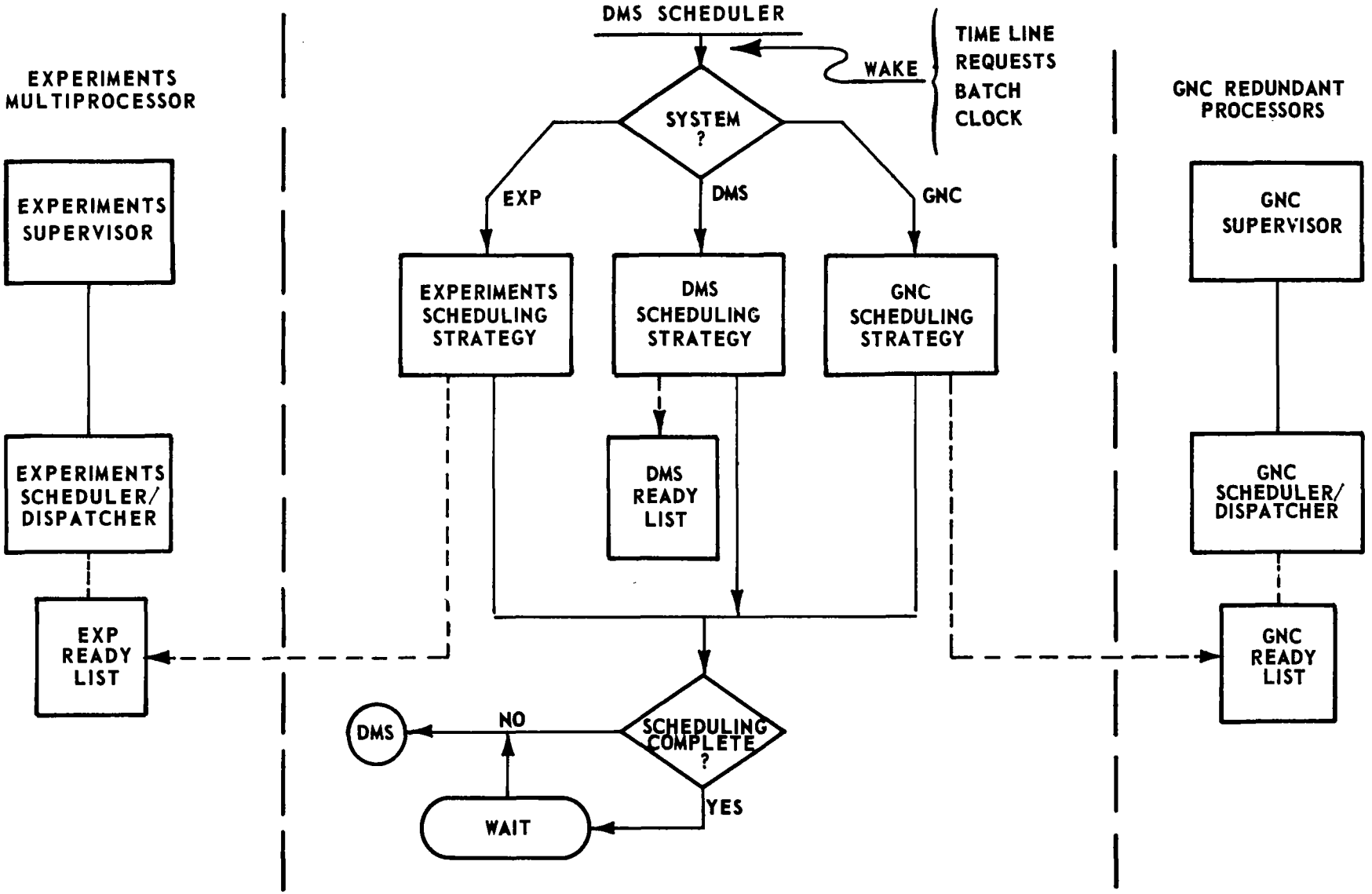

FIGURE 8. INDEPENDENT ALGORITHMIC SCHEDULING

suitable for the experiments (large volumes of acquired data, investigator analysis) and the GNC (extensive calculations, perhaps several control signals) at various times.

The scheduling algorithms are the driving force of the Executive system. They control the extent of multiprogramming, the time processes start and complete, and the memory activity. Memory activity greatly affects throughput. The DMS Scheduler (DMSS) will support multiprogramming and multiprocessing on the DMS computer and the Experiments computer. There will be separate algorithms for each of the multiprocessors and for the GNC. The Biomedical computer is a stand-alone system.

An alternate approach would be one algorithm for the total IMS. Each processor would then be treated as a multiattribute resource. This paradigm could be coded to require less main memory. However, it would require greater running time and be less flexible than the separate algorithm approach. The most stringent requirement would be that the DMS computer perform all allocations and maintain full knowledge of every system's device conditions. It would also require increased validation time for each change.

The process scheduling algorithm for the DMS will be more powerful and adaptive than the similar versions for the other computers. The DMS has a wider range of demands placed upon it and thus requires greater flexibility (e.g., timeline, data bus control, program preparation, and astronaut intervention). The Experiments and GNC operations are more stylized and firmer planning can be implemented.

As shown in figure 6, the scheduling algorithms are the decision makers. The Supervisor is the interface with the processes and all system devices. The Supervisor maintains the system data bases. The Scheduler, then, must generate a new schedule based on the requirements at that instant. It is not required to seek or give any information. It must only update the ready list which is a part of the system data base.

The ready list is a table that describes each of the processes that have requested resources. Assuming that the nucleus of the Executive is always "ready," unless in execution, requires that the ready list contain the status of code, the processor (ALU) assigned, or the capability for concurrent execution. Each process will have codes representing the resources required and the status toward acquiring them. Several algorithms will require determination through further study and simulation. For example, priority and priority changes, time quantum and its variance, and the allocation/deallocation of resources should be studied in detail.

A ready list need not consist of "ready for run state" processes, but rather all those processes that make up a manageable segment of the job stream. The individual process descriptors contain codes indicating that they are ready for ALU time and their relative priority for that Time. Sometimes referred to as a "switch list," these ready processes acquire an ALU (run state) via a dispatching algorithm. The switch list processes are maintained in a contiguous segment of the ready list so that the dispatcher does not search processes that are not prepared for the "ready" state.

Several studies have been suggested above which will determine parameters for the algorithms dispatching the scheduled processes. These will differ for each of the computers due to variations in time-line, demands, and transients (components off-line for maintenance, failures, etc.).

1. DMS Scheduling Strategy. The scheduling procedure is illustrated in figure 9 for the DMS multiprocessor. This is also illustrative of the Experiment scheduling since, functionally, the DMS and Experiment computers are very similar. The Scheduler is entered when a new process requests ALU time or when resources are returned to the system. Therefore, if a new process causes entry, it precludes a queued process that might be satisfied by the available resources. Requests for resources are matched against the available resources. Requests are queued for busy resources, and the process is added to the ready list as "not ready."

Returned resources are another interesting area for study. The algorithm for assigning the resource among several requests of similar priority can be as simple as first-in first-out, but there are better methods. For example, the process whose completion will release the most resources (or utilize the most resources) can be determined, but what would be the result of implementing such constraints? The returned resource might be assigned to the process that will complete the fastest and re-release or deallocate the resource quickly. Is it cost effective to "look ahead" and see what resources are delaying the ready list?

"Current Situation" is an algorithm to continually evaluate the multiprogramming and multiprocessing effectiveness. Samplings from Current Situation should be regularly, on demand, and at critical points (filling queues, low multiprogramming, etc.) be sent to the housekeeping routine for future analysis. A real-time feedback to the scheduling algorithm is a realistic prospect, also. These are resources or indicators of resource utilization.

If the requests of a process can be satisfied, it joins the ready list to compete for ALU time. A process control block is constructed and entered into memory (protected storage is recommended). Its queue representation is added to the ready or switch list.

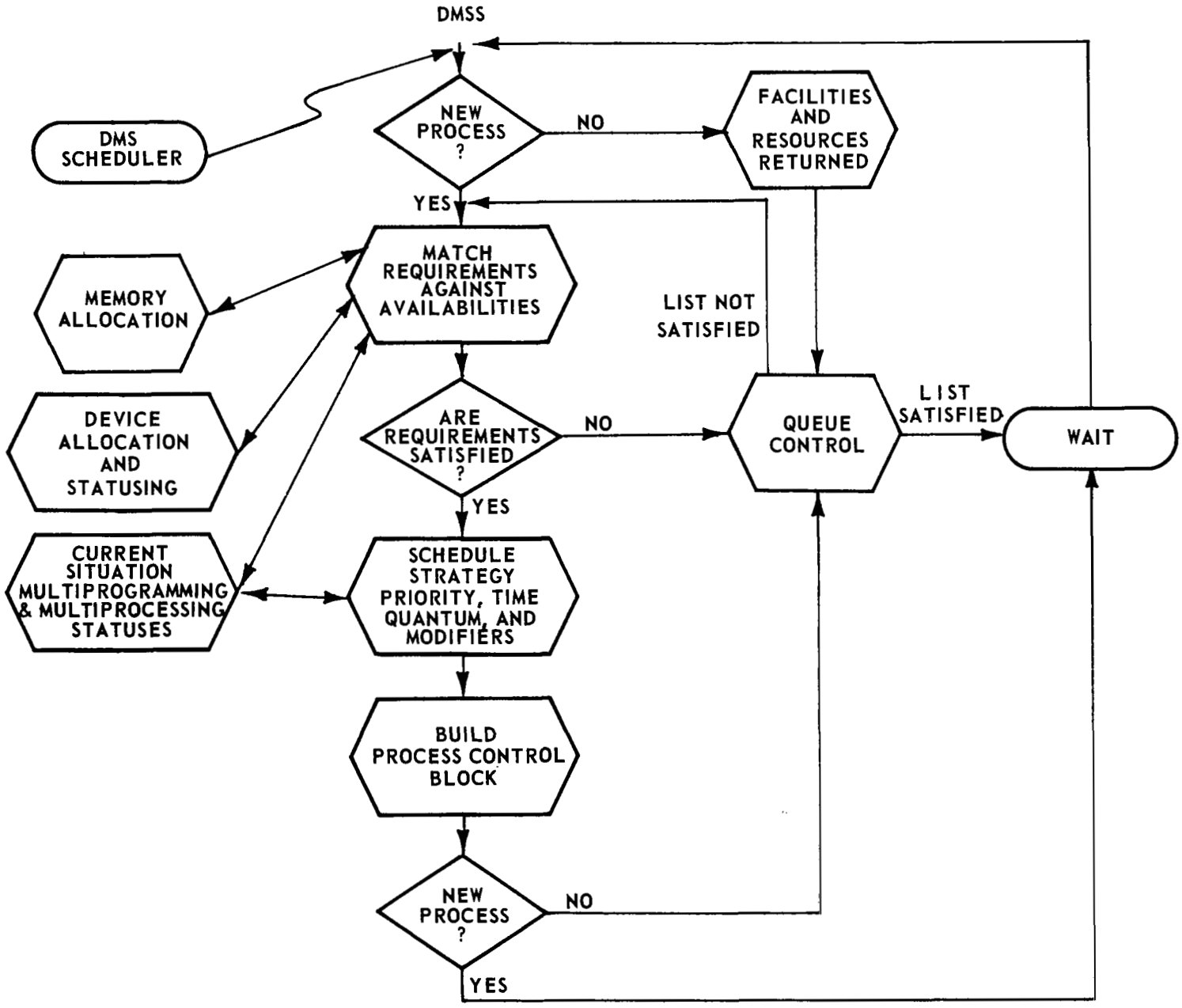

FIGURE 9. DATA MANAGEMENT COMPUTER SCHEDULING

The algorithms described could be primitives at the system design level. Device Allocation, for example, might be implemented in either read only storage or microcode and have attributes similar to a macro. The Scheduler referencing the Allocator would identify, through a parameter list, the device requested, priority of the requesting process, etc. Reentrant routines would reduce the coding required for the allocation procedures. However, it would increase communications requirements.

Modularity should not be less than that indicated in figure 9 for ease of change and validation. Greater modularity will increase overhead but may be required at individual device types, access methods, allocation algorithms, and queue management.

Memory Allocation is illustrated in figure 10. This points out the necessity, not only for modularity, but for flexibility within and among modules. The allocation of storage is basically the same for any requirement: Is a device available and a path to that device; is there sufficient storage (logical, physical, contiguous, etc.); does the requesting process's priority (relative to active processes) justify more drastic steps?

If the storage is available, it can be assigned. If not available, it may be either queued to compete for released storage or other satisfaction may be sought. For example, a fragmented memory may (possibly) be collapsed to produce a contiguous, useful area or a portion of memory may be rolled out onto mass storage. The tradeoff is twofold: capability vs. the expense of overhead plus extra developmental expense. These determinations can be made upon a basis of configuration, job stream, system demand evaluation, and environmental simulators.

A sample storage map and its implications are illustrated in figure 11. Various types of protection can be built into such control. For example, a method of sharing a memory segment among processes (the shaded memory area) is shown. Protection locks that require specific keys for access are indicated. Shared memory is basic to reentrant Executive subprograms. Depending upon particular design, shared memory can be assigned at load-time or dynamically.

The user name actually exists only in a reference catalog. The system will assign a nonconflicting, error resistant process identifier (PROCESSID) to each name for internal use. This will insure against interactive failures should a dropped bit or other name changing failure occur.

The structure of memory hierarchy and addressing is a driving feature in the overall system design. There are many designs in use (ground based) today and more are described as developmental in the computing literature. These schemes should receive extensive simulation and analysis prior to any detailed design. Such basic features are resistant to change after baselining.

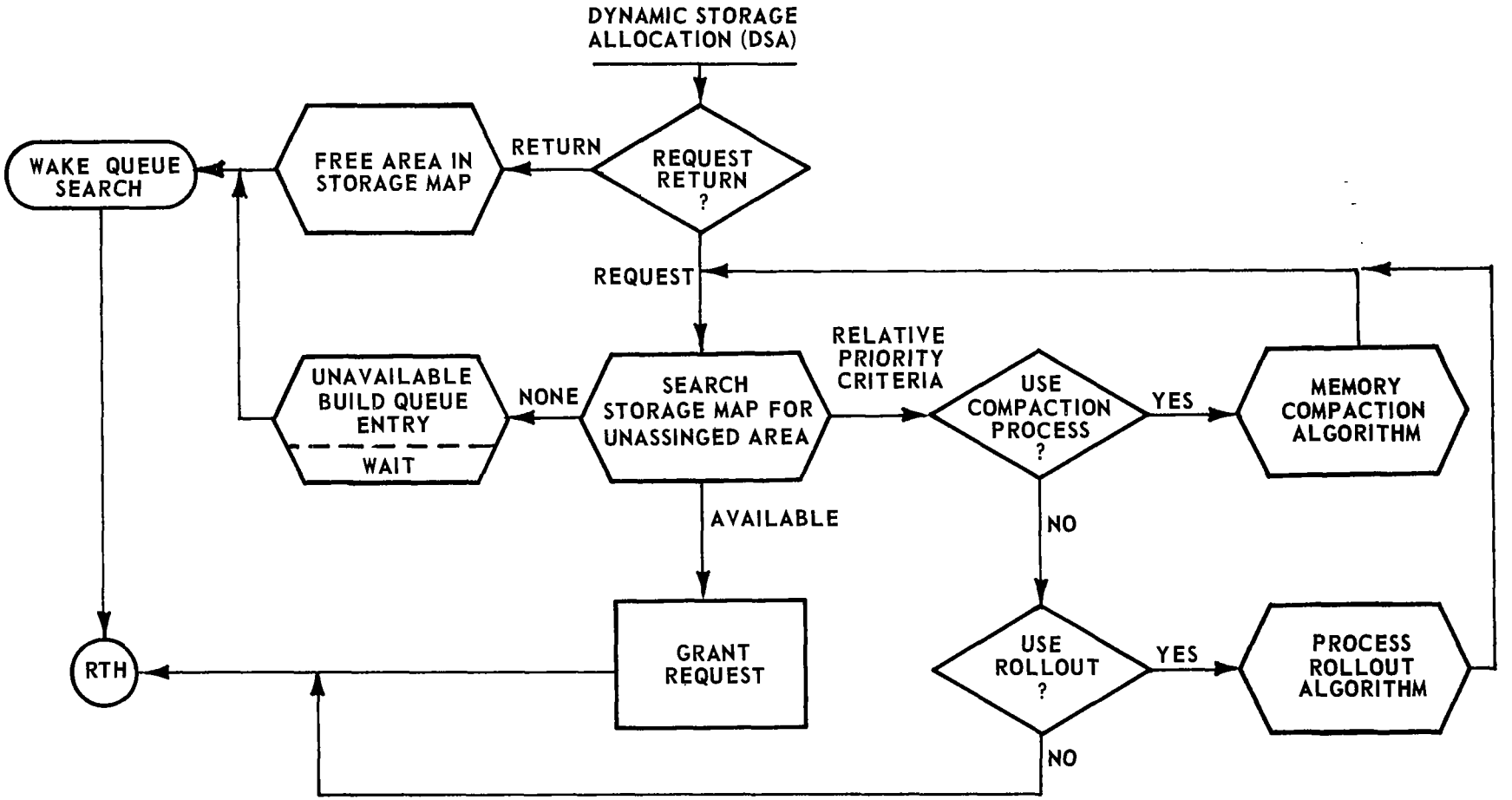

FIGURE 10. DYNAMIC ALLOCATOR

USER NAME	STORE ID	PRIORITY	STATUS	PROTECT KEYS	SHARE KEYS	POINTER	
						FROM	TO
1	37	A	RUN	410	1,3	2000	4750
2	21	B	WAIT	300	0	0	2000
3	19	A	RUN	065	1,3	4000	6000
4	36	C	STOP	723	0		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

