NASA CONTRACTOR REPORT

NASA CR-1446

STUDY OE SPACEBORNE MULTIPROCESSING

PHASE I

by Louis J. Koczela

Prepared by
NORTH AMERICAN ROCKWELL CORPORATION
Anaheim, Calif.
for Electronics Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • FEBRUARY 1970

STUDY OF SPACEBORNE MULTIPROCESSING

PHASE I

By Louis J. Koczela

Issued by Originator as Report No. C6-1476.10/33

Prepared under Contract No. NAS 12-108 by
NORTH AMERICAN ROCKWELL CORPORATION
Anaheim, Calif.

for Electronics Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

For sale by the Clearinghouse for Federal Scientific and Technical Information
Springfield, Virginia 22151 – Price $3.00

FOREWORD

The study described in this report was performed by Data Systems Division of Autonetics, a Division of North American Rockwell Corporation, Anaheim, California. The work was done under NASA Contract NAS 12-108 with Mr. F. Hills, Electronics Research Center, Computer Research Laboratories, Cambridge, Massachusetts, as the NASA project engineer.

The study began in March 1966. The contract participants included:

L. J. Koczela – Principal Investigator
A. O. Williman
G. J. Burnett
F. H. Fowler
J. S. Hirsch
R. A. Hokom

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS (CONT)

ILLUSTRATIONS (CONT)

TABLES

STUDY OF SPACEBORNE MULTIPROCESSING

Louis J. Koczela
Autonetics, A Division of North American Aviation, Inc.
Anaheim, Calif.

SUMMARY

This final report presents the results of a research study of multiprocessing computer organizations and their application to future space missions. A manned Mars lander mission in the 1980 time period was investigated and computer requirements defined. Three multiprocessing computer organizations were developed: the multicomputer, the modular multiprocessor, and the distributed processor. An evaluation of the three organizations resulted in the modular multiprocessor as the optimum candidate for the selected mission; this organization was then subject to a detailed design investigation.

I. INTRODUCTION

The purpose of this study was to investigate multiprocessing computer organizations and their application to future space missions. A block diagram of the study approach is given in Figure 1-1. As a base for the study, manned space missions in the 1980 time period were selected to define the computational requirements. The particular mission selected for a detailed investigation to define the requirements was the manned Mars landing mission. This mission covers a broad spectrum of requirements (long duration, widely varying computational loads, and high reliability demands); therefore, using it as a base will result in applicability to many other missions in the same time period such as extended earth orbital space stations.

The selected mission was investigated in detail to define the requirements. This effort is covered in Section II of this report. Based on an analysis of the mission and computational and data processing functions, the computer requirements were defined for each phase or mode of the mission.

In addition to defining the requirements, it was necessary to define the technology to be considered as state-of-the-art for the time period of interest before proceeding with an investigation of multiprocessor configurations. Although the time period of the missions considered is 1980, it is necessary to use technology that will be available for designers in approximately 1975, so that reliability has been established. The investigation and definition of the technology base is given in Section III.

Multiprocessing organizations are considered to offer considerable advantages in application to future manned space missions. These organizations can result in: (a) efficiently meeting the widely varying computational loads of different phases of a mission, (b) efficiently mechanizing the diverse requirements of various subsystems of a mission such as a command module and a lander module, (c) an overall net reduction in power due to the ability to turn modules on and off, (d) increase in reliability, given that failure rates of dormant equipment are lower than operating equipment, and (e) enhancement of probability of mission success and availability due to reconfiguration around failures at a low module level.

Using the requirements and technology defined as a base, three organizational approaches to multiprocessing were investigated. These were (a) multicomputer, (b) modular multiprocessor, and (c) the distributed processor. The general organizational features such as word length, instruction format, were evaluated and traded off. Each of the three organizations was subject to a preliminary logic design, a failure analysis, and a software analysis. This topic is treated in Section IV.

Using the results of the preliminary design a simulation (reliability) and critical evaluation of the three candidates was performed (Section V). From the results of the evaluation the modular multiprocessor was selected for further investigation.

In order to evaluate the candidate multiprocessing organizations, the computer system characteristics were weighted by NASA ERC in terms of relative importance as follows: (a) Computer Probability of Mission Success - 100, (b) Power - 10, (c) Growth Potential - 4, (d) Development risk - 1, (e) Weight - 1, (f) Size - 1, (g) Cost - 1.

Extensive investigation was performed on the selected organization in terms of design, failure analysis and software considerations, these topics are covered in Section VI.

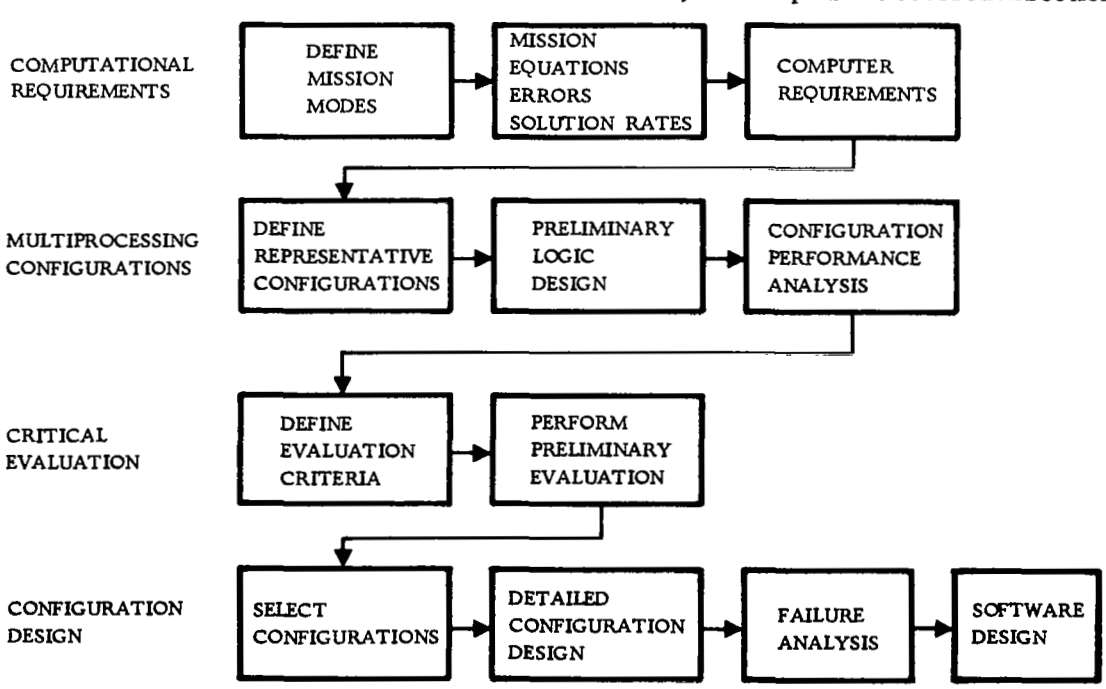


Figure 1-1. Block Diagram of Study Approach

II. COMPUTER REQUIREMENTS

2.1 INTRODUCTION

In the study of Multiprocessing Systems, a problem immediately arises as to the meaning of several terms. Since the extended use of more complex structural computers is rather recent, the terms associated with them have not become firm. A great deal of confusion arises in reading reports and in verbal communication if the meaning of terms is not clear; therefore, further definitions shall be given. The following is an attempt to define some common terms in accordance with the most common usage. All use of the terms in this report will be consistent with the definitions.

Availability (operational)

The probability that a system or equipment when used under stated conditions and in an actual supply environment shall operate satisfactorily at any given time. It may be expressed as:

$$A_o = \frac{MTBM}{MTBM + MDT}$$

where

MTBM = mean time between maintenance and ready time during the same interval

MDT = mean downtime

Availability (inherent)

Similar to A_o except relates to ideal supply environment and does not consider scheduled or preventive maintenance.

$$A_i = \frac{MTBF}{MTBF + MTTR}$$

Backup

Refers to a function or hardware not involved in a primary mode, function, or task which will be used in case of failure of the primary.

Cellular Arrays

An arrangement of computational cells (generally rectangular), all perform basic logic or arithmetic operations and can derive inputs and send outputs to each of its neighbors. (The information is generally transmitted through the array in parallel with operations being performed on a bit per cell per operation basis.) One form of distributed logic.

Distributed Logic

The decentralization of the logic elements on an array basis. Each element (cell) of the array can communicate with a number of other cells. Each of the cells has some memory associated with it. The complexity of each cell can vary from the execution of a few logic operations to a (small computer.) The control for execution of a program is distributed among the cells.

Iterative Array

Synonymous with "Cellular Array."

Iterative Circuit Computer

Synonymous with "Distributed Logic."

Maintainability (Ref: MIL-STD-778, 8 April 1964)

Maintainability is a characteristic of design and installation which is expressed as the probability that an item will conform to specified conditions within a given period of time when maintenance action is performed in accordance with prescribed procedures and resources.

Microprogramming

Computer control mechanization wherein the instructions are handled as macros. Each macro is interpreted in terms of micro-operations by either programmed or modifiable logic (core or diode memory). The micro operations are the basic instruction set of the computer and are defined in terms basic, logic, shift, and transfer operations. (Same as "stored logic.")

Multicomputer

Two or more computers with intercommunication which operate on one or more programs. The computer implied consists of an arithmetic unit-memory-and input/output unit.

Multiprocessing

Simultaneous execution of two or more programs or sequences of instructions by a multipath structure.

Multiprocessor

A computer capable of multiprocessing (multiple arithmetic units, memories, and input/output units with versatile communication is one hardware approach). Cellular arrays is another.

Multiprogramming

Interleaved execution of two or more programs by a computer complex.

Parallel Processing

Simultaneous execution of two or more sequences of instructions (generally branches of same program) by a computer having multiple arithmetic or logic units.

Probability of Mission Success

Probability that mission objectives are attained.

Reconfiguration

Changing pieces of hardware performing a function. This may be manual or automatic and may be performed as a result of system failure or change in mission mode.

Redundancy

Additional time, computation, or hardware used above the basic requirements of a function so that a required probability of success of that function can be attained.

> Functional Redundancy - Use additional or backup functions.
>
> Active Redundancy - Techniques which sense faults, isolate them and switch out or replace failed equipment.
>
> Passive Redundancy - Faults are not detected, they are masked by extra equipment. Defective equipment remains in place.