STORAGE MAP

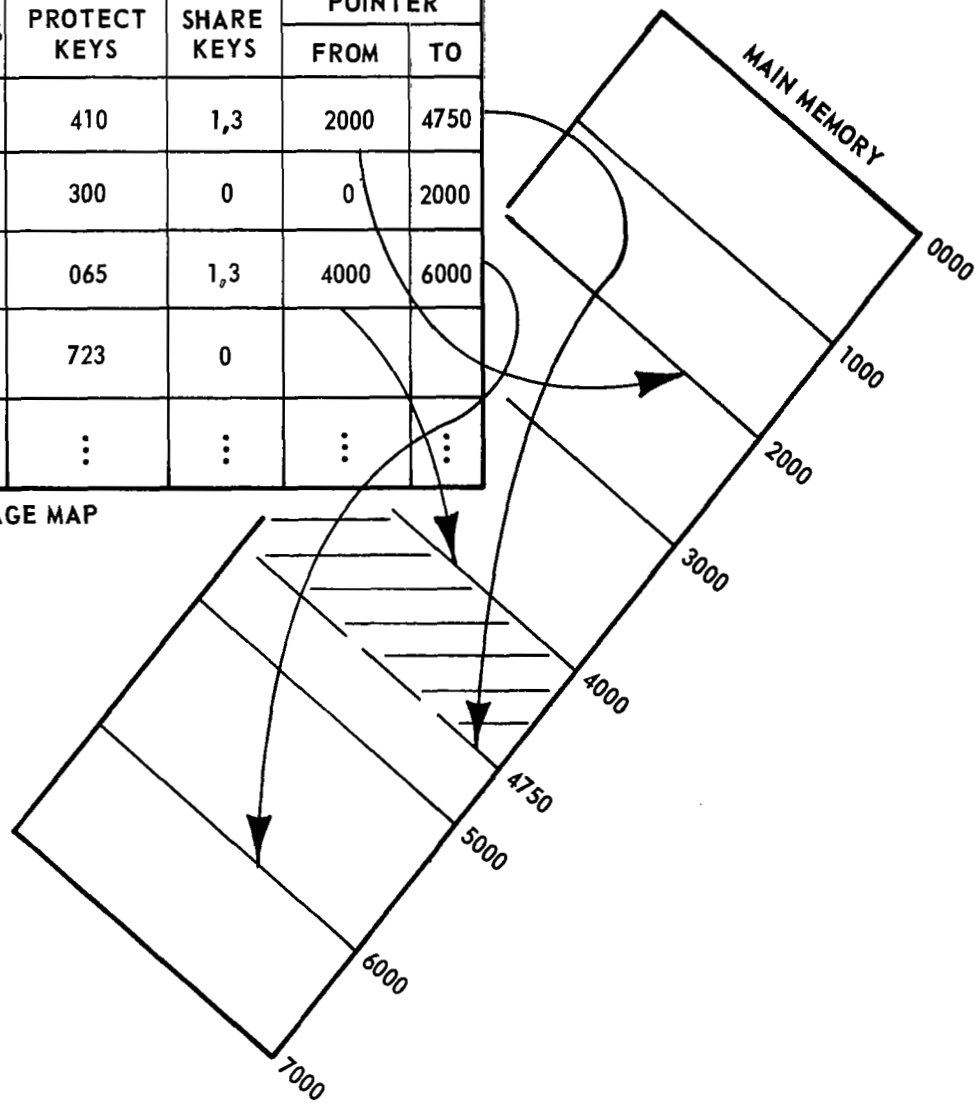

FIGURE 11. SAMPLE STORAGE MAP

A standard interface format is suggested among modules to improve detailed design, programming, and test. This is illustrated in figure 12. A standard format has been defined or assumed for the process control block, the timeline, and other essential and pervasive modules. There will be extensive communication both within and between the computer systems. Therefore, the development of standards and formats is necessary. Figure 12 is intended to illustrate one such implementation. All requests and deallocation returns of memory are made via the Storage Communication Format. The format contains positions for any information required in a storage transaction and fully supplies required queuing information.

The Memory Request Queue (described earlier) is depicted as a functional collection of such communications that are waiting for free memory and completes the information loop for the format and the request queue. For example, as shown in figure 10, if USER #1 completes processing, a communication format is generated and transmitted to return its resources, but the portion of memory shared with active USER #3 will not be released.

The system level primitives (or macros) will use straightforward communication formats and queueing arrangements to maintain a viable Executive. The many interfaces, hardware/software and software/software, will benefit from simplicity of design in this area. The extent to which standardization should be carried requires further study.

The difference in DMS and Experiment scheduling is represented by the "Scheduling Strategy" and "Scheduler/Dispatcher" blocks of figure 8. The differences exist due to the nature of determinability inherent in each system.

The DMS Scheduling Strategy requires flexibility, response to frequent interruption, and a "flat structure" arrangement. Flat structure refers to the capability of responding at low levels to changing requirements for all systems (DMS, Experiment, GNC). The interactive consoles and display devices suggest man/machine interaction, timesharing, program preparation/modification. The requirements for managing the Data Bus Controller and the interactive interruption process suggest a flexible system that is relatively free to adjust to changing time constraints.

2. Experiment Scheduling. The Experiments Scheduling Strategy suggests a free-standing laboratory environment. The experiments appear (except for rare exceptions) prescheduled on the timeline. Prescheduled here refers to the time-skills-performance analysis that application programs will supply. The experiments to be conducted will be well defined and must be scheduled in accordance with the location and attitude of the Space Station, the availability of equipment and crew skills, and within the capabilities of the overall system.

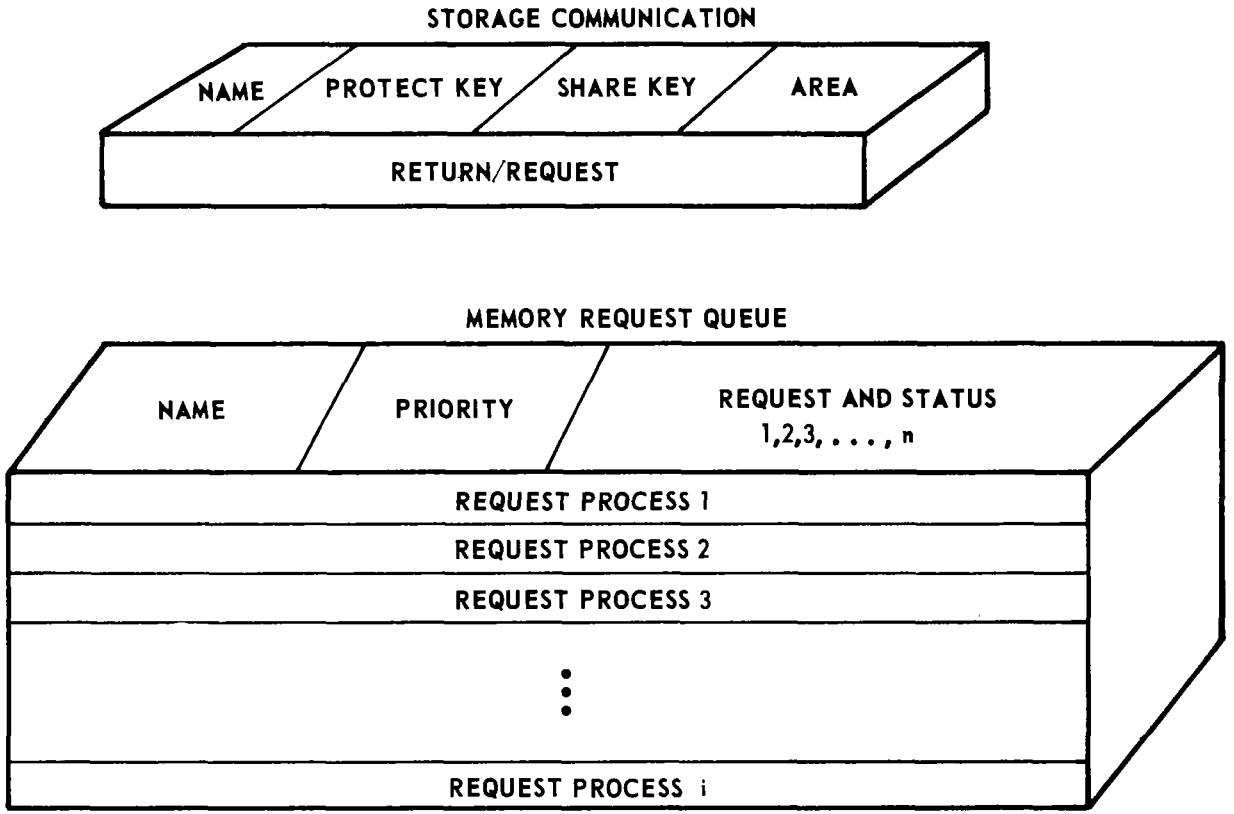

FIGURE 12. STANDARD STORAGE INTERFACE

The experiments scheduler is a more highly structured and, compared to the DMS, less flexible system. The prime scheduling requirements include data acquisition and analysis procedures (calibration, signal conditioning, multiplexing, conversion to engineering units, smoothing-filtering, trending, garbage attrition, compaction, display, storage, and retrieval). The Experiments Multiprocessor receives a message from the DMS that initiates the scheduling procedure (figure 8). The scheduled experiments then call for the more detailed functional routines as they are required.

The DMS system creates a scheduling request according to its criteria. The scheduled experiment then becomes an experiment supervisor which causes the Experiments Multiprocessor's resident scheduler to complete the schedule table. The table contains the sequences of states and attributes that the various processes which support the experiment can assume. This scheduling strategy should provide the flexibility of servicing data acquisition and real-time controls with a minimum of Executive overhead time. The data processing functions should tend toward a run-to-completion schedule with dispatching that favors I/O bound processes. Favoring I/O bound processes will help to maximize multiprogramming.

3. GNC Scheduling Strategy. Processes initiated by the DMS computer for the GNC Scheduling strategy have three primary types of functions to perform. These are continuous, periodic and interrupt processes. Continuous is used in the sense of essentially unconstrained as to completion time. Status monitoring, testing, and self-test are examples. These programs and lengthy calculation programs that can be initiated long before results are required can be run in the background of a foreground/background system.

Periodic programs, such as attitude computations, will have varying periods according to the position on the mission timeline. For the Space Station, some minimum period (perhaps 100 msec) would suffice to provide the reference base for antenna and telescope pointing and general attitude control. A much more frequent measurement might be required during docking maneuvers.

The periodic programs require data that must be acquired and may generate control values that are immediate commands and, thus, the requirement for interrupt processes. Interrupt routines are often a part of the operating Executive software. In this case, they may be part of the Executive and always in core or scheduled for loading in anticipation of possible requirement due to scheduling of periodic processes. In any event, they are executed as foreground processes.

The GNC Scheduling Strategy (as implemented on the DMS computer) is to provide a list of processes to be run, foreground/background designator, and a begin execution time. These items are defined by the maneuver to be executed as described by the timeline. The GNC Scheduler then conducts scheduling and resource allocation.

There is a large body of knowledge to be drawn upon to describe the GNC scheduling requirements. Though there may have been some less than optimum designs, there is much to be learned in the sizing, timing, and requirements areas of these efforts. The essential differences are the synchronization of ALUs and the input/output control (external to the DMS rather than internal).

D. Dispatching

Scheduling discussions have required frequent reference to, and definition of, dispatching in the IMS. The dispatcher function is dependent upon a method of gaining control of an ALU with which to implement an algorithm that describes the sharing of ALU resources.

1. DMS Dispatching. The DMS system will operate on a time-switched basis in order to provide the required timesharing capabilities for computations, program preparation, and interactive analysis, as illustrated in figure 13. If an interrupt system is not specified (polling is discussed later), the interval timer will have to be well within the response required by the system. Buffering, liberally applied, can ease this constraint.

The dispatching process is quite simple; select, according to some criteria, the next process to receive ALU time; set the interrupt timer for the amount of time to be given; initialize the process from the process control block; set the program address register to the return address saved in the process control block. The procedure is illustrated in figure 13, and primarily represents references to the Ready List, Process Control Blocks, and Supervisor Calls.

2. Experiments Dispatching. The experiments are dispatched either immediately, by time of day, or by time interval. Since many experiments are dependent upon time-position, the dispatching would be expected to be on a time of day basis, which could be represented in some other time frame. The scheduler will have taken into account the position/attitude reference taken from the output of the GNC. This dispatching can be similar to the DMS Dispatcher. However, more of a table-driven dispatching design is recommended. The experiments are greatly dependent upon data acquisition which prefers a run to completion or interrupt environment. Thus, individual experiment supervisors will time-event conditionally release an ALU. The scheduling strategy will resolve key differences by assigning the process within the ALU's Switch List with appropriate priority.

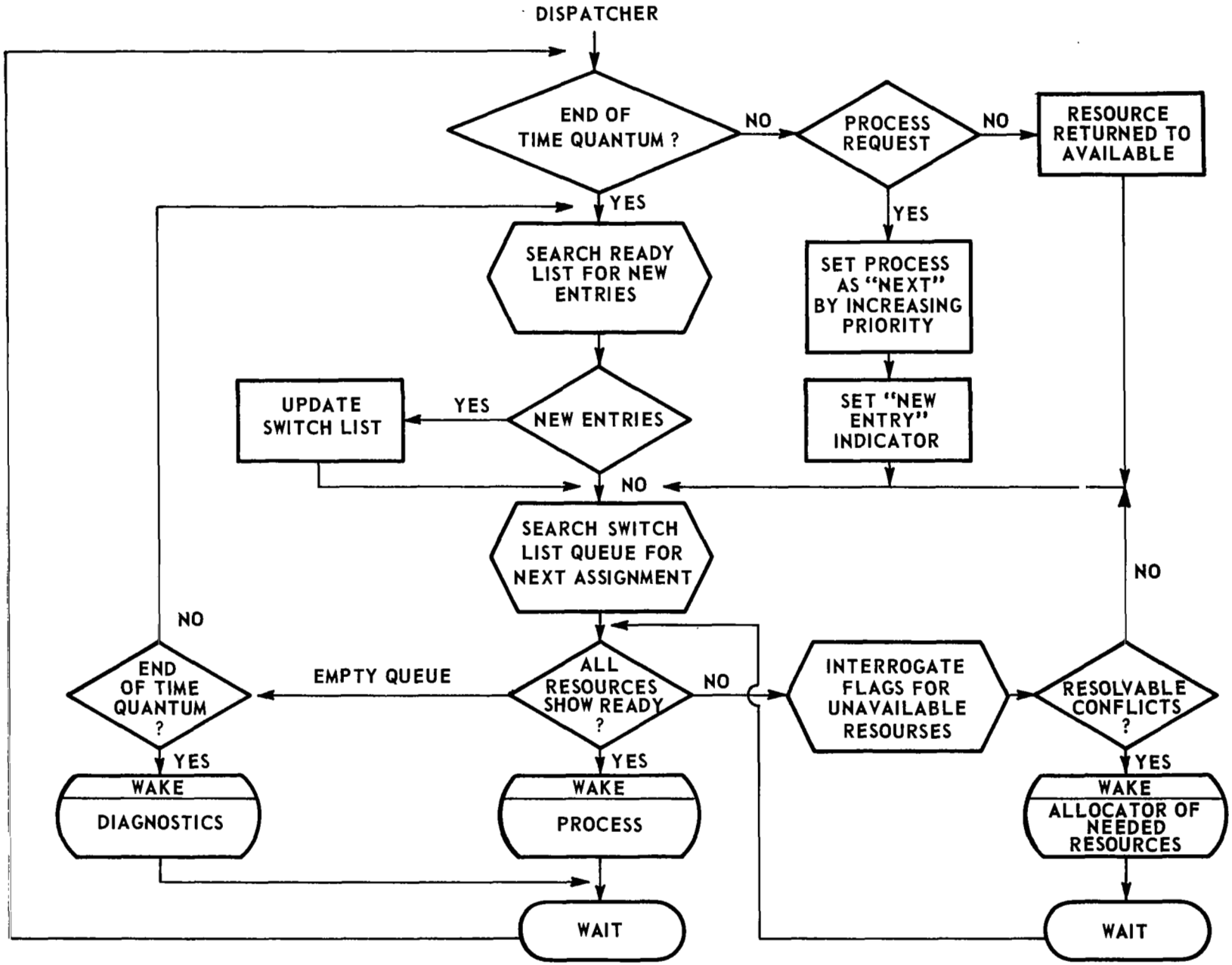

FIGURE 13. DMS DISPATCHER

Dispatching, then, is primarily the proper initialization of processes after which they become essentially event driven. An event would be the acquisition of equipment (telescope), the required function (pointing) and initiation of data transfer. The terminal condition would be passing out of a field of view, the initiation of higher priority experiments requesting the same resources, or the expiration of a predefined time interval. The algorithm will be one that shares the processing power to maximize multiprogramming but faces the reality of minimizing overhead and seizure of the ALU during high computation periods.