Reliability (RETMA definition)

Reliability is the probability of a device performing its purpose adequately for the period of time intended under the operating conditions encountered. Measure commonly used is Mean-Time-Between-Failure (MTBF).

Repair

The process of returning an item to a specified condition including preparation, fault location, item procurement, fault correction, adjustment and calibration, and final test.

> Active Repair Time - The time during which one or more technicians are working on the item to effect a repair.
>
> Mean Time to Repair (MTTR) - The statistical mean of the distribution of times-to-repair. The summation of the active repair times during a given period of time divided by the total number of malfunctions during the same time interval.
>
> Repairability - The capability of an item to be repaired.

Self-Organizing

Processes employing neural network type redundancy. System reorganizes around faulty modules or cells. (Infers random type techniques.)

Self-Repairing

A self-repairing system is one which has the capability to continue to work correctly, even if some of its elements malfunction.

> In high redundancy techniques errors may be masked by voting techniques. In lesser redundant techniques errors must be detected, isolated, and then the system is reconfigured around the fault by changing its mode of operation.

Time Sharing

Time multiplexing of several users on a computer. This will, in general, require processing of different programs for each user. No restrictions on type of computer are implied by time sharing definition.

The manned Mars landing and exploration mission was selected as representative for application of spaceborne multiprocessing techniques. The establishment of the mission requirements is a necessary first step in the process of developing an appropriate multiprocessor concept. A number of studies relating to manned Mars missions have been conducted throughout industry under numerous NASA contracts. The documented results of these studies were reviewed with the objective of establishing the particular mission requirements which would influence the on-board computational and data processing facility. By taking full advantage of related studies, a realistic appraisal of mission requirements was obtained with a minimum of delay in the conduct of the multiprocessor study. The following discussion, as it relates to the manned Mars mission description is based primarily upon information provided in references 1 through 11.

2.2 MISSION PROFILE

The selected manned Mars mission covers a 420-day period and consists of three primary phases, i.e., Trans Mars, Mars Stay and Trans-Earth. The following is a brief description of each phase.

2.2.1 Trans-Mars

The trans-Mars phase lasts approximately 120 days and begins after the space craft is placed in an Earth orbit and checkout of all subsystems is completed. The first operation is an Earth to Mars injection maneuver. The trajectory is then determined utilizing Earth tracking facilities in addition to the on-board system to accurately determine the trajectory errors and the required corrections. The next operation is the selection of the desired spin plane and subsequent spin-up. This establishes the artificial gravity environment required for a major part of the mission duration. Navigational fixes are made as the mission proceeds with velocity corrections performed when necessary. The spacecraft is de-spun about five days prior to Mars arrival. It is then lined up for proper altitude and attitude into the entry corridor to perform the aerodynamic braking maneuver. A circular Mars orbit is obtained by applying a velocity increment utilizing the mid-course propulsion system.

2.2.2 Mars Stay

During the Mars stay period, which lasts about 40 days, three major operations occur. These are: separation of the Mars Excursion Module (MEM) probably during the first day in orbit; rendezvous and docking of the MEM with the mission module, about the last day in orbit; and transfer of scientific samples and equipment from the MEM to the mission module. The MEM is then abandoned in Mars orbit where it may, using automated sensors, continue to measure and transmit data to Earth. Communications with the MEM and Earth as well as scientific observation of Mars is continuously maintained by the mission module during the Mars stay period.

2.2.3 Trans-Earth

This phase lasts about 260 days. The first operation, following navigational determinations of exit trajectory and launch time, is the Mars to Earth injection maneuver. Navigational fixes are performed for the next several days and corrections carried out utilizing the mid-course stage. As the trajectory is finalized, the spacecraft is spun up. Navigational fixes are periodically made to provide corrections during spin coast. Approximately five days prior to Earth arrival the spacecraft is de-spun. The final entry trajectory is determined and corrections made. About two days to as little as three hours before arrival, the crew enters the Earth Re-entry Module (ERM) and separates from the mission module.

2.3 MISSION OBJECTIVES

The primary objective of the manned Mars mission is exploration of the surface of the planet Mars to develop knowledge concerning its composition, structure and life forms. Exploration of the surface will be carried out by personnel and scientific equipment which will be landed on the planet. The mission module supports the landing party in this effort by providing communications and data processing for the MEM and also by performing cooperative experimentation and observation. Additionally, while in Mars orbit, it can take advantage of its wide coverage of the planet to acquire data beyond the exploration radius of the landing party.

Although Mars exploration is the primary objective, only ten percent of the overall mission time is spent in the Mars area. Consequently, there is considerable experimentation and observation activity carried out during the trans-Mars and trans-Earth phases.

2.4 MISSION FUNCTIONS

There are three major functions that must take place throughout the entire mission in order to assure successful accomplishment. These mission functions, so called, because they are of a broad enough nature not to be included within the subsystem functions but rather establish the requirements for subsystem functions, are:

1. Life support (Crew Survival)
2. Command and Control
3. Scientific Experimentation and Exploration

The life support functions assure the physical and psychological health of the crew. These functions would include physiological and psychological testing, health preservation activities, radiation protection, rescue operations, and checkout operations prior to major maneuvers. It is not likely that the computational and data processing complex will be essential to the critical crew survival aspects of the life support function. However, the computer will be used to perform tasks necessary for the testing and checkout operations, which are essential to overall mission success.

The command and control function is concerned with monitoring operations, control of subsystems, control of interfaces, abort decision and control, command locations (internal and external repair crews, and MEM crew). The major components of command and control include spacecraft guidance and control, telecommunications, crew displays and controls, power distribution and overall mission module interfaces. A simplified interface diagram of the command and control function is shown in Figure 2-1. Through computation, data collection, storage and display, the command and control function provides appropriate orientation and sequencing command signals to major subsystems and informs the crew of subsystem operation, consumption rate and storage level of storables, navigational position, spacecraft attitude, and antenna and instrument orientation. It also records important information on subsystem malfunctions, command messages and mission history.

The scientific experimentation and exploration function is closely related to the mission profile. For example, on arrival or departure from the planets (Mars and Earth), the main experiments are those connected with planetary observations. During the trans-Mars and trans-Earth phases, the experimental effort is concentrated on the interplanetary bodies and solar physics. The interplanetary environment is continuously monitored.

The crew members are an integral part of the scientific subsystem and therefore, will participate in the preparation and operation of certain experiments. They will program times for data acquisition, assist in observations, and reduce some data prior to transmission. Some operations will be automated but with provisions for manual data check and override.

A simplified interface diagram illustrating the data handling associated with the scientific experiment and exploration function is shown in Figure 2-2.

2.5 SPACECRAFT SYSTEM DESCRIPTION

2.5.1 Spacecraft Configuration

The major elements of the spacecraft, exclusive of propulsion units, are the Mars Mission Module (MMM), the Mars Excursion Module (MEM), and the Earth Re-entry Module (ERM). All three modules will contain subsystems which require computational and data processing support. For conducting the multiprocessor study, emphasis has been placed upon the MMM. However, the necessity for compatibility between subsystems throughout the overall spacecraft, makes the multiprocessor concept for satisfying computational and data processing requirements applicable to all three major modules.

2.5.2 Major Subsystems

There are six major subsystems which require varying degrees of computational and data processing activity and consequently exert the primary influence on

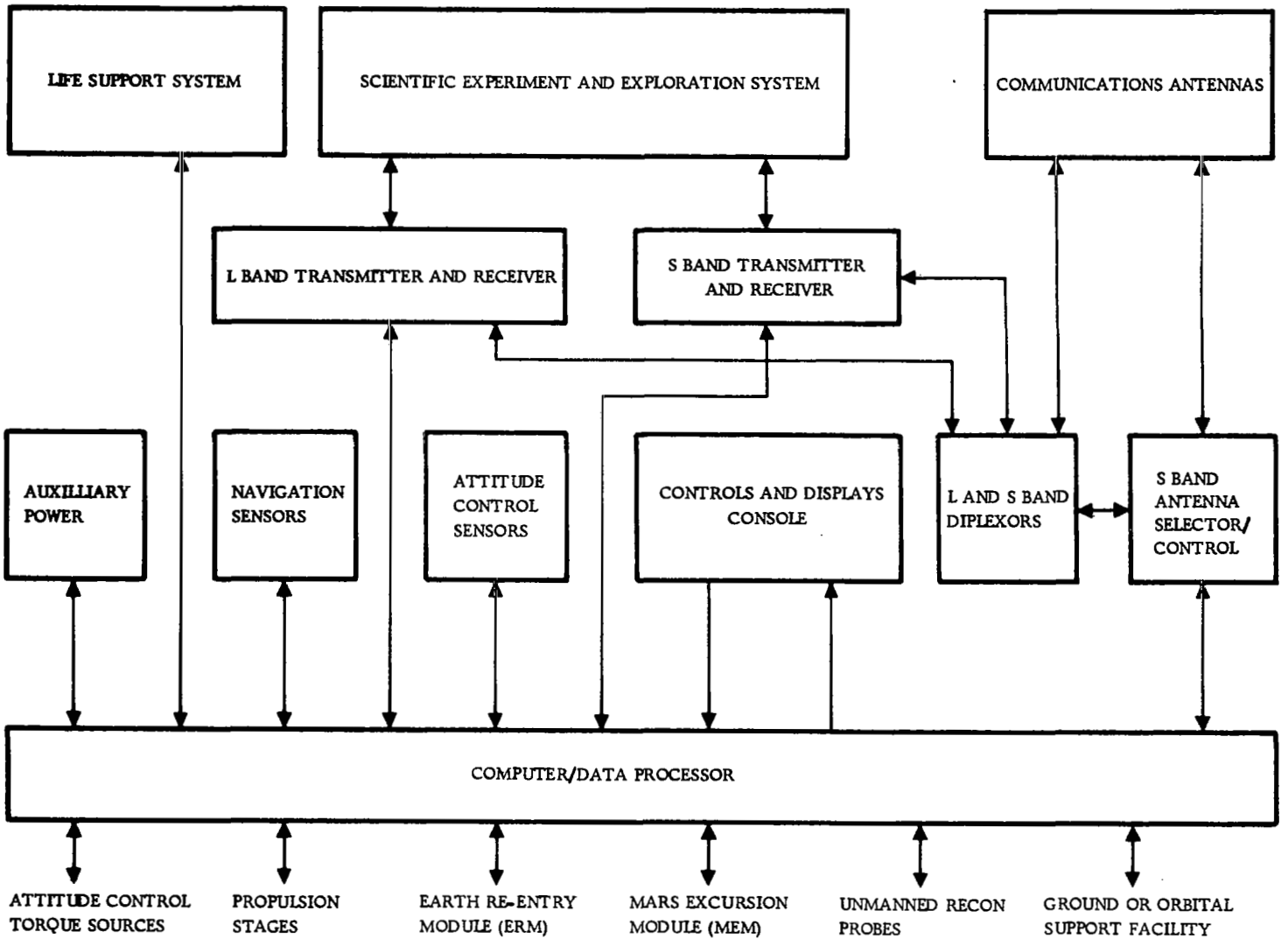


Figure 2-1. Command and Control Function Interface Diagram

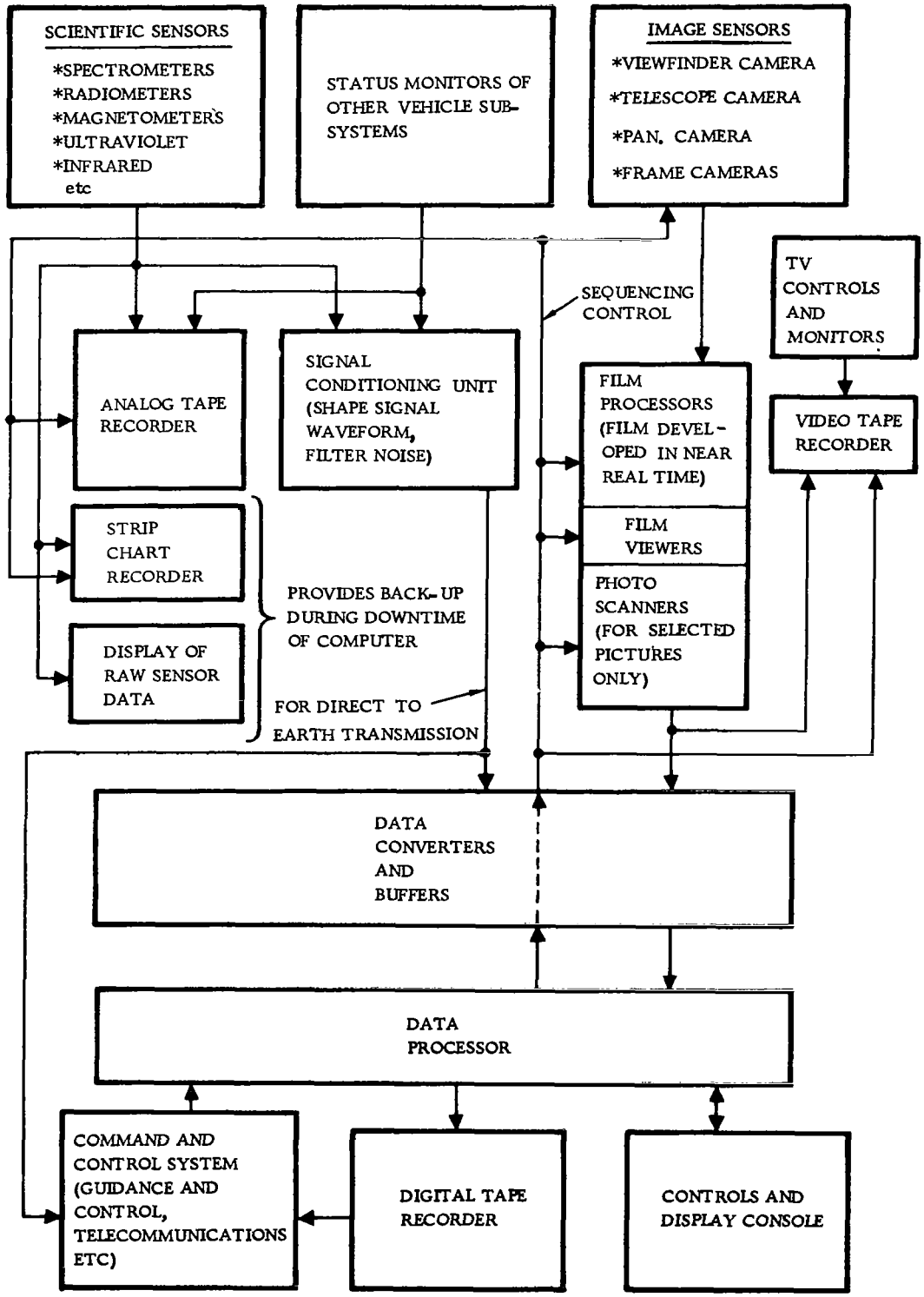


Figure 2-2. Scientific Experiment and Exploration Function Data Handling Interface Diagram

requirements for the on-board computer system. These subsystems are: (1) Guidance and Navigation, (2) Attitude Control, (3) Telecommunications, (4) Scientific Sensor Experiments, (5) Reconnaissance, and (6) Life Support. A simplified interface block diagram which identifies the major components of each subsystem is shown in Figure 2-3. The following paragraphs are brief discussions of the functional requirements of each subsystem.

2.5.2.1 Guidance and Navigation

The guidance unit of any system concept must supervise the flight according to a flight plan. In doing so, the guidance unit is required to generate and issue commands to the attitude control subsystem, propulsion subsystem, and communication subsystem. The navigation unit is required to determine all kinematic variables compatible with the functions performed by the guidance unit. In general, these kinematic variables are position, velocity, acceleration, attitude, angular velocity, angular acceleration, and time. The variables are determined with respect to a specified reference frame.

The guidance and navigation subsystem, for utilization throughout the manned Mars mission, must be able to determine the kinematic state of the spacecraft at all times during the mission and through comparison with the required kinematic state at that moment, as derived from targeting data, be able to generate suitable commands for the velocity-to-be-gained required for the correction of any navigational errors. As shown in Figure 2-3, the main elements in the subsystem are a stable platform inertial measuring unit (IMU), a scanning telescope, a sextant, and a ranging sensor (radar).

2.5.2.2 Attitude Control

The attitude control subsystem operates in conjunction with the guidance and navigation subsystem to provide: (1) angular orientation and stabilization of the spacecraft about three axes, (2) translation control during rendezvous and docking maneuver, and (3) thrust vector control during coast corrections.

The major elements of the attitude control subsystem are shown in Figure 2-3. The body fixed accelerometers and gyros also provide a back-up inertial measuring unit (strapdown) for the guidance and navigation subsystem. The three-axis rate gyro package contains the sensing elements and associated circuitry required to provide angular rate stabilization control and display information. The horizon sensor senses the location of local vertical with respect to the body reference while the sun sensor senses the line of sight (LOS) of the sun with respect to the body reference. The astrotracker senses the LOS with respect to selected stars.

The propulsion engines and reaction jets are also included as a part of the attitude control subsystem. The attitude and translation commands are combined, decoded and converted to jet selection signals which activate the reaction jet control valves. Similarly, for propulsion engine control, attitude commands are converted to engine deflection commands.

2.5.2.3 Tele-communications

Each of the three modules (MMM, MEM, ERM) making up the manned Mars spacecraft will have its own communication functions to perform, with the added

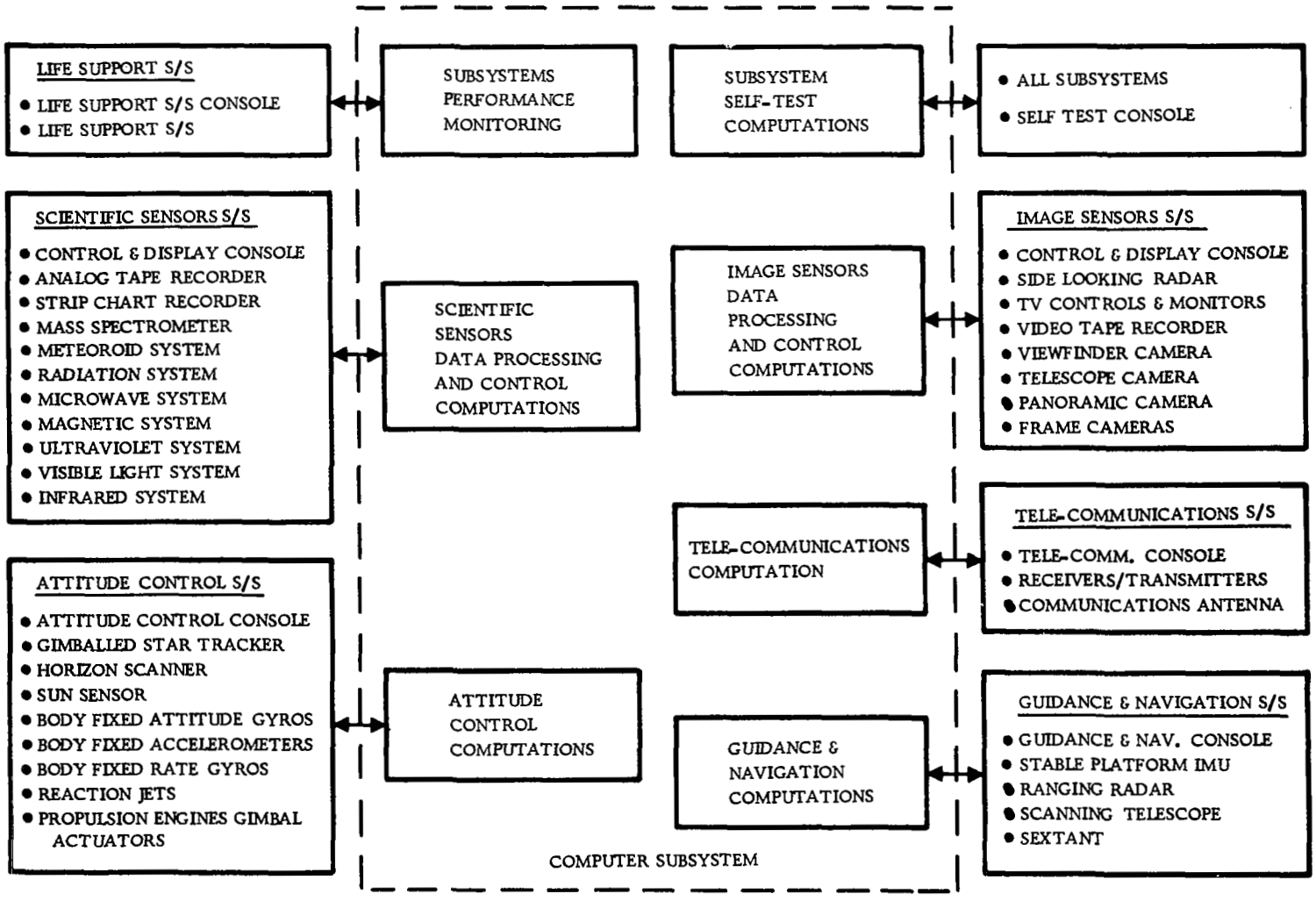


Figure 2-3. Subsystems - Computer Interfaces

provision that each system be compatible with one another whenever operations so require. The tele-communications subsystem has the following basic types of functional requirements:

1. Voice - Two-way voice communication capability between the individual crew members, between the three modules and between the spacecraft and Earth should exist at all times.