3. GNC Dispatching. There will not be a complex Dispatcher for what will operate as a single ALU system. From the previous assumptions (Section II. Configuration) and the discussion of scheduling procedures, a synchronizing function will be important. This assumes the requisite hardware for stopping and starting three ALUs to make their operation essentially identical.

A foreground/background event driven system from the schedule table better describes the work in progress function of the GNC Dispatcher. A great deal of further definition can be accomplished in this area based on firm operational requirements. The primary function of the Dispatcher is then supporting a multiprogramming environment.

E. Supervisor

The basic modules and interfaces that are described or alluded to in this report are brought together in figure 14. Many of the modules listed are self-explanatory and may fall into the applications area. For example, "Time Initialize, WWV Correction" obviously refers to the maintenance of system time from a transmitted earth time, taking propagation and reception delays into account. Other modules may be system supported but application implemented.

1. Supervisor Call. Figure 14 indicates that an interrupt type procedure is used to interface the software processes to the Supervisor Call (SVC) Interpreter. The Supervisor Call Processor's relationship to the processes is illustrated in figure 15. There is interaction directly with the process and directly with the Process Control Block. Even though the process is linked to the Control Block, the process must go through a SVC to alter the Control Block. The Interpreter will perform a check of the Process Control Block to determine the validity of the request. If the request is not valid, the operation is in error and the ABORT primitive is executed to terminate the process.

A full range of supervisor services (such as utilities, communication, storage management, and device handlers) is available through SVCs to authorized users. The various tasks and buffers that are attached to a process are

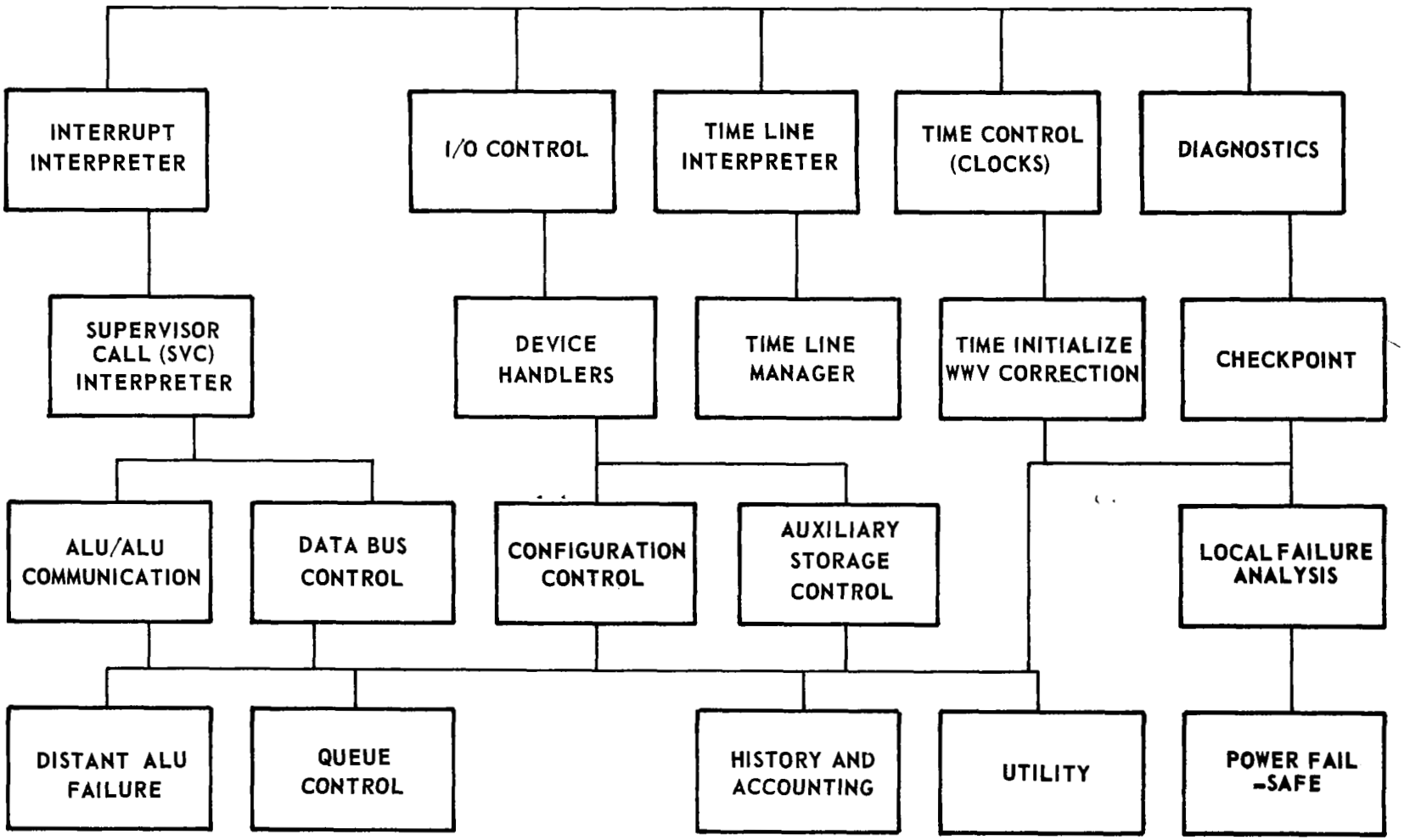

FIGURE 14. AIRBORNE EXECUTIVE BASIC SUPERVISOR MODULES

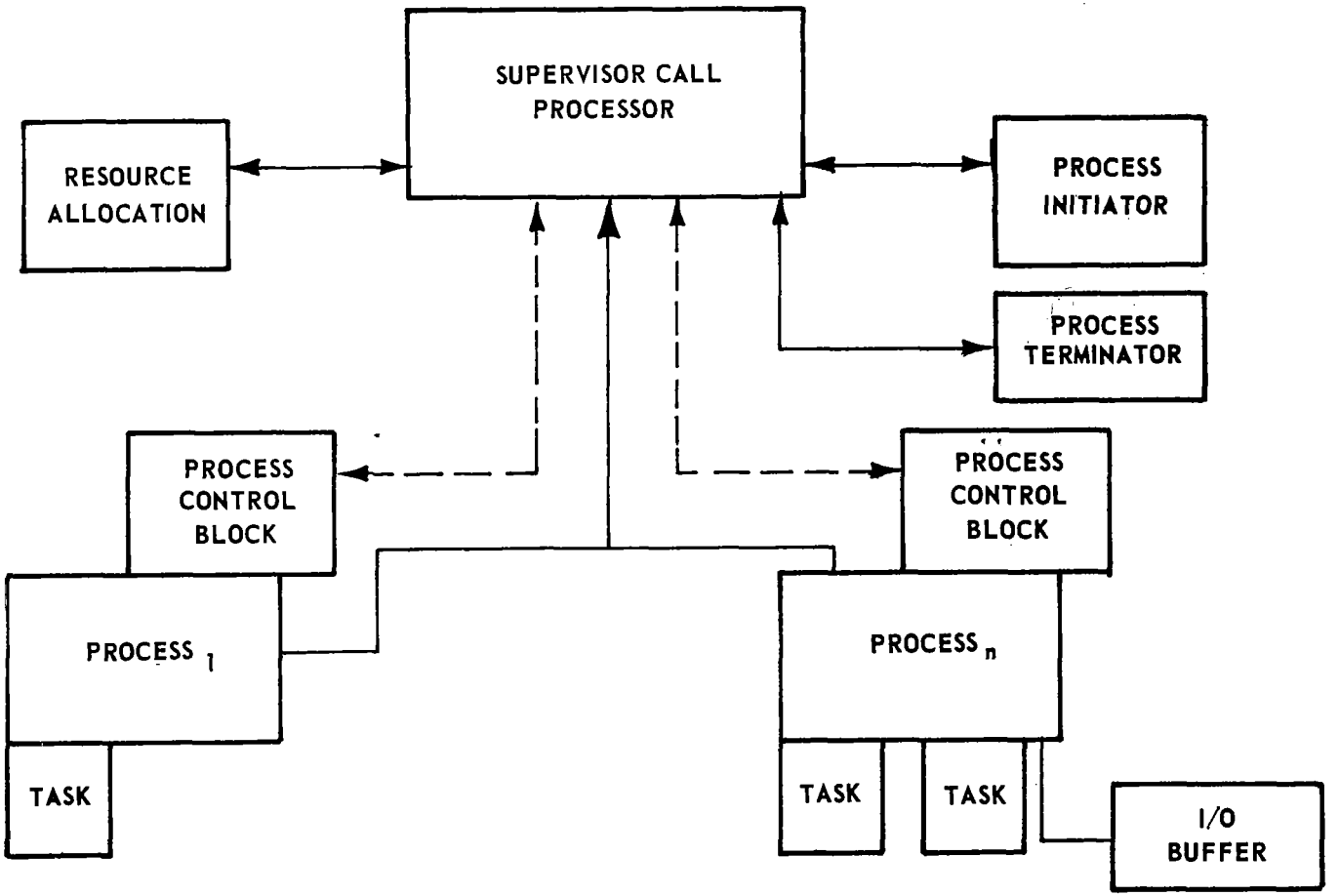

FIGURE 15. SUPERVISOR CALLS

represented in the Process Control Block (PCB). Since the PCB is in protected storage and is manipulated via privileged instructions, and is not readily subject to destruction, an interface control mechanism is established. The opportunity for an accidental supervisor function initiation is remote.

Through SVCs the process can initiate other process executions. These executions may be sequential or parallel as defined by the type of call. The process may wait or proceed (to a point) also dependent upon the call. This flexibility is provided through utilization of the System Primitives.

Resources can be requested (allocation) or released (deallocated) by SVCs. For example, the I/O buffer attached to $Process_n$ (see figure 15) can be returned when it is no longer needed. Main memory or auxiliary device space will be defined by the total mission requirement, configuration design, and subsystem interfaces. System traps will be provided to get memory, release memory, and perform other user requested functions.

2. Input Job Processor. The timeline (or other entry) will cause the initiation of a job which requires that a process control block be constructed and that all of the required factors be available for "ready" condition. A major function of the input job processing is the interpretation of the job control header. The job control header contains the coded attributes of the environment within which the job can become a ready process as illustrated in figure 16

When a process is added to the Ready List, it is then available to the Scheduler which will give it entry to the Switch List. In the Switch List, it will compete for ALU time with other processes. The Switch List contains only those processes that are eligible and prepared to use ALU time. If a process cannot be activated, it goes back to the Ready List until such time as the preventive indicator is removed.

3. Timeline Control. The timeline will originate from three or more sources. A predefined timeline will be prepared prior to orbit, uplink timelines will be received, and the onboard team will require the ability to alter the timeline (see figure 17). The assumption made here is that the timeline prepared on the ground will exist (in orbit) in a useful language that can be translated into discrete time-ordered events which will be used to WAKE tasks and to RELEASE suspended processes.

The maintenance of a high-level language timeline will require an encoding process to build the discrete event stream which must be decoded into timer intervals. However, application programs will be required to permit the onboard team to review the coming shift's (or any interval's) timed procedures. Further software will be provided to permit changes in the timeline by manual