2. Telemetry - The telemetry system is required to transmit measurements related to the engineering status of the on-board systems, crew status concerning psychological and physiological data and scientific experimentation data.

3. Television - Television transmission is part of the spacecraft's communication system and serves the dual purpose of monitoring scientific data and furnishing public information.

4. Data Processing - The large number of engineering and status measurements which must be processed, analyzed and transmitted, require on-board data displays, data storage and data control.

5. Tracking - This function provides near Earth and Mars tracking capability and aids in the recovery of the crew at the end of the mission.

6. Rendezvous - Upon completion of the Mars surface exploration, the MMM and MEM must accomplish a rendezvous and docking maneuver which requires knowledge of range, range rate and bearing measurements to the target vehicle.

2.5.2.4 Scientific Experimentation

As shown in Figure 2-3 and Table 2-1, the measurement techniques for scientific experimentation include microwave, infra-red, visible-optical, ultra-violet, radiation, magnetic field, meteoroid and mass spectrometry. Table 2-1 shows the relationship between the scientific experimentation subsystem function, the mission profile and the measurement techniques.

2.5.2.5 Reconnaissance

As shown in Figure 2-3, the reconnaissance or image sensor subsystem consists of photographic and optical sensors. Employment of such sensors in the manned Mars spacecraft is designed to make full use of man's capabilities. The crew members are most effectively used in the collection, processing, handling, selection, and transmission of imagery data.

The major subsystem elements include television, viewfinder camera, telescope camera, panoramic camera and stereo high resolution frame cameras. The television system is intended to serve a backup to the photographic cameras and to provide a real-time view of the Mar's surface. The viewfinder provides a visual reference to the area which is mapped by the cameras. It also is used as a pointing and tracking aid for the high resolution telescopic camera. The panoramic camera provides a maximum of ground coverage with equal angular resolution at every scan

Table 2-1. Scientific Experimentation Subsystem Functions

Measurement Technique	Depart and Arrive Earth	Trans-Mars	Arrive and Depart Mars	Mars Orbit	Trans-Earth
Microwave Spectroscopy	Absorption spectrum of Earth; thermal emission spectrum. Microwave albedo	Solar microwave emission	Mars absorption spectrum; thermal emission spectrum; albedo	Mars atmosphere and surface absorption and emission spectra	Solar microwave emission
Infra-Red Spectroscopy	Earth absorption and thermal emission spectra. Infra-red albedo	Solar Infra-red emission	Mars absorption and thermal emission spectra. Infra-red albedo	Mars atmosphere and surface absorption and emission spectra. Twilight phenomena. Airglow. Satellite absorption and emission	Solar corona Infra-red emission. Photosphere emission
Visible-Optical Spectroscopy	Earth absorption and emission spectra, albedo	Solar line spectra. Spectra of selected astronomical objects	Mars absorption and emission spectra. Albedo	Mars atmospheric and surface absorption and emission spectra. Airglow. Aurorae. Twilight phenomena	Solar line spectra. Spectra from corona.
Ultra-Violet Spectroscopy	Earth absorption and thermal emission spectra. Albedo	Solar line spectra. Spectra of selected astronomical objects	Mars absorption and emission spectra. Albedo	Mars atmosphere absorption spectra. Aurorae. Spectra of radiation belts.	Solar photosphere and corona spectra
Radiation Spectroscopy	Flux, directionality, energy spectrum of trapped radiation and impinging radiation	Flux, directionality, species, energy spectrum of radiation in interplanetary space	Flux, directionality, species, energy spectrum in Mars neighborhood	Flux, directionality, species, energy spectrum of trapped radiation	Flux, directionality, species, energy spectrum of radiation in interplanetary space
Magnetic Field	Local magnetic field	Magnetic field in space	Aeromagnetosphere	Aeromagnetosphere	Magnetic field in space
Meteoroids	Flux, directionality of meteoroids	Flux, directionality of meteoroids	Flux, directionality of meteoroids	Flux, directionality of meteoroids	Flux, directionality of meteoroids
Mass Spectrometry		Interplanetary gas composition, Micrometeoroid composition		Mars upper atmosphere composition	Interplanetary gas composition, Micrometeoroid composition

angle so that complete planetary coverage can be obtained in a minimum time. Two frame cameras provide high-resolution stereo pictures of the planet's surface by viewing the same surface area from two different aspects.

2.5.2.6 Life Support

The life support subsystem is one of the most critical subsystems in the MMM. The necessity for maintaining a habitable environment for the whole 420 day mission leads to exacting fail-safe operation, reliability and maintenance requirements. Interface with the computer complex, if any, would be through the instrumentation, controls and displays required for the environmental control system. The implementation of a digital computer in life support appears to be limited to the monitoring of direct indicating instrumentation as illustrated in Figure 2-3. However, it is possible that use of the computer for life support control functions, rather than individual controller electronic "black boxes," will become feasible.

2.6 COMPUTATIONAL AND DATA PROCESSING FUNCTIONS

2.6.1 General

Three primary mission functions were described previously. These functions require computer operations that fall into two general categories. The first is command and control computation, which obviously relates to the command and control function. The second group is mission data processing, which is primarily concerned with the processing of the scientific experiments and exploration data but also includes some limited processing of data from the life support system.

The type of computer operation required for command and control is decidedly different from that employed for mission data processing. Consequently, any multiprocessing computer system should logically evolve from this natural separation of computational tasks.

In order to investigate varying degrees of multiprocessing and also establish their respective computational requirements, it is necessary to partition further the two major computational and data processing categories. The subdivision of functions through four levels is illustrated in Figure 2-4. The guide line for partitioning functions is not limited simply to separation according to computational characteristics but is also influenced by failure protection objectives. Such factors as the ability to implement alternate modes of operation as well as the opportunity for maintenance and repair, also have to be considered.

2.6.2 Computer Operations

In paragraph 2.8, relating to computer requirements, the individual functions and their respective requirements are discussed in some detail. It is the intention here to describe in a general manner the computational and data processing tasks that are required of the computer.

Processing for guidance, navigation and control is primarily computational in nature. Input data from the various inertial and optical sensors are used to compute location, orientation and directional acceleration of the spacecraft. When combined with previously processed data from the same sources, the past and predicted flight path and velocity of the vehicle is computed. This path is compared with the pre-established desired trajectory to determine deviations and drift rates. Should an

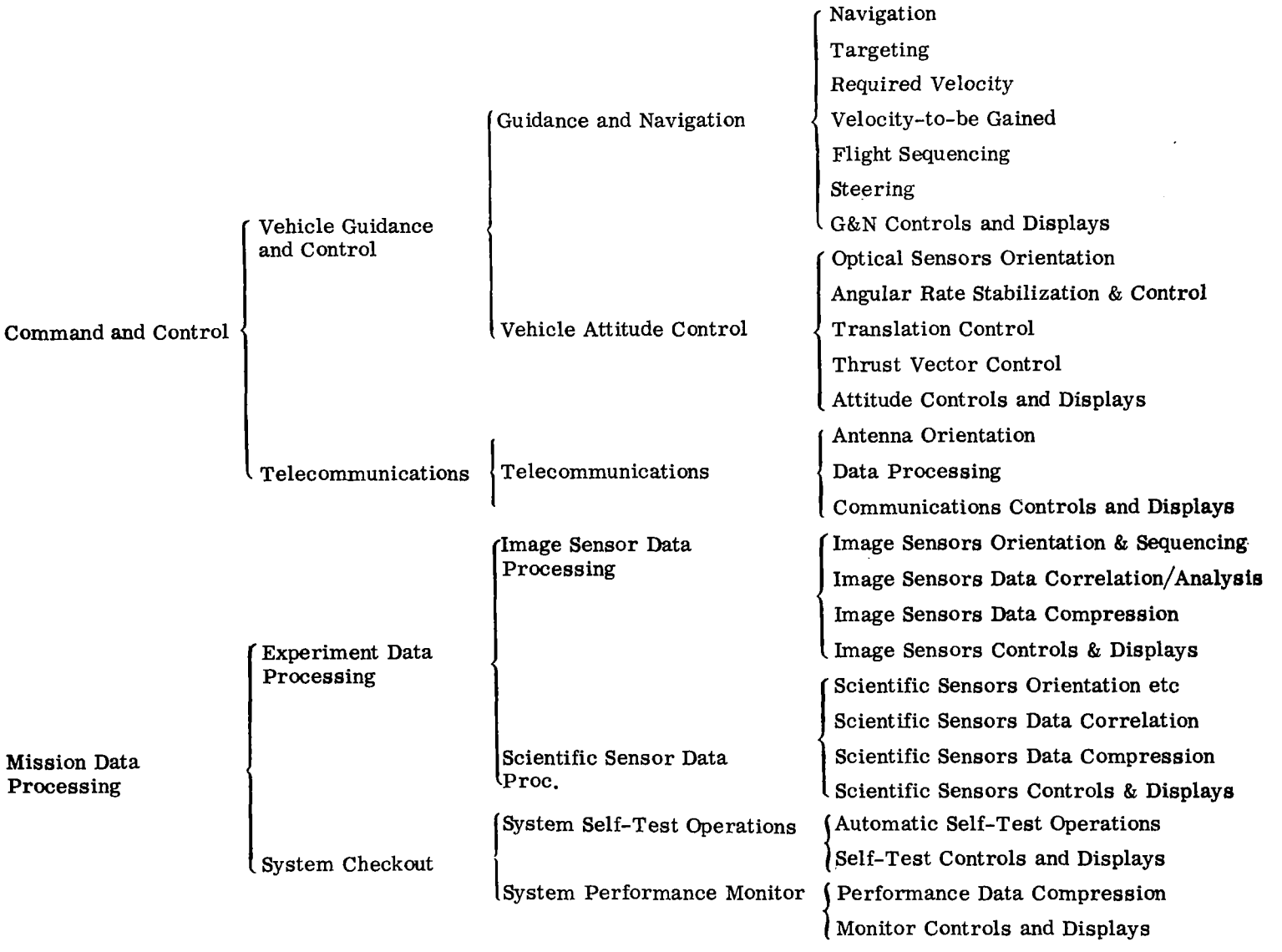


Figure 2-4. Computational and Data Processing Functions

alteration of the predicted path be required, the optimum method of achieving the desired path, velocity components, and/or implementation of the spin up and de-spin maneuver, is computed.

The computation task most characteristic of guidance, navigation and attitude control processing is state (position and velocity) estimation based upon statistical filtering. The process consists of using a statistical filter to optimally estimate the components of the vehicle state vector from a sequence of measurements made by the imperfect instruments on board the spacecraft.

In general, the estimation method works as follows:

In anticipation of the i^{th} group of observations and the i^{th} estimation cycle, the computer integrates the equations of motion from the previous best estimate of the state at the time of the i^{th} estimate. The estimator K is determined by use of the expected state and covariance matrix of estimation error corresponding to the time of the i^{th} estimation cycle. The expected values of the observables (space angles) are computed. The observations are made from the spacecraft actual position and as such contain the sensor error. The computed space angles are subtracted from the observed angles. The residuals are operated upon by the estimator which produces an optimal estimate of the deviation between the actual and expected spacecraft state. This estimated deviation is added to the expected state to form a new, corrected set of initial conditions for the next integration cycle. Finally, the covariance matrix of estimation error is corrected to reflect the latest estimation.

Optimal filtering techniques may also be applied to attitude determination. Because of long term gyro drift characteristics, devices such as sun sensors, horizon sensors, and star trackers are needed to provide long term attitude reference. However, these instruments are not sufficiently accurate to provide the precision attitude information needed. The recursive optimal filtering technique is one of several methods available for utilizing the observables to update knowledge of vehicle attitude. Others include least squares and partial correction. The latter two methods require less computational complexity but at the expense of attainable accuracy.

An additional computation of special note is employed in the event that strapdown inertial sensors are used for either primary or backup navigation. The strapdown system uses body fixed gyro displacement signals in a high speed direction cosine computation which effectively simulates the gimbal system in a stable platform inertial mechanization.

Processing for tele-communications is concerned primarily with reduction of the volumes of data received from numerous sources to the minimum quantity consistent with maintaining the integrity of the information. This is accomplished through a variety of data sampling and compaction techniques. Data processing aids in the communications task of determining transmission times and durations, power requirements, antenna boresight calculations, and the information content of specific transmissions. This determination is based upon considerations of possible interference from intervening bodies, predicted power and equipment utilization requirements. These factors are of particular concern during the return to Earth phase of the mission or in the event of degradation in communication facilities.

Computer operation in support of the scientific experimentation and exploration function consists primarily of data handling and data processing. Its objective is to optimize the flow of information from the data gathering sensors to the crew and to the communication subsystem. A large portion of the data to be processed is in the form of imagery. It is likely that a significant saving in transmission time can be realized, without loss of information or increase in communications bandwidth, by application of data compression techniques.

In addition to data compression techniques, transmission time can be reduced by the avoidance of unnecessary overlap or duplication in the photographic imagery. The computer can be used to control the image collection automatically thus relieving the crew of the task of keeping track of picture overlap or duplication. With the aid of navigational data, the computer schedules the operation of the cameras and insures collection of precisely the right amount of duplicate imagery.

The processing associated with the scientific experiment sensors includes sensor operation control, data compression and data analysis. The control functions pertain to computer selection and sampling of inputs according to some programmable criteria as in the following examples: (1) certain sensors or groups of sensors are activated according to the mission phase, (2) the indications from an active sensor may call for the use of an otherwise inactive sensor, (3) the selection of the different input samples from the same sensor can vary during the course of the mission or according to the status of the sensor, and (4) the sampling rate of each active sensor can be varied according to such criteria as the amount of change in the data magnitude, the relative change when compared to other sensor readings, the maximum allowable data storage rate or data transmission rate, a priority basis or when the readings pass through maximum and minimum conditions.

As in the case of the image sensor data processing, data compression techniques are used by the computer to reduce the vast amounts of data being collected. In addition to reducing the data through compression methods, the computer can conduct an analysis of the contents of the data. This is normally done when the results of the analysis is required for on-board operation and evaluation by the crew. An example of this type of processing by the computer, is the comparison of a multicomponent spectrum, as detected by an active sensor, to a pre-stored series of reference spectra. Various curve fitting and correlation procedures can be used in order to establish the best match and identify the elements in the sample. Correlation and matching between sensors can also be used as an aid to automatic instrument calibration.

Performance monitoring, and in some cases the actual testing of subsystems, is a significant data processing function. The objective is the immediate detection of sub-standard performance on the part of any major element in each subsystem. The monitoring task requires simply the sampling and recording of performance data for purposes of display or telemetry. Some data compaction may be used in order to reduce the data transmission load. Comparison of test point measurements against pre-stored tolerance limits is perhaps the most prominent test under computer control. However, a number of rate tests and cross-checks, may also be conducted, all of which help in establishing the operating status of each subsystem.

A more sophisticated utilization of the computer takes place in the context of an on-board checkout system. Here, the computer, operating in conjunction with special purpose equipment, would control checkout operations such as: (1) selection and control of function generators, (2) selection of stimulus and measurement points,

(3) selection and control of measurement devices, (4) timing and sequencing of the stimuli and measurement signals, (5) comparison of observed with expected results, (6) setting or sensing of the state of the system under test, and (7) communication with the crew.

2.7 MISSION-FUNCTION TIME LINE PROFILE

The Mars mission flight profile defined previously indicated a mission time to Mars of 420 days. This imposes a duty cycle, for some on-board equipment, of over 10,000 hours for a flight initiating from Earth orbit. To this must be added ground checkout time and operating time while in Earth orbit.

In order to provide a base for defining computer requirements as a function of time and assess the computer reliability requirements, but without constraining the computer configuration, a mission-computer function time line profile was generated. The results are presented in Table 2-2. It can be seen from the profile that many of the functions are expected to operate in excess of 10,000 mission hours. In the case of guidance and navigation it appears that advantage can be taken of shut down periods. During trans-Mars and trans-Earth coasting phases, which make up about 80 percent of the total mission time, the guidance and navigation functions are active only for short periods of time (for computing trajectory corrections) and computations are made at widely spaced intervals of time. Most of the remaining functions are active throughout the entire mission time, although for some it is not necessary that the computing be done continuously.

2.8 DETAILED COMPUTATIONAL FUNCTIONS

In the previous sections a description of the system and a general discussion of the computational functions was given. This section will discuss in more depth the computational algorithms associated with the computer functions. The functions will be described in four parts as previously identified, namely: (1) Vehicle Guidance and Control (2) Telecommunications, (3) Experiment Data Processing, and (4) System Checkout. These descriptions of the computational functions will provide the base for defining the computer requirements and also provide insight into the types of computations required.

2.8.1 Vehicle Guidance and Control

The Vehicle Guidance and Control (hereafter referred to as Navigation and Guidance or simply N&G) consists of 15 basic modes:

1. Atmospheric Ascent
2. Earth Orbital Coast
3. Trans Mars Injection
4. Trans Mars Coast
5. Trajectory Correction
6. Spin Up
7. Trans Mars Spin Cruise

Table 2-2. Mission-Function Time Line Profile

* CONTINUOUS ** PERIODIC MISSION PHASES	EARTH AREA (14 DAYS)				TRANS-MARS (120 DAYS)							MARS AREA (40 DAYS)						TRANS-EARTH (260 DAYS)						
	PRE-LAUNCH CHECKOUT	EARTH ORBIT BOOST	EARTH ORBIT CHECKOUT	PRE TRANS-MARS FINAL CHECKOUT	INJECTION	COAST-CORRECTION-COAST	SPIN UP	SPIN COAST-SPIN CORRECTION-SPIN COAST	DE-SPIN	COAST-CORRECTION-COAST	MARS AEROBRAKING	ORBIT INJECTION	MEM CHECKOUT-LAUNCH	MARS ORBIT COAST-CORRECTION-COAST	MEM RENDEZVOUS-DOCK	MEM SEPARATE	PRE TRANS-EARTH CHECKOUT	INJECTION	COAST-CORRECTION-COAST	SPIN UP	SPIN COAST-SPIN CORRECTION-SPIN COAST	DE-SPIN	COAST-CORRECTION-COAST	ERM CHECKOUT-SEPARATE
PHASE TIME (HRS)	240	0.2	20	60	0.2	120	1.2	2640	1.2	120	0.4	0.6	0.9	960	8.0	4.4	1.0	0.2	120	1.2	6000	1.2	120	1.0
TOTAL RUNNING TIME (HRS)	240	240.2	260.2	320.2	0.2	120.2	121.4	2761.4	2762.6	2882.6	2883	2883.6	-	3843.6	-	-	-	3843.8	3963.8	3965	9965	9966.2	10,086.2	10,087.2
COMPUTER FUNCTIONS																								
GUIDANCE & NAVIGATION		*	**	**	*	**	*	**	*	**	*	*	*	**	*	**	-	*	**	*	**	*	**	*
VEHICLE ATTITUDE CONTROL		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
TELE-COMMUNICATIONS		*	*	*	*	*	-	*	-	*	*	*	*	*	*	*	*	*	*	-	*	-	*	*
IMAGE SENSOR DATA PROCESSING		-	-	-	-	**	-	**	-	**	-	-	-	**	-	-	-	-	**	-	**	-	**	-
SCIENTIFIC SENSOR DATA PROCESSING		-	-	-	-	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	-
SUBSYSTEM SELF-TEST OPERATIONS	*	-	*	*	-	*	-	*	-	*	-	-	*	*	-	-	*	-	*	-	*	-	*	*
SYSTEM PERFORMANCE MONITOR	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

8. De Spin

9. Mars Approach Correction

10. Aerobraking

11. Mars Orbit Injection

12. Mars Orbital Coast

13. Trans Earth Injection

14. Earth Approach Correction

15. Earth Re-entry

This is not intended to imply that there are 15 phases to the mission since some of the basic modes will be entered into several times (e.g. spin up and de spin for trajectory corrections on both trans Mars and trans Earth portions of the mission). A description of the N&G functions for each mode will be given below. These modes will then be translated into computer requirements per mission phase. It should be noted that references 5, 12, and 13 were used throughout paragraph 2.8.1.