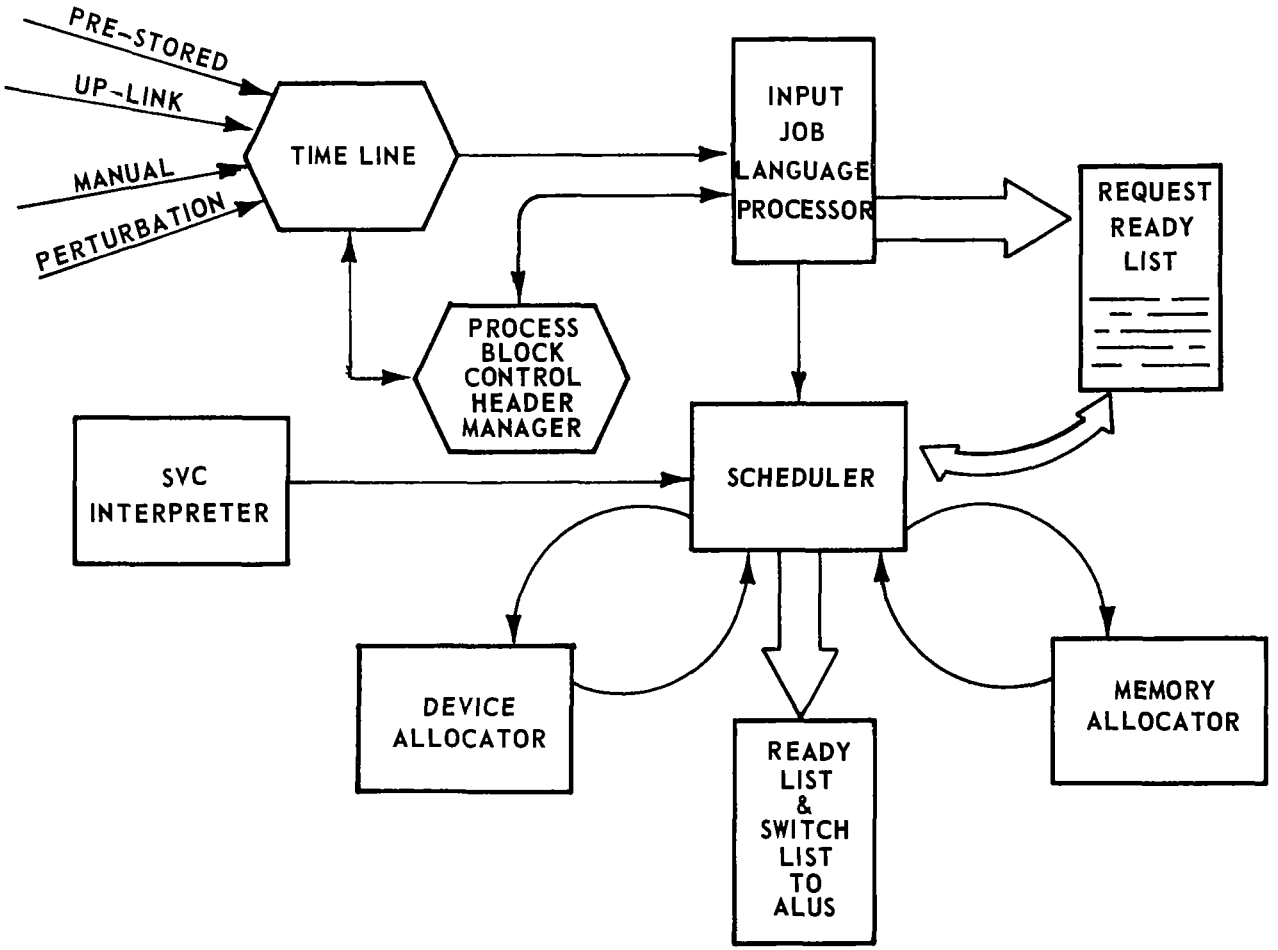

FIGURE 16. INPUT JOB PROCESSING

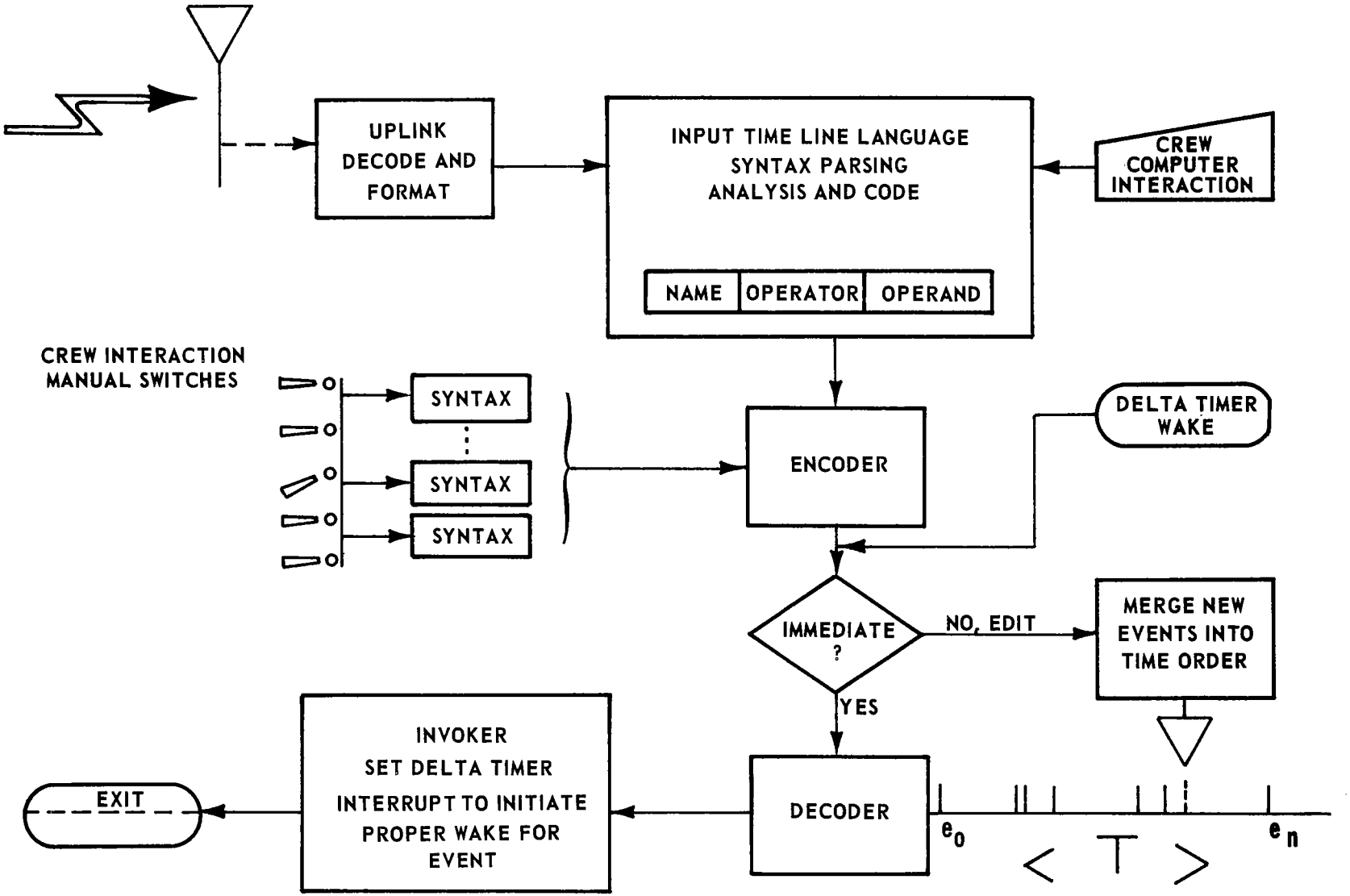

FIGURE 17. TIME LINE CONTROL

switch for predefined conditions and by input of the timeline language through one of the console input devices. The highest level application program would be of the sort that would accept the station reports (change of personnel, equipment status changes, crew requests) and generate a set of nearly optimum timelines. These would be displayed to the crew commander in various forms to illustrate the impacts upon crew assignments, utilization of expendables, and achieving maximum results from the experiments.

Modification to the timeline is accomplished in the high-level language. Each timeline entry is then coded and merged into the discrete time-event stream as their interval grows small (milliseconds to seconds) dependent upon type of event and storage requirements.

The delta timer, set by the Decoder, will initiate the desired sequence of events specified by that timed entry. It also causes an entry within the Timeline Controller to reset the Delta Timer for the next event. Entries, changes, and execution requests are reported to the system accounting function.

4. Data Bus Control. The Data Bus is under the regulation of the Data Bus Controller. The descriptions indicate that of several techniques available, either Polling (discussed later) or Oversample will be employed. The Oversample technique is implemented by sampling every device attached to the bus faster than the device could require. In other words, time granularity specifications would be developed for every device on the Data Bus and would be tested for ready, waiting-out, waiting-in, and error conditions. These tests would be made within the time constraints of each device.

Polling (discussed under Input/Output Control) requires periodic sampling of devices. It is similar to the Oversample approach but does not imply the hectic activity connotated by digital oversample techniques. A polling signal is captured by waiting devices to request data bus attention.

The assumption is made, from these and additional considerations, that the Data Bus Controller is programmable. Device characteristics will change according to the progression of experiments and the processing of data. Devices will be connected and/or removed from the bus. Thus, the Bus Control must be as dynamic and flexible as the system it serves.

The relation between computer (DMS) Executive and the Data Bus Controller is shown in figure 18. A "Data Bus Programmer" routine will maintain a Data Bus Control sequence basic set of instructions. From the Device Inventory List (for oversample) or Active Device List (for polling) a complete sequence of control instructions is created and mapped into the Data Bus Controller. The Programmer updates control sequences and maintains queues associated with the Data Bus. It records a change history by waking the accounting routine.

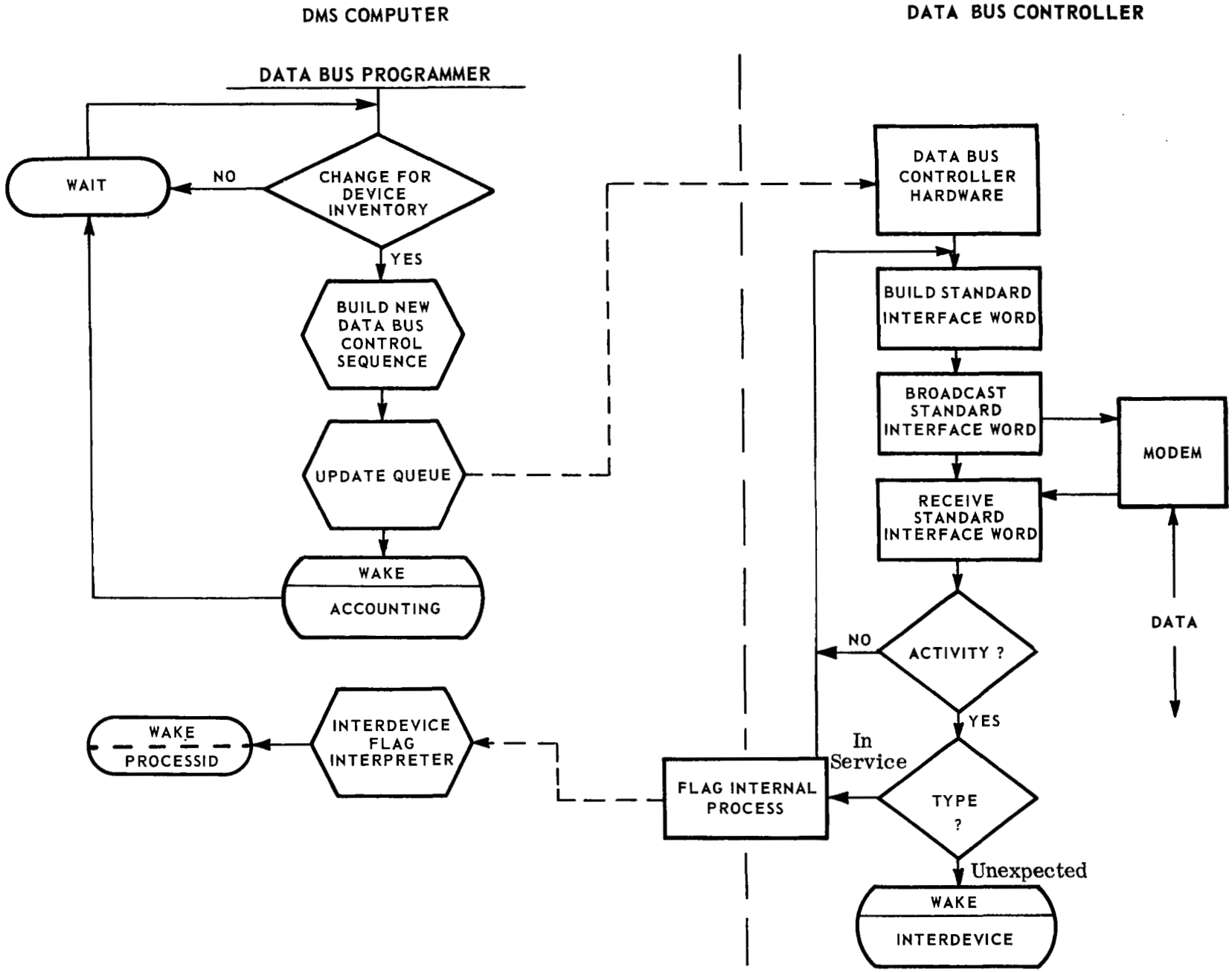

FIGURE 18. RELATION BETWEEN DMS EXECUTIVE AND THE DATA BUS CONTROLLER

The Data Bus Controller will, theoretically, cycle through the programmed sequence relieving the computers from considerable repetitive actions. The Programmable Controller, for example, will build the standard interface word, broadcast the standard word, and interpret the replies. The DMS computer has already armed the Controller by specifying the frequencies to be used and the sequencing of devices.

If there is device activity that requires computer attention, a flag is set to indicate the proper internal process. This will be accomplished similar to the interrupt procedure. In short, a flag bit is sent to all participating ALUs. Only one ALU will have a corresponding mask bit set and will make the required response. The mask registers should be alterable by the DMS computer.

A special case occurs when a direct device to device transfer is requested (see figure 19). The combination of time division and frequency division multiplexing will permit design for direct transfer of data between devices (possibly not in a pure digital oversample environment). Such capability would require a flexible relationship between computers, controller, bus and devices.

It is assumed that devices could not initiate an independent action of this sort. Rather, the Executive will have made the proper resource allocations and programmed the Data Bus Controller to react with the correct assignment connection control. As illustrated in figure 20, the requested frequency or device may be busy and require the same queuing that would be incurred under any control circumstances. An extensive amount of work is needed to define the optimum time division multiplexing (TDM) scheme. The Data Bus Controller will perform the switching, prevent interfering assignments, approve orders and mix ancillary information. These functions are prearranged by the DMS Computer Executive. A prearrangement could include the assignment of devices to other Executives for allocation and control. The devices peculiar to various experiments, for example, will be allocated by the experiments multiprocessor Executive. The experiments Executive will require interaction with the Data Bus Controller through commands/requests sent to the DMS computer Executive.

A particularly interesting involvement here is in the proposed device-to-device transfers which would include shared resources (i.e., bulk storage, transmitters, etc.). The concept, functionally reasonable, could be complex to implement when the possibilities are carried to their ultimate conclusions. A functional simulation would enable the description of the bus control in parametric terms and permit the necessary variation of design features. A detailed design could result from this investigation.

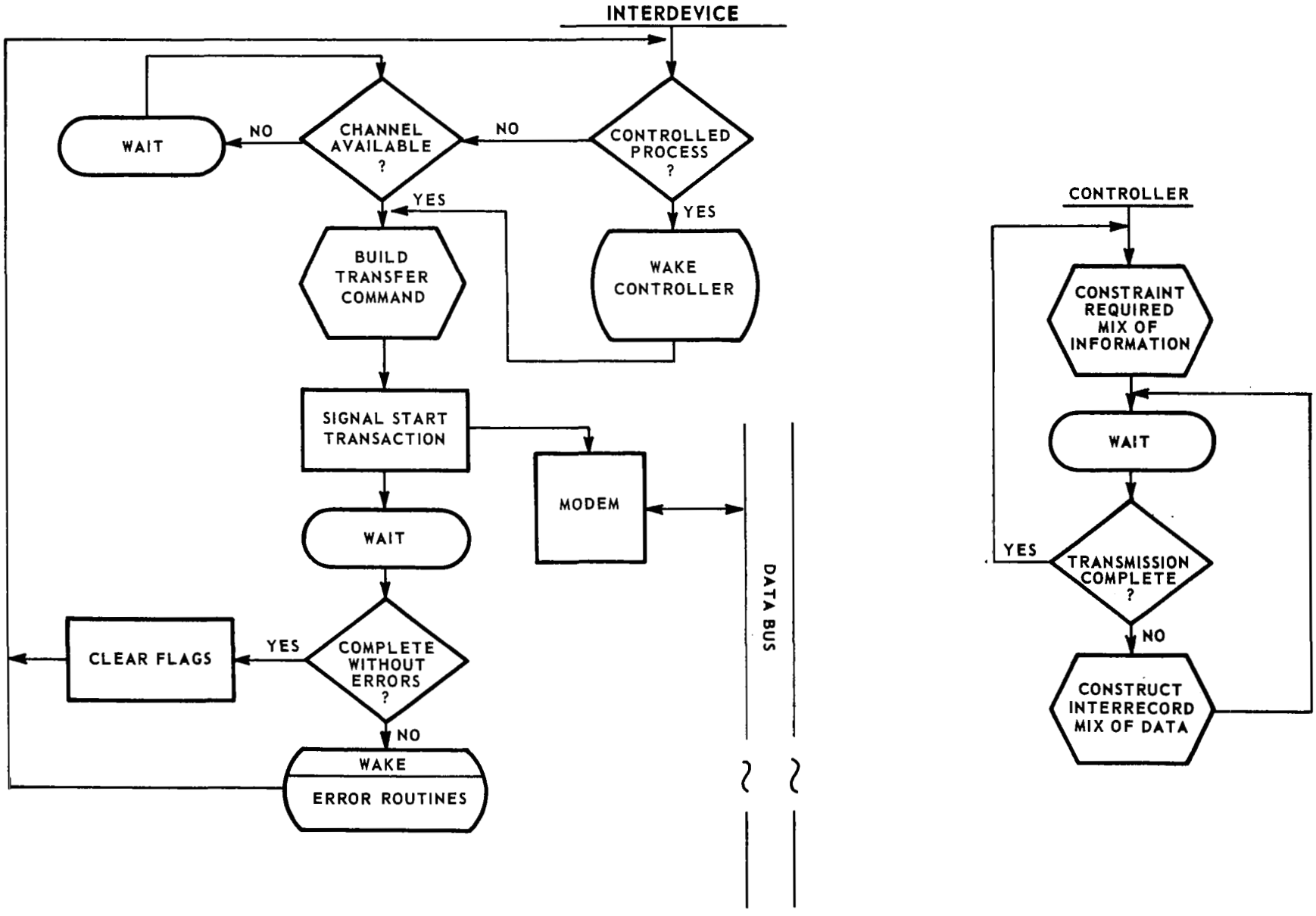

FIGURE 19. INTERDEVICE TRANSFER

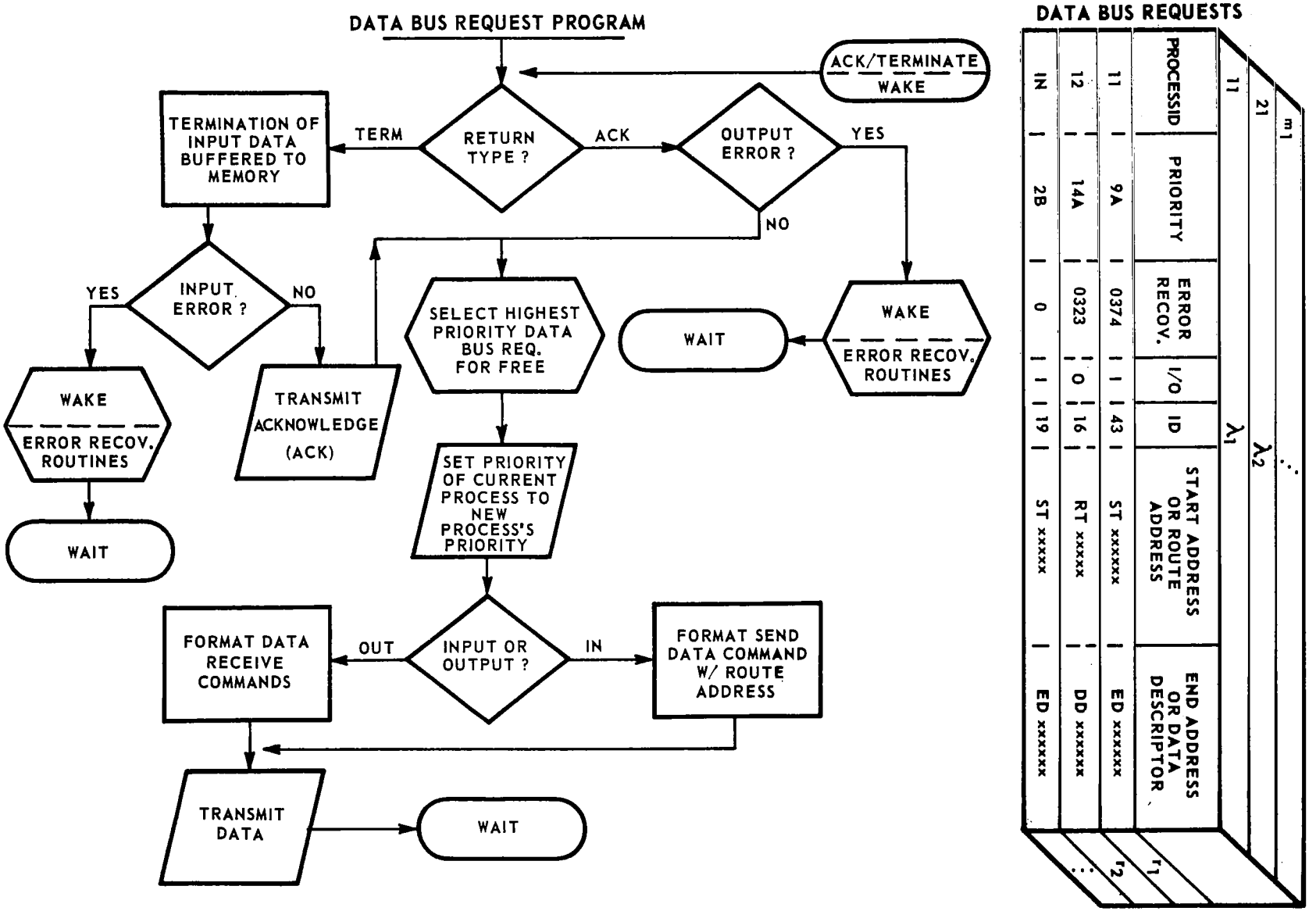

FIGURE 20. DATA BUS INITIATOR

An interdevice transfer capability should be of significant assistance in relieving the computers of redundant data handling. However, every innovation has a price and that price should be determined before detailed design. As a simple example, it could be necessary to return to the Controller Programmer for optimization of seek times, assignment of priorities, etc. The implications for added switching time, intercomputer signalling, Data Bus Controller reprogramming, frequency reassignments, TDM formatting and other interactive complications would not be simple. A study should be undertaken to define the data bus activity and thoroughly exercise the total concept in a simulated environment.