2.8.1.1 Atmospheric Ascent

The atmospheric ascent or "boost" will be primarily controlled by the booster guidance system. However, it is possible that the spacecraft computer will have access to the information from the booster guidance instrumentation, in this case it is likely the spacecraft computer will compute position and velocity during boost.

2.8.1.1.1 Computations

The IMU Mechanization function requires the following functions to be implemented:

1. Process Accelerometer Outputs

2. Navigation Computation:

   a. Compute Navigation Reference

   b. Extrapolate Gravity Velocity

Detailed mechanization equations will not be given here for these functions. These detailed equations were given in reference 16, the first quarterly report of the study. The same holds true for the remainder of the N&G functional description; appropriate comments will be made regarding the equations where necessary or they will be included where they hold particular significance to computer mechanization or complexity.

2.8.1.2 Earth Orbital Coast

During this mode of navigation the basic equation describing the motion of the vehicle is given by

$$\frac{d^2}{dt^2}\bar{r} + \frac{u}{r^3}\bar{r} = \bar{a}$$

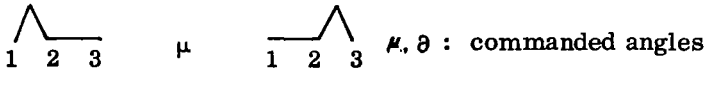
Where $\bar{r}$ is the vector position of the vehicle, u is the gravitational constant of the planet, and $\bar{a}$ is the vector acceleration which prevents the motion of the vehicle from being precisely a conic with the planet at the focus. Basically the motion of the vehicle is computed using a method such as Encke's. Periodically, due to accumulated errors, orbit determination is performed to update the state of the vehicle. The method of determining an orbit described here is that of using star-landmark tracking. The star trackers are used to provide a precise attitude reference while a landmark tracker is used to provide data for updating the state vector of the vehicle. Therefore, the functions are broken down into four sections: (1) Attitude Reference, (2) Landmark Tracker Operation, (3) Orbit Determination, and (4) Navigation Computation.

2.8.1.2.1 Computations - Attitude Reference

1. Star Selection Routine - This routine outputs the Line of Sight to two stars in inertial coordinates. The stars are selected which will be in the telescopes field of view and checks are made to see that a chosen star is not in a prescribed cone about the sun, moon, and earth.

2. Star Tracker Pointing

3. Tracker Acquisition and Tracking - The tracker will require a scan program to be superimposed on the commanded angles. The scan dither program may appear as:



Time of star presence must be accepted by the computer to interpolate the star angles.

4. Kalman Filter Star Data -

$$P_{n-1} = (I - B_{n-1} M_{n-1}) P^1_{n-1}$$

$$P^1_n = \Phi_{n-1} P_{n-1} + \Phi^T_{n-1} + K$$

$$B_n = P_n M^T_n (M_n P_n M^T_n + C_n)^{-1}$$

$$\hat{X}_n = B_n \Delta Y_n$$

$$\dot{\Phi} = F\Phi$$

Where

P_n : 11 × 11 covariance matrix of the estimation errors X_n evaluated at time t_n

I : 11 × 11 Identity Matrix

B_n : 11 × 2 Filter Matrix

M_n : 2 × 11 Output Matrix

K : 11 × 11 Constant Matrix (gyrodrifts, biases, etc.)

$\hat{X}_n$: 11 × 1 Optimum Estimate of Errors

ΔY_n : 2 × 1 Pointing Residuals

Φ: 11 × 11 Transition Matrix for Propagation of the Covariance Matrix

F : 11 × 11 System Matrix

C_n : 2 × 2 Covariance of White Observation Noise

5. Compute Body to Inertial Transformation from Gimbal Angles
6. Compute Locally Level to Inertial Matrix
7. Compute Locally Level to Body Matrix
8. Generate Attitude Control Signal

2.8.1.2.2 Landmark Tracker Operation

1. Landmark Tracker Pointing:
   a. Compute Initial Estimate of Landmark Position
   b. Correct Tracker Gimbal Angles
   c. Compute Angular Rates and Incremental Conditions
   d. Rapid Updating of the Gimbal Angles

2. **Landmark Tracker Data Processing**

a. Pattern Correlation and Tracking - This function requires the computer to store the present and previous digital scan of a landmark and shift the two scans relative to each other so as to obtain a best match; the shift between scans is then used to compute the observational residual.

b. Computation of the Observational Residuals

c. Computation of Expected Variance

2.8.1.2.3 Orbit Determination Computation

1. Prefilter Observational Residuals - Observational residuals may be obtained at a rate different from that used to perform the orbit determination computation. Therefore, a prefiltering such as a least squares method may employed between iterations.

2. Computation of the Output Matrix

3. Computation of the System Description Matrix

4. Initial Estimates of Covariance Matrices

5. Optimum Filter Computations

$$P(t_{k-1}) = P'(t_{k-1}) \left[I - M^T(t_{k-1})\, b^T(t_{k-1})\right]$$

$$P'(t_k) = \Phi_{k-1}\, P(t_{k-1})\, \Phi_{k-1}^T + R(t_k)$$

$$b(t_k) = P'(t_k)\, M^T(t_k) \left[M(t_k)\, P'(t_k)\, M^T(t_k) + C_k\right]^{-1}$$

similar definitions as given in the attitude reference section apply except the matrices involved are reduced to 9×9 and 2×9.

6. Propagation of Covariance Using the Transition Matrix

$$\dot{\Phi} = F\Phi$$

where a fourth order Runge-Kutta integration is used.

7. State Vector Correction

$$\overline{\Delta\xi} = b \cdot \Delta_{\overline{m}}$$

where:

$\overline{\Delta\xi}$: correction to be applied to the state vector

b : optimum filter computed in 5 above

2.8.1.2.4 Navigation Computations

1. Rectify Osculating Orbit
2. Update Osculating Orbit - The difference in the eccentric anamoly, ΔE, is computed in an iterative manner.
3. Compute Perturbations and Evaluate Derivatives
4. Update for Runge-Kutta Integration
5. Update State Vector Estimate
6. Correct the State Vector

2.8.1.3 Trans Mars Injection

During this mode powered flight is again encountered and the navigation functions of 2.8.1.1, Atmospheric Ascent, are applicable. In addition, the computer is required to compute the required velocity to achieve the desired trajectory and the velocity-to-be-gained to implement steering during the maneuver.

2.8.1.3.1 Process Accelerometer Outputs

2.8.1.3.2 Navigation Computation

2.8.1.3.3 Required Velocity Computation

The required velocity for trans mars injection may be defined as that velocity, at the present position, that will place the vehicle on a conic passing through a specified time.

2.8.1.3.4 Velocity-to-be-Gained Steering

The steering mechanization is a combination of two methods (a) alignment of the thrust vector, $\bar{a}_T$, with the velocity-to-be-gained, $\bar{V}_G$, vector and (b) alignment of the thrust vector to cause the time rate of change of the $\bar{V}_G$ vector, $\dot{\bar{V}}_G$, to be parallel to $\bar{V}_G$ and oppositely directed. A scaler mixing parameter, V, of these two methods is chosen to maximize fuel economy during the maneuver.

2.8.1.4 Trans Mars Coast

This mode of navigation consists of making sightings on planetary bodies to determine position and velocity and monitoring the velocity-to-be-gained for a trajectory correction.

2.8.1.4.1 Attitude Reference

2.8.1.4.2 Navigation Computation

Time did not permit defining a suitable mechanization for this function.

2.8.1.4.3 Velocity-to-be-Gained (Monitor)

2.8.1.5 Trajectory Correction

The basic functions required during this navigation mode are the powered flight functions of 2.8.1.1, the required velocity to be gained as described in section 2.8.1.4, and the steering function.

2.8.1.5.1 Process Accelerometer Outputs

2.8.1.5.2 Navigation Computation

2.8.1.5.3 Velocity-to-be-Gained

(Same as 2.8.1.4.3 Velocity-to-be-Gained (Monitor) except when $\Delta\bar{V}$ exceed some value and thrusting is initiated.)

2.8.1.5.4 Velocity-to-be-Gained Steering

2.8.1.6 Spin Up

During this mode the functions required are a determination of the angular velocity to be gained and steering commands to achieve the desired angular velocity. Time has not permitted defining a representative computer mechanization for these functions.

2.8.1.7 Trans Mars Spin Cruise

The functions required during this mode are similar to those required during the coast mode described in 2.8.1.4 (Attitude Reference, Navigation Computation, Velocity-to-be-Gained Monitor), in addition, the angular velocity to be gained will need to be monitored.

2.8.1.8 Despin

This mode is identical to 2.8.1.6 (Spin Up) with the exception that the desired angular velocity is 0.

2.8.1.9 Mars Approach Correction

The functions for this mode are the same as those required in 2.8.1.5 (Trajectory Correction) except that a different mechanization is used for the velocity to be gained computation (variable time of arrival guidance instead of fixed time of arrival guidance).