A more detailed description of the Data Bus Control procedure can be envisioned by considering requests for bus activity. As illustrated in figure 20, a two-dimensional queue describing the bus requests is constructed. Since operations can occur simultaneously on each frequency division of the data bus ($_1$, $_2$, ..., $_i$), an assignment for input and, if different, output frequency is made. The arriving requests are ordered (r_1, r_2, ..., r_j) according to an Executive algorithm.

Requests could be serviced in a simple first-in/first-out, or last-in/first-out procedure. However, at least a priority rearrangement is expected. The algorithm to establish such an order would be based upon original process priority, time in the queue, the service time required, and the interaction (such as minimizing arm movements between tracks) with other requests. The priority rearrangement should not eliminate some processes from gaining a requested device, but neither should it block very high priority requests.

The Space Station IMS is a major subsystem that will effect the fulfillment of mission objectives. The hardware and software that constitute the data paths for communication, command, control, data acquisition and data dissemination should be extensively investigated before baselining. When a basic system design is selected, the software will probably become the critical-path item. Prior definition of hardware/software interaction in the Executive communication area will reduce the relative criticality of the overall software development effort.

5. Input/Output Control. The major function exercised by the I/O Control Program is the allocation of access to modem frequencies and unit controllers. Many paths are available after the RDAUs which will cause some complication. Buffering at the terminal, within the RDAUs, and at individual devices will be individually specified. A fast Data Bus is proposed which could cause multiple transmissions to the same terminal, for different end devices, to interfere at intermediate buffering and switching. Thus, I/O control may be several Executive routines to manage and manipulate the I/O functional requirements. The procedure could be further complicated by the fractionating of responsibility of device allocation among different computers while attempting to maintain central control.

A secondary control function is to minimize device idle time. This will be done by maintaining tables of available units, the paths to them, status, and assignment. Assignments will be made to leave a maximum number of paths to devices. A queue search, such as illustrated in figure 21, will be used to match requests with paths to the end items. Status will then be locked to prevent multiple assignment unless message merging can be established.

There are many ways of providing the I/O services required. A mix of these procedures will be required dependent upon frequency of usage, memory requirements, designs of peripherals, and overall configuration.

A pool of routines can be maintained which would derive their uniqueness from the process control block of the user process. Thus, multiple copies exist, or if reentrant code, only one copy exists for similar devices. The Process Control Block will maintain the current status of the process's relationship to the device. For example, a process would not be removed from main memory while an I/O operation was in progress, under present thinking. Thus, a WAKE executed by completion of an I/O operation would have a WAITing process to accept execution.

A Common Pool (COMPOOL) method can be used for the exchange of messages between ALU and devices or other ALUs. The COMPOOL procedure provides good isolation since there is no direct contact or signalling between any divisible units. A shared buffer is filled and a bit set to indicate that information has been changed. Keys can also be used to controll access and destruction of any message in the buffer area. A test and set instruction should be provided to hold the buffer while information is undergoing change.

The problems associated with the COMPOOL concept include the buffer area (which might grow quite large) and the necessity for test and set locking instructions which might not be available. Due to an expected high frequency of message transmissions, an instruction or microcoded procedure would be required to prevent the COMPOOL access from absorbing an unjustifiable amount of time. This would be a good area for investigation by simulating the variance in buffer requirements, response times, queue lengths, waiting times, etc. versus various system loadings and configurations.

The major advantage of the COMPOOL concept is that processes could be removed from the system before their data transfers were theoretically completed. If a breakdown should prevent the normal data transaction, the process or an error routine would be located/loaded in memory and awakened with a high priority.

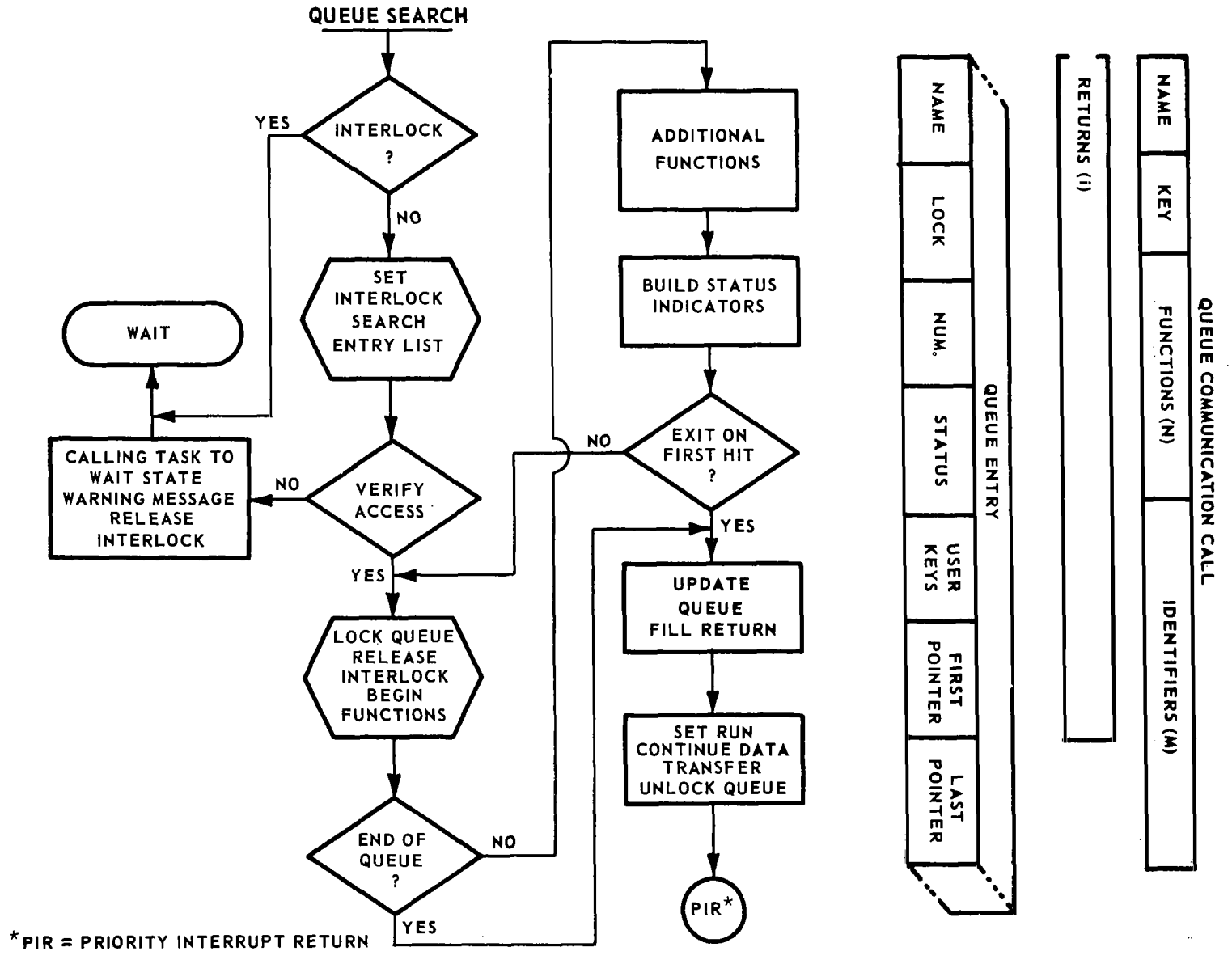

FIGURE 21. DATA BUS CONTROLLER BUS REQUESTS

A Spooling routine will be utilized to free the multiprocessors from devices that are slow or absent. Spooling is a specific term for any of several manufacturers' procedures for transferring information between devices and auxiliary storage which may be needed in the future or may have been produced in the past. For example, transmissions to earth will be made on a periodic basis. Data collected during the entire orbit must be output during these periods. Data processed by application programs that cannot be printed, plotted, or transmitted for some reason can be retained on mass storage. An algorithm must be used to determine the priority of the output as the device becomes available. Some of the decision parameters are original priority, comparative length of waiting time, deadline time priorities, length of device time required, etc.

Various I/O control programs will be resident as a part of the Executive nucleus and some will be periodically scheduled. Periodic testing of switch settings and instrument readings, for example, may perform all necessary I/O functions and raise a flag for the awakening of application programs. In other situations, programs may be called into memory expressly to provide I/O services upon interrupt demand.

If extensive time sharing by scientists at terminals is a requirement, an I/O package for the reception of codes and collection/distribution of messages will be a further requirement. The similarities of Space Station Executive software and ground-based time sharing facilities will be greatly influenced by the extent of these requirements. This is further discussed and more evident in the scheduling and the allocation of ALU time. The astronauts' faster response time and specialized training could pose an unusual "service satisfaction" problem for the I/O control programs.

Control of the frequencies (channels) on the data bus will be performed, as illustrated in figures 20 and 22, on the basis of waiting queues and resources. Sufficient information is not yet available for detailed description of data management services. However, we can define some requirements and functional procedures at a high level. The basic assumption is that there are waiting queues of requests and a limited number of resources to be shared. The algorithms to actually control the sharing process will be a function identified by memory requirements, time constraints, device availability, priority, waiting time, etc.

I/O control requirements can be carried much further within the Executive by comparing seek times, rewind times, and optimizing service (by user or by resource). A relative assignment priority is thus produced to determine the allocation of the frequency (channel) to the requesting process. As illustrated in the figure, each time the frequency slot becomes available through a change of state, the waiting queues will be scanned and an optimum assignment made.

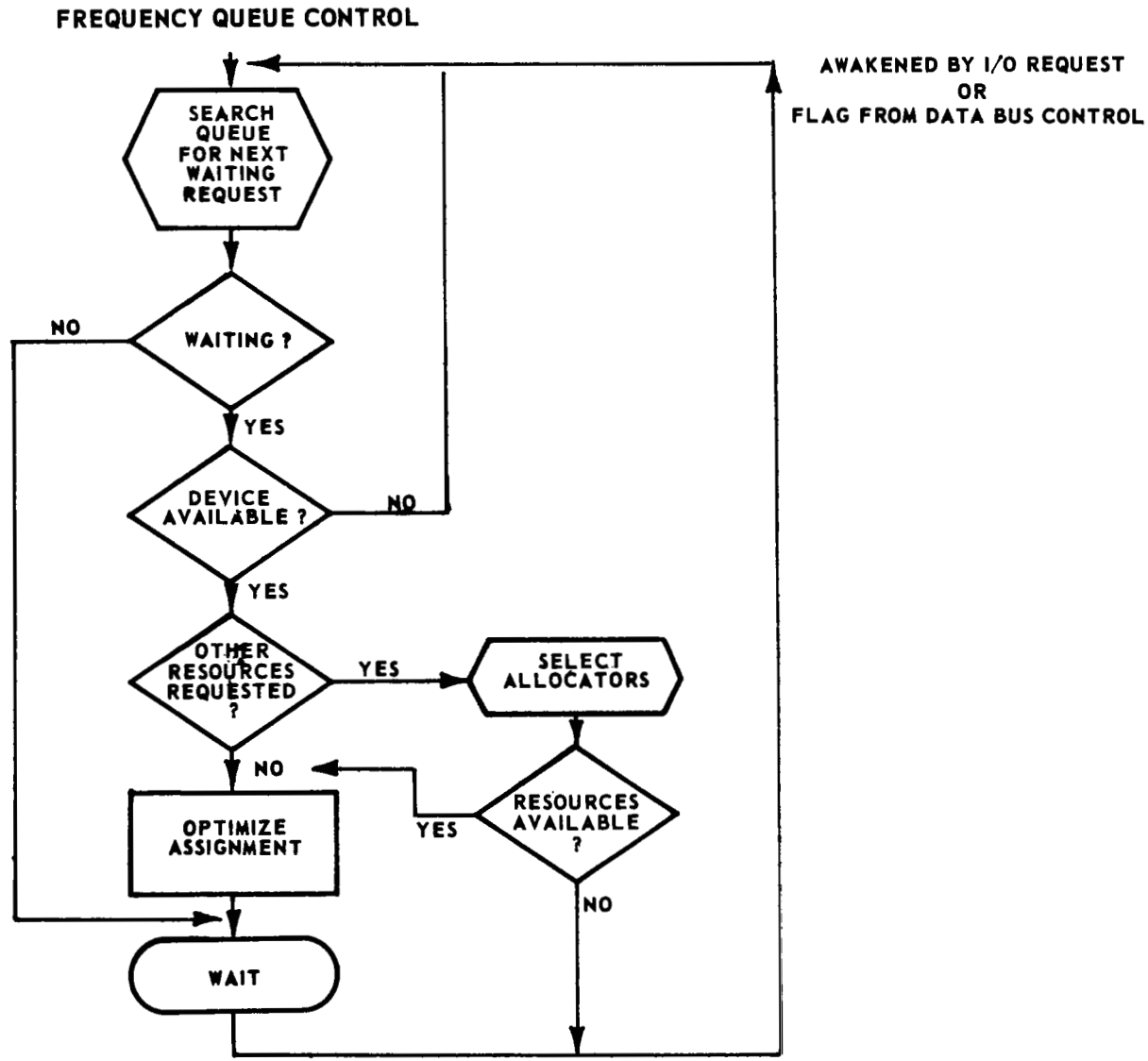

FIGURE 22. FREQUENCY QUEUE CONTROL

The organization of data in files will complement the maximum use of both data bus and devices. The file organization should be such as to permit direct memory access with minimum directory assistance after a transfer has begun. A table of application files in use will be maintained in auxiliary memory to facilitate this function. A hash coded procedure for "spilling" information down a memory hierarchy where each succeeding level is slower than the preceding is a possibility. File organization should be extensively investigated before any detailed design recommendations are made.

Connection between the I/O device (or end equipment) and the data in storage is dependent upon good configuration design tradeoffs. The two most commonly used procedures for interdevice communication are interrupts and polling.

a. Interrupts. When a device requires service, it raises a line to the ALUs. The hierarchical position of the line indicates its relative priority to all the other possible incoming lines. For flexibility, a study should be performed to tradeoff the advantages of preprocessing interrupts either in a special unit (perhaps similar to a Bus Controller type function) or within the computer under special design considerations. The interrupt, historically, sets a latch which the software will reset after the service has been performed, thus permitting the next highest set latch to seize the processor.

Interrupts should probably appear at all ALU interrupt interface points. However, according to Executive control, only one ALU could be interrupted by a sufficiently high demand. The Executive could arm an interrupt mask register for each ALU corresponding to the possible assigned interrupts that are active. An AND operation would set latches for relative interrupts and ignore irrelevent ones.

The recommended study should determine the feasibility of changing the priorities of incoming lines via software, recognizing interrupts and queuing them in a pop-up hardware stack, arm/disarm/acknowledge, and other facets of interrupt processing. Considerable complication could possibly result from an overdependence upon interrupts, especially if primary circuitry is through the data bus. An alternative to an extensive interrupt system is polling.

b. Polling. Periodic Polling is illustrated in figure 23. The illustration primarily is addressed to the RDAUs, but a Polling List and periodicity can be established for each class of equipment. A queue of the devices to be polled and the required attributes and descriptors of each is maintained by the Executive (which might suggest a fast auxiliary memory). At the required polling interval, each device with action status falling within that period is commanded to transmit a formatted word. A simpler method is to transmit a

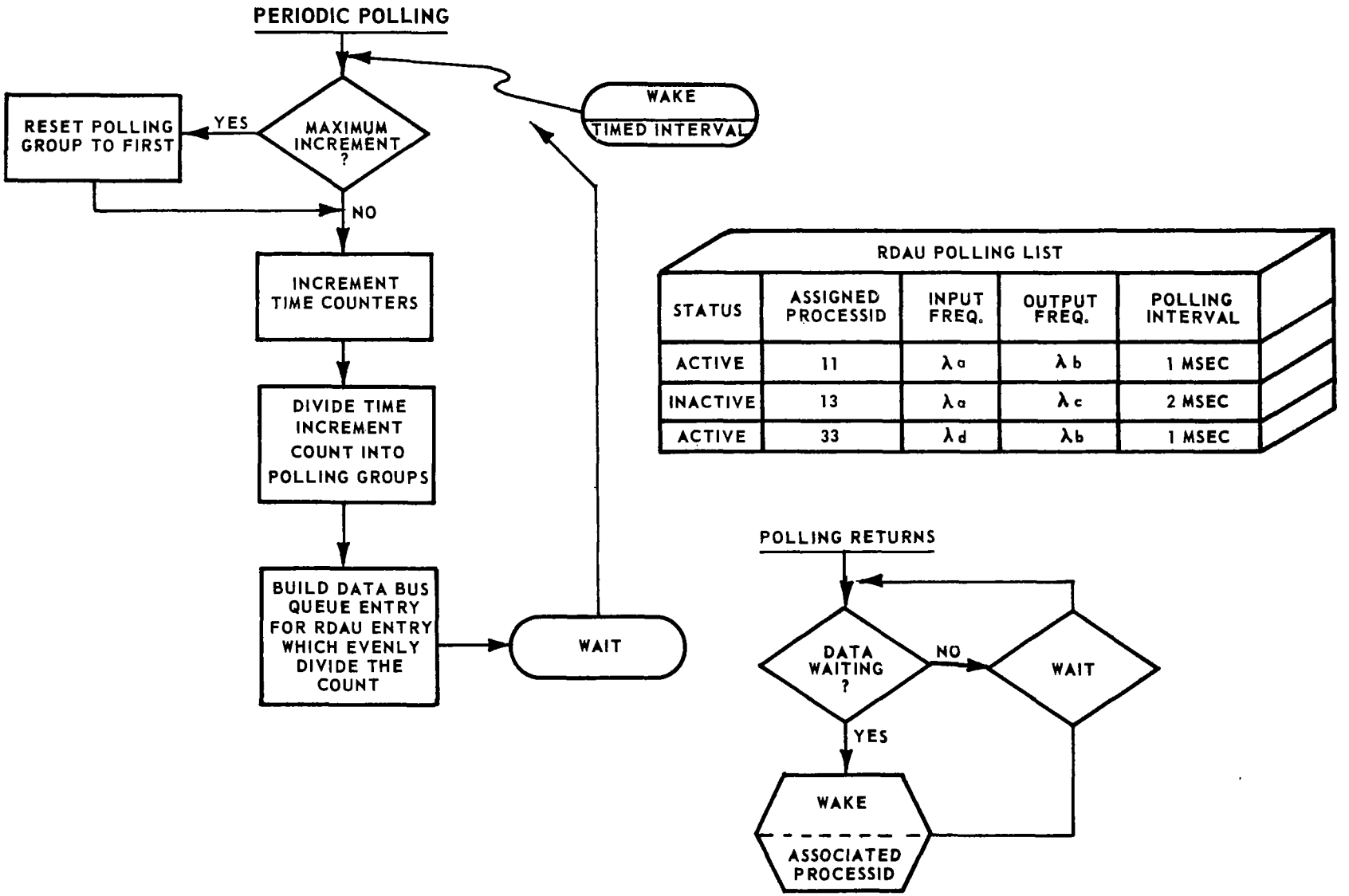

FIGURE 23. PERIODIC POLLING

polling "signal" whenever the transmission facility is available. The signal is captured by the first device that is prepared to utilize the facility.

Here we have alternatives to consider. At this level of functional design consideration, let us assume that there are several possible responses (indicated either within the format or within the Polling List). The object is to provide necessary service, best possible service, and flexibility.

For example, the arrangement could be such as to poll all the items on the Polling List for their descriptive format word. The Polling List can contain the relative priorities of each device demanding access in each frequency slot, so that priority could be utilized as a factor. Some of the devices responding as "ready" may require a minor data transmission which is being held by the Executive (or will be accepted by it; e.g., status), and should be quickly serviced to remove them from the holding condition and to release tasks that may be waiting.

Polling should have a sufficiently high priority to allow it to continue through a significant amount of work without being preempted. But, this is a matter to be determined by simulation after hardware design considerations of Polling versus Interrupts has been better defined.

6. I/O Error Recovery. I/O error recovery will be planned for every phase of information exchange. Our present context is illustrated in figure 24. The type of recovery attempted will depend upon the device involved, the available information describing the failure (channel status bits, etc.), and the type of data being exchanged.

For devices, tape or disc for example, the normal procedure is to attempt to complete the read or write operation a sufficient number of times to establish that it is a hard failure. If other paths (e.g., other controllers, etc.) are available, a reconfiguration may be attempted.

Output data, generally, can be spooled during reconfiguration or repair. When the original device is again made available, the data recorded on intermediate storage will be routed to the correct device. The spooling operation will continue until the device catches up with the saved data.

Data that cannot be input may be disastrous to the subprocess and require the suspension of the subprocess which would lead to suspension of the process. If data cannot be obtained, it will be the application program's decision to perform unaffected processing or to provide warning messages and request suspension (removal from the system via SUSPEND procedure). An error exit, EXITI procedure, will also be provided so that executive system functions can be utilized to obtain intermediate parameters, dumps, snapshots, and process-control-block images.

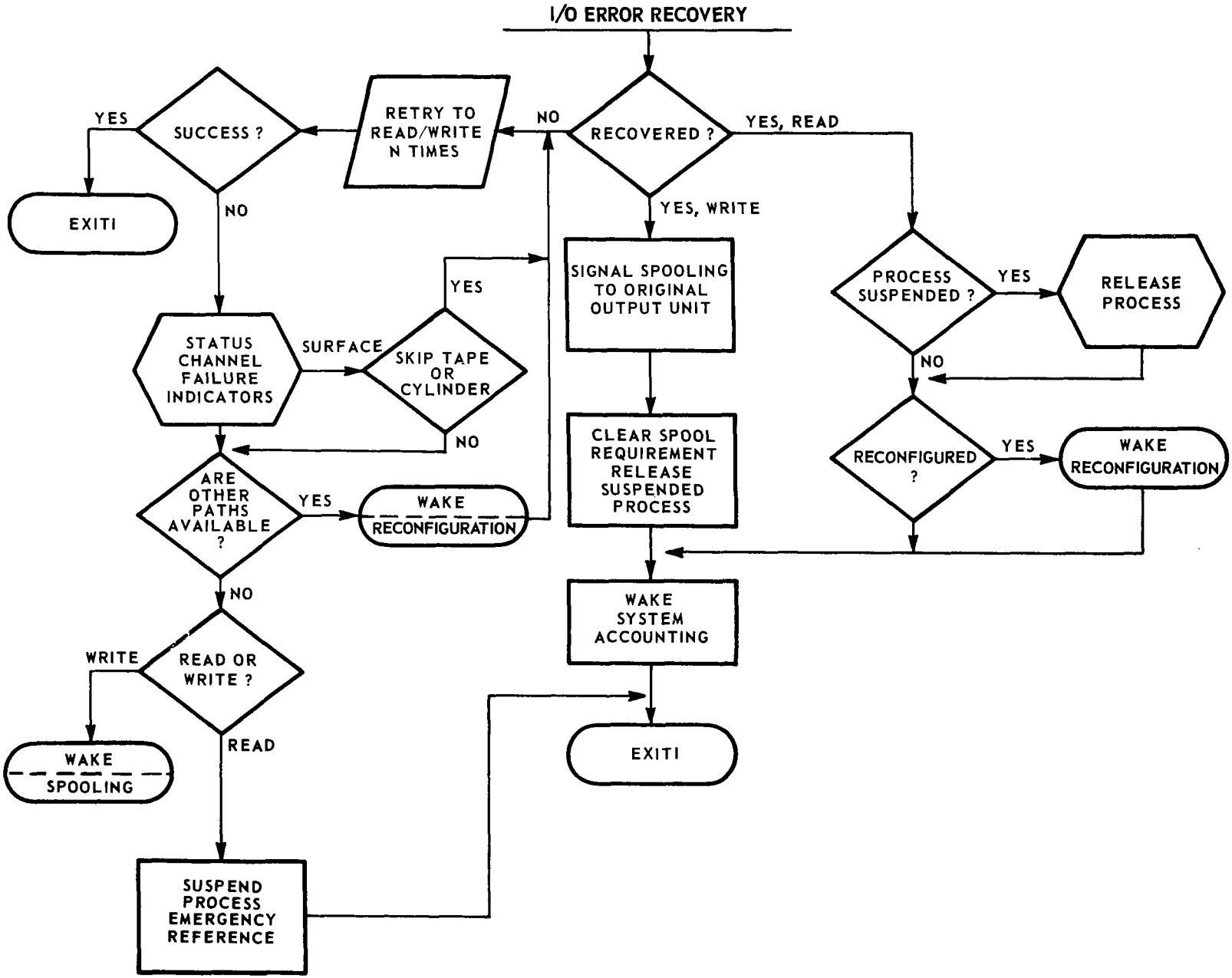

FIGURE 24. I/O ERROR RECOVERY

7. Communications. Both uplink and downlink transmission/reception must be considered by the computer configuration. At this time, realistic estimates of the loads that will be imposed upon the individual computers and the total system cannot be made. However, it appears that a one-hour dump (downlink) per day will be available for collected experiment information. Due to orbit and ground station physical relationships, this hour would probably be spread across discrete intervals of the 24-hour day. Other communications would be on a random demand basis from the ground or by crew-initiated application programs.

The uplink information may be decoded either by the receiving hardware or the associated ALU (figure 25). This procedure will involve type of format, timing, addresses, check and test codes, and error protection. Elaborate measures may be required (such as redundant transmissions) for operational commands or software program changes prepared on the ground. General information or long-range assignments subject to frequent change might require only normal validity checks.

The downlink is simply the converse of the uplink. Data is buffered from mass storage to main memory via any required encoding and a direct memory access function with which to supply the transmitter at its required rate. Of course, these are all basic notions and subject to the configuration design. The Data Bus and the Data Bus Controller designs may have significant impact on the software functions required and the extent to which processing can continue. For example, a cycle stealing mode versus block transfers to controller buffers versus dedications of memory units are a few of the possibilities to be considered.

Application programs will precondition and code the data for downlink transmissions. The supervisory software will be concerned with construction of an identification block with timing, coding and addressing information. Such information as data decompression techniques will be agreed upon in advance or imbedded in the information transmitted. The message will, then, be clocked to the transmitter buffer under control of the transmitter's encoding-timing circuits.

Greater complication can be added to both uplink and downlink processing by requiring that a message be retained until its accuracy is verified and that unique recoverable points are identified if processing is proceeding parallel to transmission/reception. It is also possible that all messages will be recorded in toto both at ground and orbital stations. This would imply application programs for data recovery/enhancement. The significant Executive impact will be to provide sufficient storage and data management procedures. Both transmission and reception are time-shared operations since they will operate concurrently but not fully utilize ALUs and auxiliary storage.

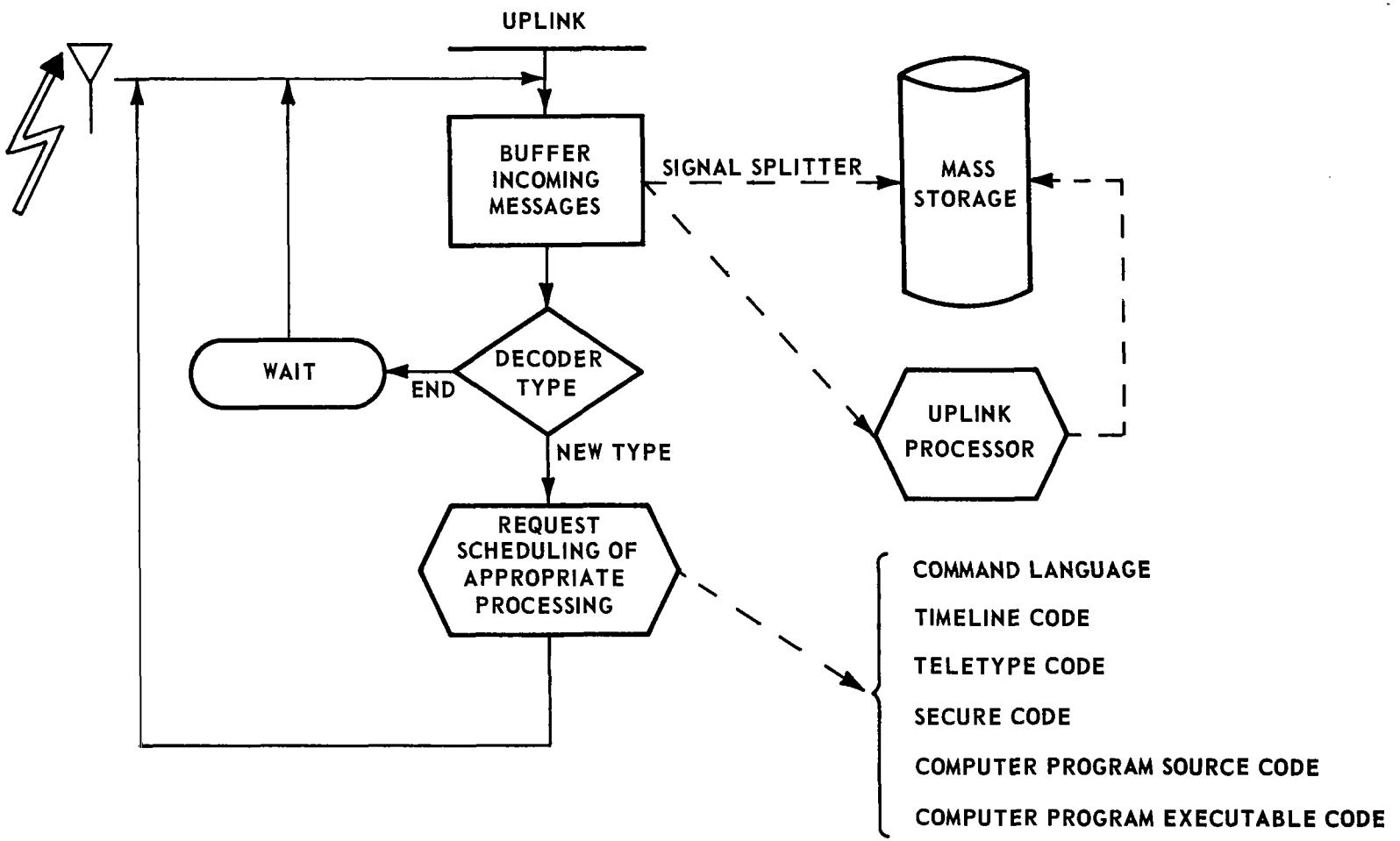

FIGURE 25. UPLINK DATA ACQUISITION

8. Fault Tolerance. Modern computer hardware/software technology produces single components with high resistance to failure. The larger the number of components, for example at the ALU and memory module level, the higher the probability for a single failure (intuitively). However, the long MTBF ratings of the components and normal computer overdesign indicates a very low probability of multiple failures in separate ALUs. The expected course of action would be to maintain a configuration map and the capability for switching the components' interconnections.

Failure analysis and error handling capabilities will require extensive planning to make maximum use of currently available and new technology to Space Station configurations. Specifications developed for error handling and failure recovery will include the specification of procedures such as SUSPEND, ABORT, and RETRY which, in following discussions, evoke intuitive relationships to memory, hardware, and microcode implementations.

A variety of diagnostic procedures and recovery algorithms will be required to support the unique Space Station IMS configuration. Some of these functions are described to illustrate the scope of the problem and some basic attacks on functional aspects of the problem.

a. Real-time Diagnostics (figure 26). There are several classes of diagnostics. For example, diagnostic procedures may involve parallel computation and comparison, or the external use of a standard input signal, or real-time diagnostics which are run periodically as a process with low priority. In this case, additional hardware circuits for self-test, parity comparisons, and error testing are included as real-time failure detections. This distinction is made to separate executive features from the maintenance procedures. Maintenance procedures may include extensive use of microcoded tests and software tests, but are not generally considered part of the real-time scheme. This restriction may be revised for the Space Station.

A failure detected in real-time is pursued dynamically (figure 26). If the failure has occurred before and been retried as a possible transient, it is now assumed to be a reasonably hard failure and an attempt is made to reconfigure the system to maintain the most viable system with the remaining components. The system accounting function may be required to maintain failures versus time. Algorithms would then determine the rate of failure and measure it against acceptable thresholds. The type and rate of failure will determine the procedure to be followed for individual components.