2.8.1.9.1 Process Accelerometer Outputs

2.8.1.9.2 Navigation Computation

2.8.1.9.3 Velocity to be Gained

$$\bar{i}_D = \text{unit} \left[(1 - \cos\theta)^2 \, \bar{i}_{r_p} + \sin\theta \left(\Delta - \cos\theta + \frac{p}{r_p} \, \bar{i}_r \times \bar{i}_n \right) \right]$$

2.8.1.9.4 Velocity to be Gained Steering

2.8.1.10 Aerobraking

Many of the functions required during this mode are expected to be similar to those of 2.8.1.15 (Earth Reentry). However, some simplification exists since it is not required to steer the vehicle to a desired landing site. The detailed equations for

the required functions were obtained from reference 13. A brief description of the required functions follows:

1. Out of Atmosphere Flight Predictor.

   Predicts flight conditions at top of atmosphere and range to top of atmosphere.

2. In-Atmosphere Flight Predictor.

   Flight prediction by integration of equations of motion with constraint and damping loops.

3. Coefficient Setup and Extrapolator.

   Linear extrapolation of predicted information with energy and time.

4. Non-Dimensional Constraint and Damping Computation.

5. In-Atmosphere Command Generation.

2.8.1.11 Mars Orbit Injection

Navigation and guidance functions during this mode are similar to those required in 2.8.1.3 (Trans Mars Injection) with the exception of the required velocity mechanization.

2.8.1.11.1 Process Accelerometer Inputs

2.8.1.11.2 Navigation Computation

(Same except gravitational calculations are on Mars)

2.8.1.11.3 Required Velocity Computation

2.8.1.11.4 Steering

2.8.1.12 Mars Orbital Coast

This mode will require the same functions as in 2.8.1.2 (Earth Orbital Coast). The method of orbit determination using star-unknown landmarks may be used to advantage here.

2.8.1.13 Trans Earth Injection

The functions of this mode are quite similar to those of 2.8.1.3 (Trans Mars Injection) with the exception of the required velocity mechanization.

2.8.1.13.1 Process Accelerometer Inputs

2.8.1.13.2 Navigation Computation

2.8.1.13.3 Required Velocity Computation

2.8.1.13.4 Steering

2.8.1.14 Earth Approach Correction

The functions in this mode are similar to those of 2.8.1.9 (Mars Approach Correction) with the exception of the required velocity computation (additional computations required due to an entry angle and landing site requirements).

2.8.1.14.1 Process Accelerometer Outputs

2.8.1.14.2 Navigation Computation

2.8.1.14.3 Velocity to be Gained

2.8.1.14.4 Velocity to be Gained Steering

2.8.1.15 Earth Re-Entry

The mechanization of this mode is quite complex and Reference 13 provides a good description and discussion of these equations. Briefly these functions are given below:

1. Out of Atmosphere Flight Predictor; Prediction of flight conditions at top of atmosphere and range to top of atmosphere

2. In Atmosphere Flight Predictor; Flight prediction by integration of equations of motion with constraint and damping loops

3. Coefficient Setup and Extrapolator; Linear extrapolation of predicted information with energy and time

4. Spherical Range Computation; Transformation of destination and target location to energy management coordinates

5. Nondimensional Ground Area Attainable and Target Overflight Ground Area Attainable prediction

6. Nondimensional Constraint and Damping Computations

7. In Atmosphere Command Generation

2.8.2 Telecommunications Requirements

This function is concerned with the data transmission function primarily. The information obtainable on this function was somewhat lacking and some interpolation had to be applied to define the requirements.

2.8.2.1 Transmission Instrumentation Pointing Commands

The functions involved here are computing desired Line of Sights, various coordinate transformations, computing pointing angles, and commands μ, ϑ. The equations are quite similar to those involved in paragraph 2.8.1.2, the attitude reference mechanization of the orbital coast phase.

2.8.2.2 Command Processing

Communication of commands from the ground will require the computer to accept and store after proper verification a number of commands. These commands may be in realtime or stored time. Stored time commands require a command storage program which is cycled through periodically to detect commands to be executed. Once a command is to be executed, the computer outputs an execute signal with the appropriate address for destination purposes.

2.8.2.3 Data Formatting

The allocation of this function is difficult to define at this time. There will undoubtedly be requirements for formatting and coding data prior to transmission. However, some coding of data will be taking place in functions described in other sections of this report (such as the data compression in paragraph 2.8.3). At certain times it may be required to process bulk data prior to transmission. Some of the algorithms described in the data compression section will be applicable here.

2.8.3 Scientific Experiment Computational Requirements

The description of computational requirements for the scientific experiments shall be given in three parts: (a) the scientific experiment instrumentation, (b) the computational algorithms to be applied to the scientific experiments, and (c) the resultant computer requirements.

2.8.3.1 Scientific Experiment Instrumentation

An investigation of References 1 and 14 provided what may be considered as representative instrumentation for achievement of the scientific experiments. The following is a list of the basic classes of experiments to be performed:

1. Investigation of Interplanetary Bodies: Comets and asteroids in close approach with the vehicle will be studied.

2. Analysis of the Interplanetary Medium: Various environmental properties need to be monitored such as: neutral gas, charged particles, neutrons, electromagnetic radiation, meteroid, magnetic fields, etc.

3. Observations of Solar Phenomena: Various observations will be made to determine properties of the photosphere, chromosphere, corona and magnetic moment and fields.

4. Analysis of the Aeromagnetosphere: Measurements will be made to determine a magnetic field map and magnetically trapped energetic charged particle belts.

5. Analysis of the Topography and Surface Composition of Mars: Measurements will be made to determine the amount of energy absorbed and reflected by the planet in different regions of the electro-magnetic spectrum, the composition of various areas of the plant, the existence and distribution of plant life and the topography of the planet.

6. Determination of the Periods and Gravitational Properties of Mars: The gravitational field and rotation of the planet will be determined and various properties of the satellites of the planet will be determined.

7. Analysis of the Martian Atmospheric Structure and Composition: Measurements must be made to determine the molecular and isotopic density distributions, the atmospheric density, the pressure, the temperature, etc.

The first four types of experiments may be considered as the cruise or interplanetary experiments while the last three are Mars orbital or Mars vicinity experiments. To achieve these desired scientific objectives the following instrumentation requiring or having the possibility of on-board data processing may be identified.

2.8.3.1.1 Interplanetary

1. 3-Axis Magnetometer

   The output of the magnetometer will be three channels representing flux intensity. Sampling requirements are expected to be at a rate of 12 channels/minute with a resultant information rate of 120 bits/minute. On-board data processing may be utilized here to compress the data prior to transmission by an encoding or curve fitting approach. (These will be covered in Section 2.8.3.2.)

2. 6-Axis Hi Energy Spectrometer, 3-Axis Moderate Energy Spectrometer, Proton Plasma Spectrometer, Electron Plasma Spectrometer, X-Ray and UV Photometer, Ion Chamber

   The output of these instruments will constitute approximately 191 channels with a total sampling rate of 224 channels/minute with a resultant information rate of 1120 bits/minute. On-board data processing can be utilized with this instrumentation to compress the data prior to transmission by utilizing encoding, curve fitting, and statistical methods.

3. Micrometeoroid Spectrometer

   Three channels of information will be received from this instrument; the sampling rate is expected to be quite low (3 channels/hour total) resulting in an information rate of 30 bits/hour. On-board data processing may be utilized to compute the mass from the momentum-velocity ratio and the data may be readily compressed using statistical methods.

4. Infrared Telescope and Spectrometer, Microwave Radiometer, Visible Wavelength Optics and Telescope

   This instrumentation is primarily required during the Mars orbital phase. However, it will probably be used during the interplanetary cruise to

measure such properties as solar microwave emission, solar infrared emission, spectra of selected astronomical objects, etc. The rates of taking measurements are expected to be considerably lower than when the instrumentation is used in the Mars orbit. It has been assumed that the average information rate is approximately 800 bits/sec for purposes of assigning requirements. Generally, the data processor may be utilized here to compress the data prior to transmission.

5. General Human Performance Measurement

This instrument is expected to be used periodically, for example, four hours/every four days. The data rates are expected to be 75 bits/minute input to the data processor and 100 bits/minute output to the instrument while the experiment is being conducted. The data processors will be utilized to activate various lights and output numerics on the display panel and to reduce the data obtained from the panel responses.

2.8.3.1.2 Mars Orbital

The discussion above for interplanetary instrumentation will also apply for the Mars orbital phase with the exception of the instrumentation identified in (4). The information rate is expected to be considerably higher in this phase for these instruments. It may be reasonable to assume that the full data transmission capability of the spacecraft will be utilized in this phase to make as many multiband spectral observations as possible. Predicted capabilities for the transmission rate for 1980 time period missions appears to be approximately (conservatively) 20,000 bits/second. The data processor may be utilized to compress the data obtained from the observations; a conservative estimate of data compression that will be obtained of 2 to 1 will be assumed. If it is assumed that the transmission consists of 80% assigned to the observations, this results in 16,000 bits/second. Application of data compression results in a capability of handling 32,000 bits/second from this instrumentation. Of course, a higher rate may be established if buffering of some form is used. However, this figure may be considered as a real time limit. The number of bits per observation per frame will depend on the region of the spectrum (visible, IR, etc.) that the observation is made in and properties of the optics (resolution, field of view, etc.). It will be assumed here that one frame will consist of approximately 10^6 bits (400 × 400 lines, 6 bit coding to a cell); this results in a maximum processing rate of one frame every 30 seconds (this may be considered a maximum average rate if buffering is available).

Data processing will be performed on the observation data to achieve compression prior to transmission; the method of curve fitting by computing coefficients of orthogonal polynomial series may be considered as representative of a compaction algorithm.

In addition to processing data from the on-board instrumentation, there are expected to be a number of remote sources of information (probes, excursion module) which will require data processing of experimentation information received via data links. These remote sources are:

1. Orbital Probe

This probe will be equipped with instrumentation identical to that listed under items 1, 2 and 3 of the interplanetary instrumentation. The

information rate for this environmental instrumentation is expected to be of the same magnitude.