The desired process includes the selection of a specific diagnostic procedure to isolate the failed component to the lowest possible level (perhaps to a line replaceable module or a line replaceable card). This is dependent

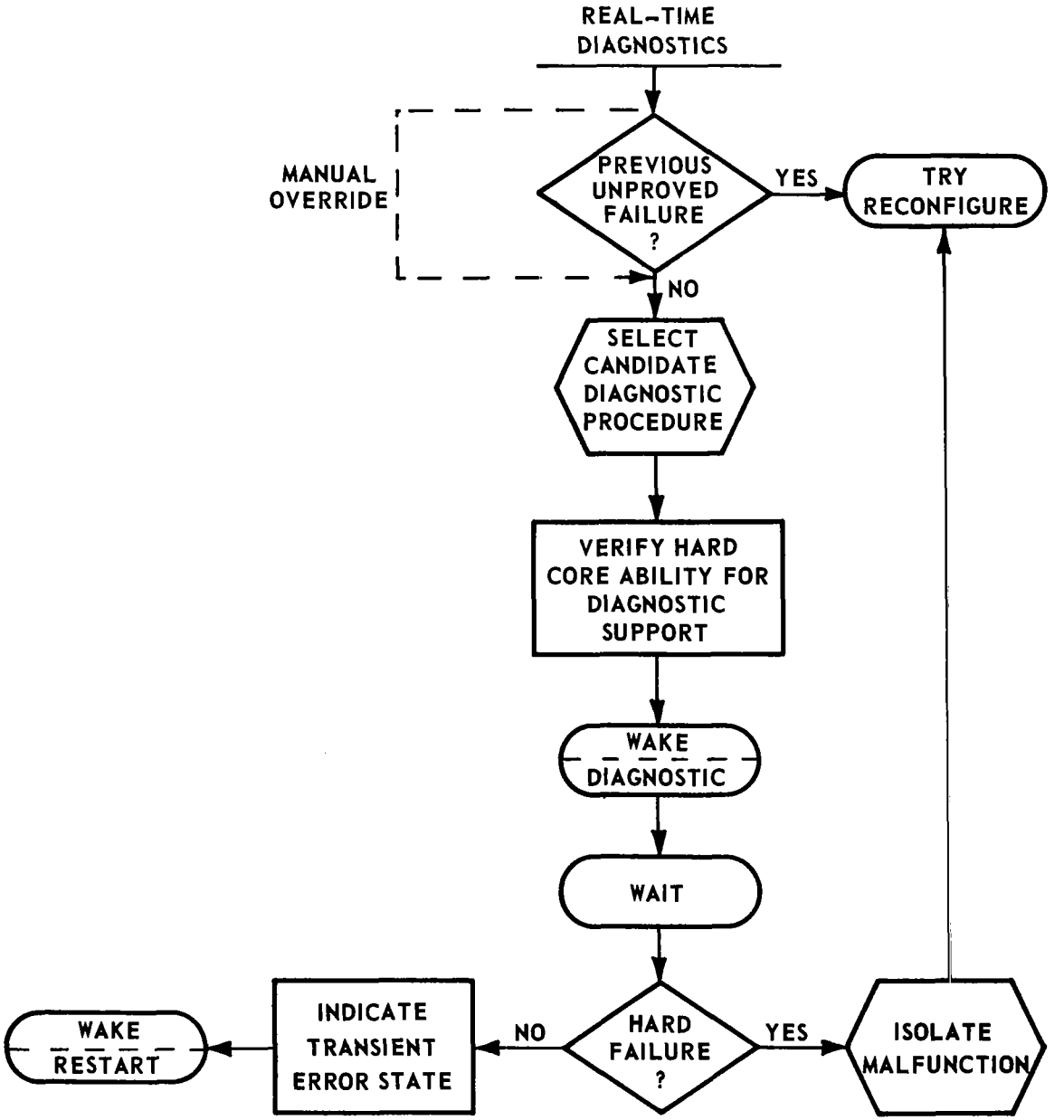

FIGURE 26. REAL-TIME (R/T) DIAGNOSTICS

upon a hard core of memory and instruction execution capability within which to start the diagnostic process. Hard core does not refer, here, to a type of memory but to an unviolated nucleus of hardware and software. Once the individual diagnostic is WAKED, it will provide additional instructions or CALL additional procedures or WAKE other diagnostic processes required to diagnose the failure.

A manual override (a configuration display and control console is recommended) to provide the astronaut with the capability to manually command diagnostics and to manually select configurations. The procedure would be used to remove a module from the system for bench tests or spare replacement.

b. Controlled Retry. The capability to attempt the execution of an instruction that resulted in failure without committing to the run state would be useful. This could possibly be executed on a single ALU. However, it is primarily designed for multiple cooperating ALUs.

The controlling ALU simulates each step of the instruction and commands the execution of that step on the failed ALU. The controlling ALU then acquires the data from the intermediate registers of the failed unit and compares it with the expected results. Failures are identified to the astronaut if all steps are not completed. If the execution is correctly performed, the process may be returned to the ready state, but this is dependent upon several parameters (i.e., did the failure destroy memory, has the process been reassigned to another ALU, can the particular instruction be reexecuted, etc.).

c. Multiprocessor Failure. The preceding discussions have applied to failures in general, but what of the concurrent failure of the multiprocessor's two ALUs?

It is assumed that a failing ALU, as illustrated in figure 27, can test its previous condition to determine whether it was in the error state and has experienced an immediate recurrence. If so, this indicates that the ALU was isolated and any results would not be dependable. Specific failures for a specific detailed configuration will produce a variety of recovery procedures, but for this case, a halt of operation is in order. If a cooperating ALU is operating, a signal would be transmitted to the surviving ALU (figure 28) and the failed ALU enters an aborted or waiting error-state.

If the ALU was not previously in the error state and has no cooperating ALU, it will attempt recovery. This includes the saving of all available internal registers and inspection of the decoded failure. If data has been destroyed, the affected processes will be terminated. In any event, an attempt is made to identify the failure as hard or transient, to recover processes in execution,

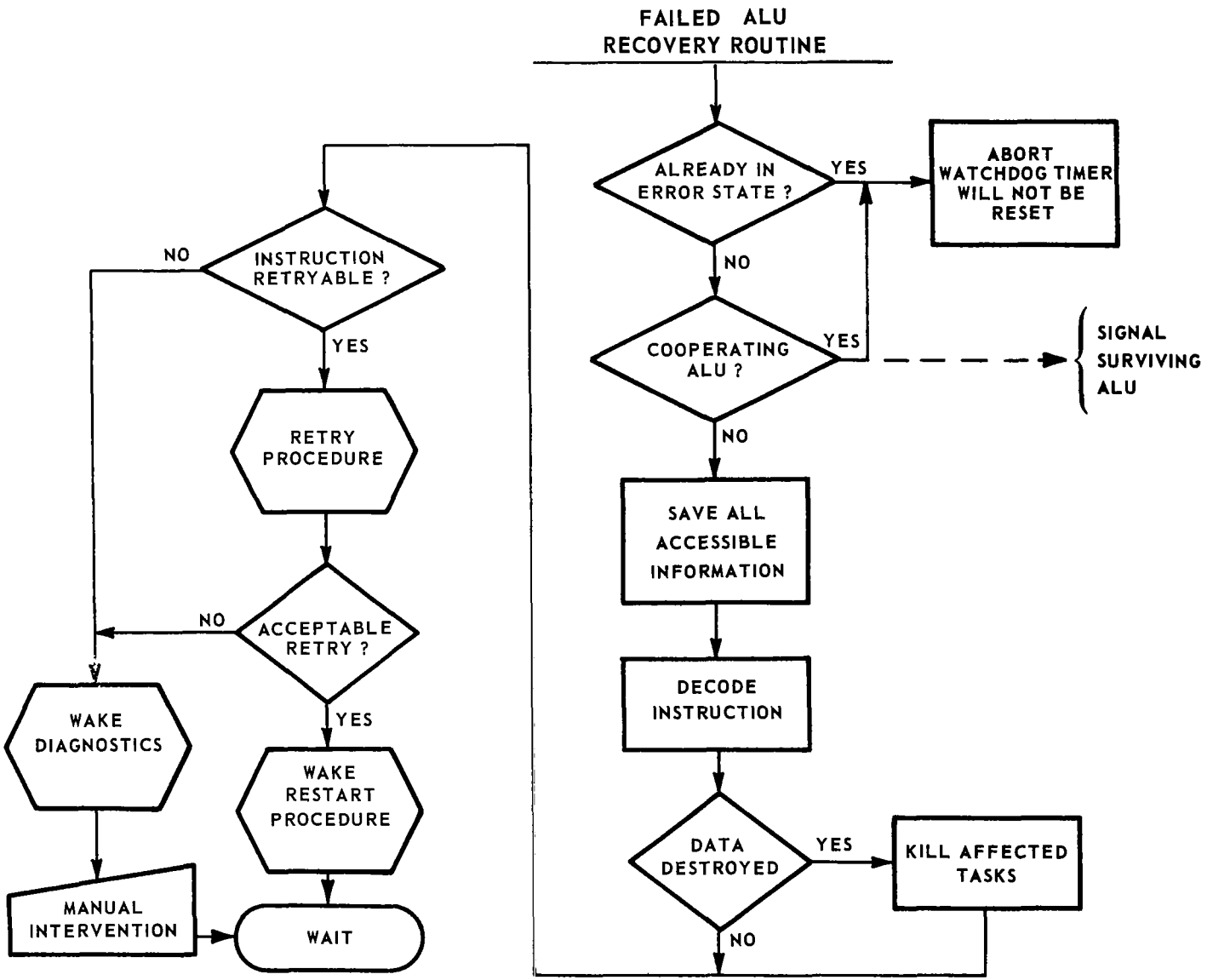

FIGURE 27. FAILED ALU RECOVERY ROUTINE

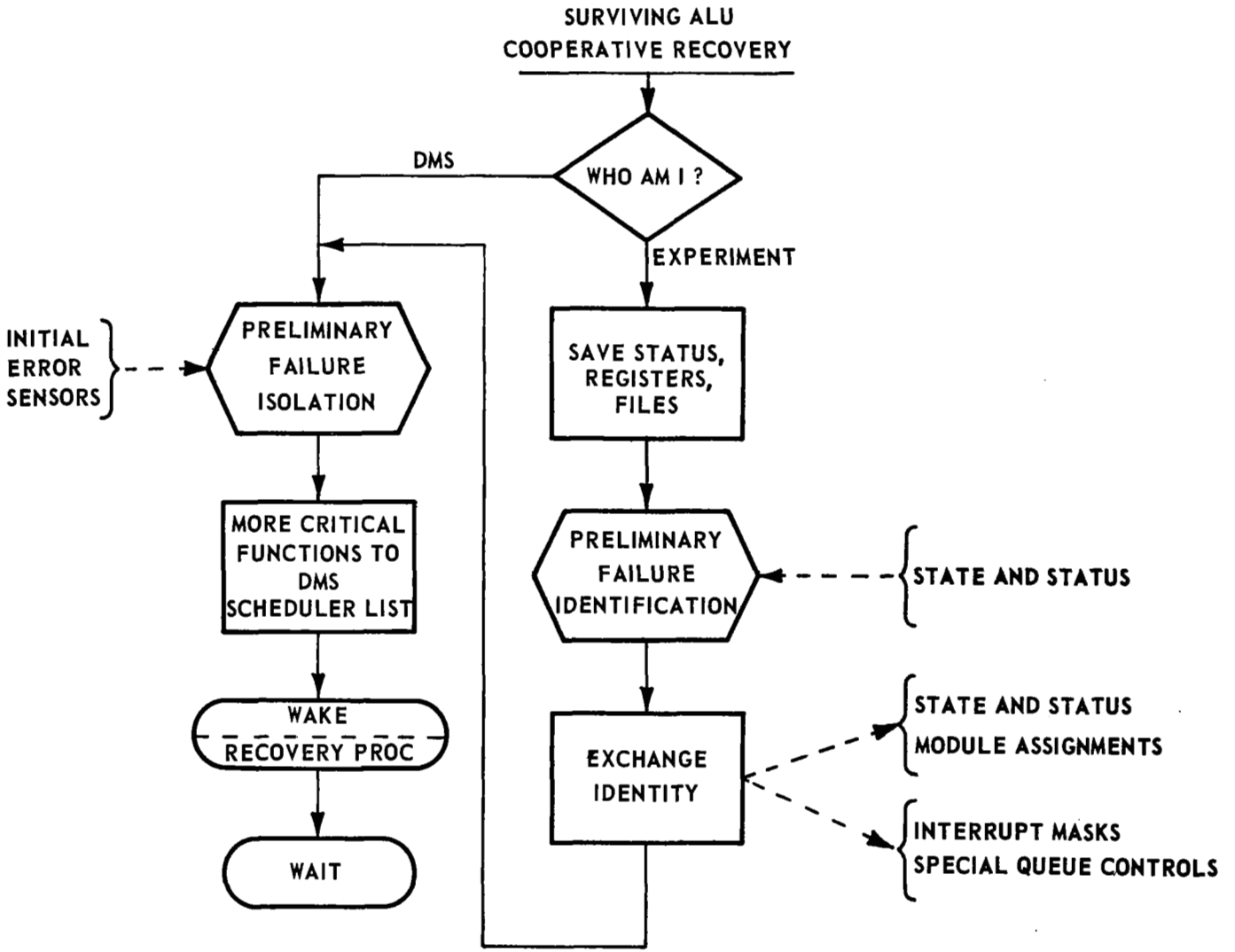

FIGURE 28. SURVIVING ALU COOPERATIVE RECOVERY

or to attempt restart. If all else fails, diagnostics for isolation of the fault and manual requirements for repair are in order.

A surviving multiprocessor ALU (recipient of a failure signal) must first determine whether it is the DMS or Experiments computer. If it is the Experiments computer, it will save its current state vectors and configure itself as the DMS. This reconfiguration may be simply the switching to a separate set of microcode or as extensive as a complete reload and initialization. This is dependent upon the system configuration, the type of failure sustained and the detail design specifications.

The surviving multiprocessor will execute the DMS functions of commanding the Data Bus Controller, scheduling, data management, etc. Certain experiments, however, could be in the process of controlling equipment that would be damaged by ignoring them. Several possibilities exist; equipment design could include automatic safety locks for loss of control signals, limit switches for antenna and telescopes could be calculated by the DMS from GNC-provided attitudes, a failure-mode scheduling algorithm could go into effect with high priorities for essential DMS and Experiment functions, etc.

The surviving multiprocessor, as illustrated in figure 28, must basically perform two essential functions: assume the DMS command, control and data management functions, and begin recovery procedures for the failed multiprocessor. The recovery procedure could be a controlled attempt to execute the failed instruction as shown in figure 29.

The state diagram (figure 30) will further clarify the discussion. The solid line encloses all failure states and the processes that may be taking place. Outside the solid rectangle is the run condition and the not-failed state. A run state ALU failing causes it to enter the error state. If it has communication with a cooperating ALU, the commands and sense transfers are between the two states. If the error is proven to be transient, the continued run is prejudiced. That is, another failure will indicate a hard or frequent failure rate that may or may not be tolerated. If the problem is removed, perhaps by reconfiguration, the run continue state can be entered without prejudice.

The recovery procedure is illustrated in more detail in figure 31. This takes into account the process functioning at time of failure. An executive failure is potentially more catastrophic, within the present discussion, so the recovery procedure must include this investigation. Extensive analysis should be performed to determine nucleus routines and procedures for testing the hard-core's viability. The figure graphically illustrates a requirement for the capability to extract information from intermediate registers and sense lines. Sufficient intelligence gained from the failed ALU will permit the decisions, which have been previously described in some detail, for maintaining a viable IMS.

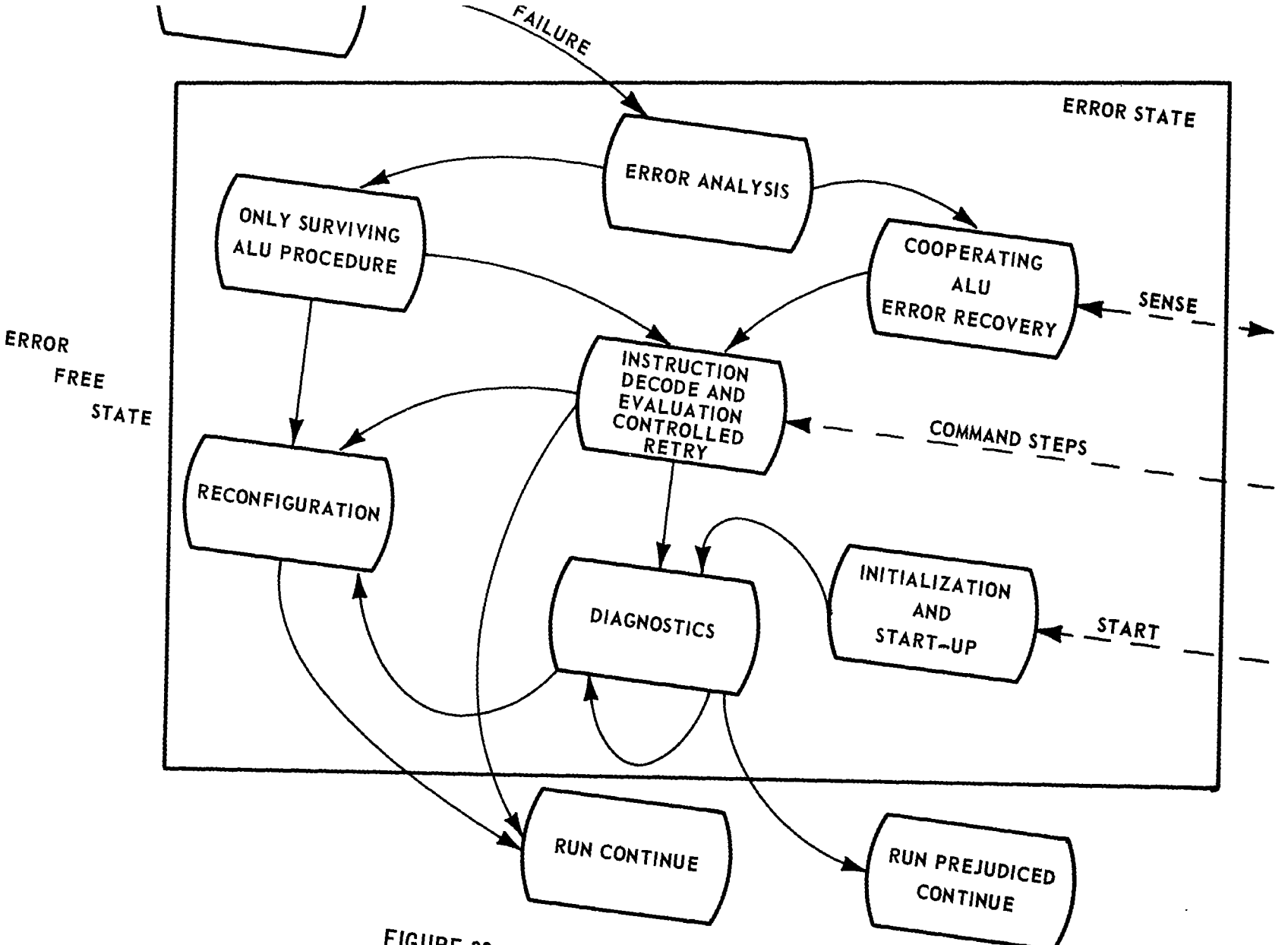

FIGURE 30. ERROR STATE TRANSITIONS

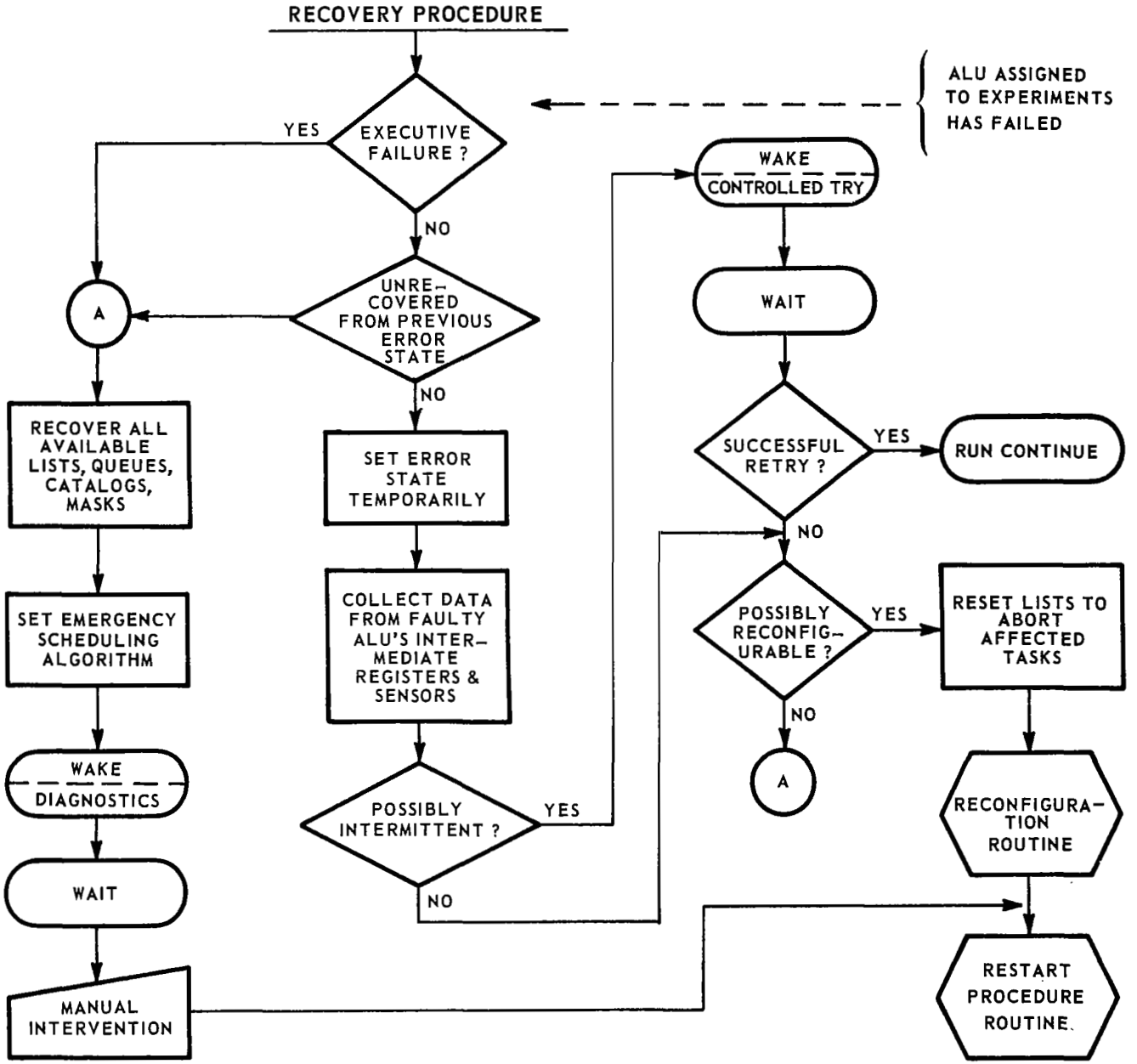

FIGURE 31. RECOVERY PROCEDURE

d. Reconfiguration. The reconfiguration of the IMS has been alluded to several times in the preceding discussions. Procedures for determining when a failure necessitates alteration of configuration have been considered with some ideas for preserving the DMS functions. However, the configuration is not yet specified in sufficient detail to design a procedure for reconfiguration, even at the functional level.

A flexible capability would include a configuration display and control console. The crew should have either a continuous or "on-request" display of the configuration status, component status, and the available spares list. The display could include a block diagram of the configuration with status, operating-hours, failure rates, utilization and data unique for each class of equipment. The computer block, for example, would include the multiprogramming efficiency, storage utilization, file lengths, allocation conflicts and other descriptors. Such a display will be of significant value during development of the Executive and integration of application programs. It will ease the difficulty normally experienced in "tuning" the Executive. Tuning refers to the process of adjusting variables (e.g., time quantum, file-lengths, etc.) to optimize processing.

The control capability is dependent upon the switching available and the routing of data. Switching is specified (figure 1) which indicates that individual system components can be used through a switching-interfacing component. This implies that control can be established over the components. The recommended procedure would be for the DMS Executive to maintain the status and control information for the overall configuration. In the event of failure or upon crew command, the DMS would reconfigure the IMS.

If all data (e.g., sensor inputs, control outputs, etc.) are on the data bus, the appropriate modems can be switched for reconfiguration. If these data signals are on buses internal to specific subsystems, additional configuration adaptation will be required to provide flexibility without incurring penalties in additional switching, signal conditioning and timing.

9. Utilities. Utility programs are basic to development and utilization of a computer system. For example, a package of programs is required to perform the mundane functions of operating a computer facility. This package will be defined by the types of operations, manual and automatic, that must be performed. For instance, the maintenance of system history, figure 32, is a utility function. The handling of mass storage, editing, debugging packages, compilers, assemblers, tape to tape copy, etc., are other examples.

Mathematical routines required include all the standard packages to support calculations (e.g., logarithmic, exponential, trigonometric, etc.). Additional programs will be required, perhaps specified as application types, for matrix manipulation, coordinate translation and rotation, display preparation and alteration (e.g., scissoring, zooming, perspective, simulated perspective, etc.). The support of display systems will be a major supporting function of the Executive for applications programs.

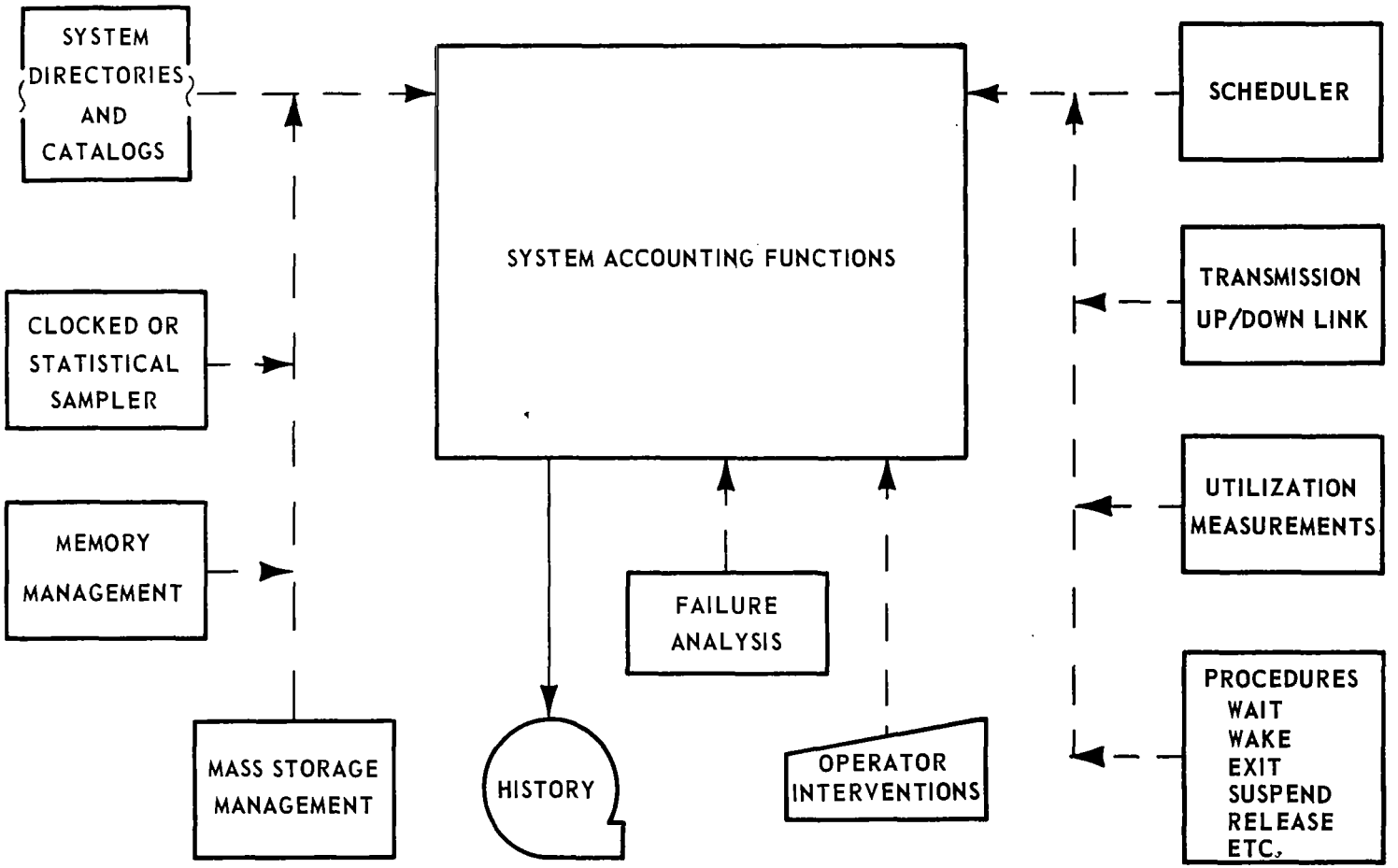

FIGURE 32. A UTILITY FEATURE

There are many graphics functions that could be used to considerable advantage by the astronaut. For example, a simulated perspective application program might be used for docking. During the docking procedure, a program using the linear radar returns, etc., could present a display that would appear to come from a camera perpendicular to the docking procedure. In essence, the astronauts would be standing outside the spacecraft and observing/controlling the docking maneuver. These application programs would make extensive use of the graphics utility package.

SECTION V. CONCLUSIONS AND/OR RECOMMENDATIONS

The report has described a computer executive functional design concept for the Space Station comprising a high-level computer Executive including control of scheduling, allocation of resources, system interactions, and real-time supervisory functions.

The approach accommodates most of the currently known system requirements but additional detailed analysis and simulation will, of course, be required to define and confirm actual design. The following conclusions and/or recommendations may prove helpful in directing further development:

- The flight computers should comprise identical basic components to enhance spares, maintenance, training and financial aspects through broad commonality.

- The features of the multiprocessor interaction should receive trade study attention and interactive capabilities should be exploited within safe limits.

- A viable, dynamic GNC computer system utilizing three ALUs operating synchronously to insure correct computations, identification of failures, and continuity of operation with an Executive to support the GNC functions is recommended.

- The Biomedical Computer will comprise a dedicated simplex computer interfaced to the data bus for the integration of sensors, data acquisition and signal conditioning, biomedical experiment equipment, computer, and investigator-astronaut (medical doctors). This computer will provide a laboratory data management tool utilizing standard formats for data presentation, automatic logging, retention of test results, statistical comparisons, calibration summaries for equipment in use, and monitor methodology to warn of nonuniform procedures.

- Data distribution will be performed under computer control through the data bus controller, the data bus, digital terminals, and remote data acquisition units.

- Remote data acquisition units (RDAU) are interfaced to the digital terminals. Each unit will accommodate up to 64 analog and/or 8-bit signals; thus, a digital terminal may ultimately interface up to 512 inputs to the data bus. The RDAU is highly self-contained, process control oriented and contains memory as well as self-test features.

- The Space Station will have no periods when 100% performance of all computer systems is required; therefore, if critical functions are identified and planned for by partitioning, a lower level of reliability can be tolerated in non-critical areas. For the GNC, an increased reliability and a functional use for spares is recommended.

- Executive functions will perform operations using the primitives imbedded in allied coding. System primitives are defined with emphasis upon the concept of process and process control. Procedures are also defined, in context, which yield Executive access to users.

- The DMS Scheduling Strategy requires flexibility, response to frequent interruption, and a "flat structure" arrangement. The interactive consoles and display devices support man/machine interaction, timesharing, and program preparation/modification.

- The Experiments Scheduling Strategy suggests a free-standing laboratory environment. The experiments are usually prescheduled from the timeline by the DMS. The experiments multiprocessor schedules operations from the DMS list and CALLs from individual experiment supervisors. Prime scheduled requirements will include data acquisition, analysis, and reduction procedures.

- The GNC Scheduling Strategy is to provide a list of processes to be run, foreground/background designator, and a begin execution time. These items are defined by the maneuver to be executed as described by the timeline. The GNC computer then schedules from the DMS list and event driven sequences required in individual maneuvers.

- The DMS system will operate on a time-switched basis to provide timesharing capabilities for computations, program preparation, and interactive analysis. Dispatching

algorithms are described to share the processing power to maximize multiprogramming while minimizing overhead and seizure of the ALU during high computation periods.

- An interrupt type procedure (or trap) is used to interface the software processes to the SVC Interpreter. There is interaction directly with the process and directly with the process control block to monitor the validity of requests.

- The data bus requirements indicate that a polling, interrupt, or oversample technique will be employed. The Data Bus Controller will be programmable and devices will be added to or removed from the bus. Device characteristics will change according to the progression of experiments and the processing of data; therefore, the bus control must be dynamic and flexible.

- The I/O Control primarily governs the allocation of access to modem frequencies and unit controllers. A feasibility study is recommended for priorities of incoming lines via software, recognizing interrupts, queuing or stacking them, arm/disarm/acknowledge, and other facets of interrupt processing. Considerable complication may result from overdependence either upon interrupts or timed occurrances.

- A functional simulation of the IMS configuration and Executives is recommended. Such a simulation will support the detailed design, permit investigation of work load and emergency load fluctuations and provide an evaluation and validation benchmark. The results obtained from a functional simulation should form the basis for detailed design specifications. Analytic simulations from the detailed design documents will produce data for validation in the functional simulators.