2. Landing Probe

One or more of these probes may be expected to be launched prior to launching the excursion module. Typical instrumentation for such a probe may include:

Gas densitometer

Barograph

Thermistor and Shield

Radar Altimeter

Gas Chromatograph (or mass spectrometer)

Hi-Energy Partical Detector

Flux-Gate Magnetometer

X-Ray and UV Photometer (5 bands)

Sound Velocity Detector

Ionospheric Charge Density Probe

The data from these instruments will be transmitted to the mission module. This data may be expected to consist of 31 channels at a sampling rate of 151 channels/sec total. The resultant total information rate is 1356 bits/sec. Data processing will be performed on the data for compression prior to transmission to earth.

3. Mars Excursion Module

A listing of some of the possible instrumentation on board the excursion module is given below; this list is by no means complete since numerous experiments currently undefined may be desired on the surface of Mars.

Barograph

Soil Penetrability Probe

Thermistors (air and ground)

Flux-gate Magnetometer

Soil Density Meter

Anemometer

Soil Chemical Composition

Hi-Energy Particle Spectrometer

Gamma Ray Scintillator

Neutron Detector

Soil Electrical Conductivity

Sound Velocity and Seismic Microphone

Seismic Detector

Radiometer, Sweep Frequency

Insolation Spectrometer

TV Subsystem

Data from these instruments will be transmitted to the mission module where data processing will compact the data prior to transmission to earth. In addition to achieving data compaction by encoding and curve fitting methods, the data may be actually processed to achieve the end results of the experiment. An example of such an end result is computing the chemical constituents of a soil sample based on the data from the soil composition analyzer subsystem.

2.8.3.2 Computational Algorithms

This section contains a discussion of some of the potential algorithms that may be implemented for processing the data to achieve data compression and a description of other computation functions required. Data compression or data reduction may be grouped into three classes for this application: (1) compression by an encoding or curve fitting method whereby the data may be reconstructed after compression, (2) compression by computing some statistical properites of the data (such as mean and variance) and transmitting only the statistical properties. This method is useful when the original data need not be reconstructed. (3) compression by computing the desired or end objective of the experiment on-board instead of on the ground. These three general classes will be discussed in more detail below:

2.8.3.2.1 Encoding and Curve Fitting Data Compression

1. Debiasing - If the signal is expected to have a small dynamic range with a large magnitude, it may be advantageous to subtract a bias value (for example, the RMS value) from each sample and transmit the deviation from this bias value. An example of the applicability is in the monitoring of a power supply voltage; the value is expected to have small variations about some RMS value say 28 volts. It will require a fewer number of bits to represent the signal if 28 is subtracted from each sample.

Computation: $Y_n - K_y = T_n$

(Where Y_n is the actual value of the n-th sample. K_y is the bias constant for the signal Y. T_n is the transmitted value for the sample.)

2. Difference Coding - Instead of transmitting the value of the sample, the difference between successive samples may be transmitted. This transmission of first order differences is quite similar in applicability as debiasing; if the dynamic range between samples is small or the signal is relatively "smooth," a smaller number of bits will be required for representation. An important property of the above two methods is that they introduce no error due to compression.

   Computation: $Y_n - Y_{n-1} = T_n$

3. Zero Order Polynomial Predictor - The Zero Order Polynomial Predictor (ZOPP) method, commonly known as the floating aperture method, may be classified as curve fitting as may be all the predictor and interpolator methods. Basically the ZOPP transmits only the differences between samples if the differences exceed some preset value (this preset value is equal to half the aperture width, A). If no value of the difference is transmitted at time t, the value of the sampled signal is assumed to be the same as at t-1.

   Computation: if $|Y_n - Y_{n-1}| > A/2$ (Transmit Y_n)

4. Zero Order Polynomial Interpolator - The Zero Order Polynomial Interpolator (ZOPI) is very similar to the ZOPP with the difference being that instead of predicting succeeding values from past values, successive data points are examined and a horizontal line fitted to as many consecutive points as possible without creating errors greater than that permitted. Rather than attempting to describe the computations mathematically, a flow chart is given below showing the computations required for a given sample of the signal:

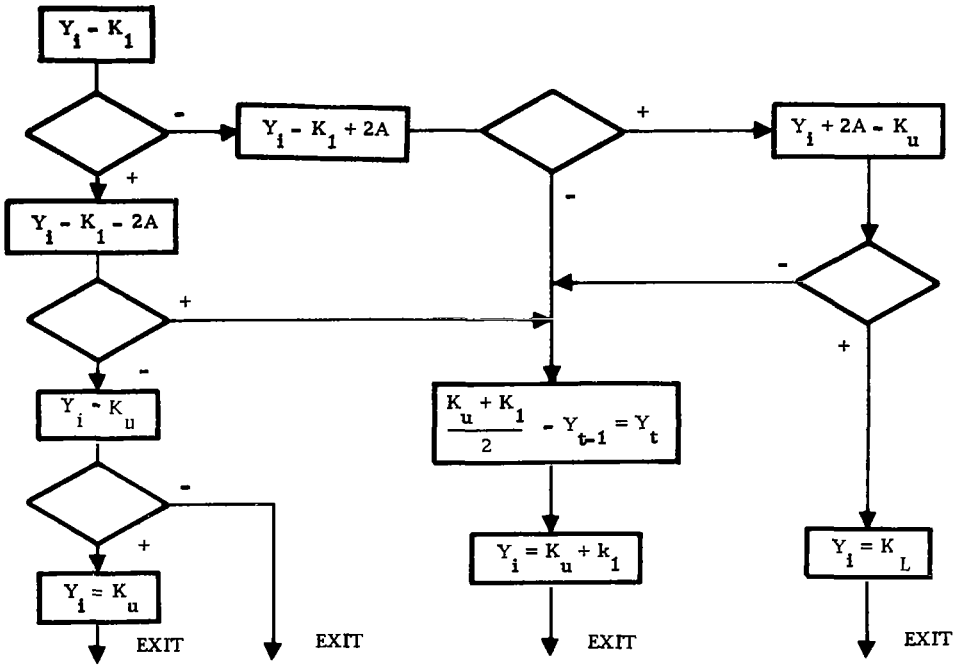
   Y_i is the current sample

   K_u is the Upper Bound

   K_l is the Lower Bound

   A is the aperture width

   Y_t is the transmitted difference (difference between horizontal line approximations)



5. First Order Polynomial Predictor - With this form of prediction, two data values are used to predict succeeding values. If the prediction is within the error tolerance, no new data is transmitted; new information is transmitted only when examination of successive data points reveals one which does not lie within the predicted region. For the first order polynomial process, straight lines are involved and the process has the form:

$$\hat{Y}_{t+n} = Y_{t+k} + (\hat{Y}_{t+k} - Y_{t+k-1})\ N$$

$$I_{t+n} = \begin{cases} 0 \text{ if } \left| \hat{Y}_{t+n} - Y_{t+N} \right| \leq A \\ \text{otherwise} \\ \left[(Y_{t+n} - Y_{t+n-1}), (Y_{t+n+1} - Y_{t+n}), N \right] \quad \text{(Case B)} \\ \text{or} \\ \left[(Y_{t+n} - Y_{t+n-1}), N \right] \quad \text{(Case A)} \end{cases}$$

Where: $\hat{Y}_t$ is the predicted value of y at time t

Y_t is the actual value of y at time t

I_t is the information at time t

N is a number of time intervals = $(t + n) - (t + k)$

k designates the time interval at which the last transmission of information occurred

A is the magnitude of the permissible maximum error

The difference between Case A and Case B is as follows: With Case A, when it is necessary to send more information to define a new line, the last predicted point becomes a point of the new line. With Case B, two new differences are transmitted, one giving the difference of the first point of the new line from the last predicted point on the old line, and the second difference giving the difference between the second point on the new line and the first point on the new line.

6. First Order Polynomial Interpolator - The First Order Polynomial Interpolator (FOPI) is an extension of the ZOPI technique. First order polynomials (straight lines) are fitted to as many succeeding data points as possible without exceeding a specified error. When the specified error is exceeded, information sufficient to define a line which fits the previous data points is transmitted; the examined data point becomes the first of a new set of points. Successive data points are added to it until the specified error is exceeded. To specify the approximating line, the following information is sent:

$$\hat{Y}_{t+n-1} - Y_{t+n}$$

$$Y_{t+n+1} - Y_{t+n}$$

and N where these terms have the same meaning as used in the discussion of the FOPP. The first piece of information sent defines the starting point of the line, the second its slope and the number of intervals, N, the length of the line.

7. Orthogonal Polynomial Series - Orthogonal function series may be employed for data compression by fitting one series to each successive time interval of sensor data, and then telemetering the resultant series coefficients rather than the sensor data itself. As an example, a four term trigonometric Fourier Series could be fitted to each successive interval of 20 sampled data values from a sensor, say a video scan, resulting in a possible compression ratio of 5 to 1. Orthogonal polynomial compactors are well suited to applications where the sensor is being sufficiently utilized so that its output data has an information density too high to be reasonably compacted by neighborhood commonality compression approaches.

This approach does not require a priori assumption or restriction on the structure of the data other than that it be continuous. Polynomials are chosen for orthogonal expansion basis functions $f_i(x)$ because of their properties of providing minimal mean square weighted error power series approximation, smoothing, and interpolation series for sampled data environments.

An n term compactor series of polynomials $f_i(x)$, over an interval T of sensor data y (t), $o \leq t \leq T$, is given by:

$$Y_a(t) = \sum_{i=o}^{n-1} a_i f_i\left(2 \frac{t}{T-1}\right)$$

where

$$a_i = k_i \int_o^T w\left(2 \frac{t}{T-1}\right) f_i\left(2 \frac{t}{T-1}\right) y(t)\, dt$$

Here $w\left(2 \frac{t}{T-1}\right)$ is a compaction error weighting function and the k_i are normalizing constants. In order to evaluate the coefficient integrals, a_i, on board, a numerical integration is made, replacing the equation by:

$$a_i \cong \sum_{j=0}^{N} d_{ij}\, Y\left(j \frac{T}{N}\right)$$

where the d_{ij} are the set of constant multipliers stored on board the spacecraft. This set of constants, d_{ij}, are computed on the ground by methods which will not be gone into here. After receipt of the transmitted coefficient, a_i, the compacted approximation $Y_a(t)$ to the original data can be recreated by the series given above.

2.8.3.2.2 Compression by Statistical Reduction

Often only the statistical nature of certain measurement data is of interest as in the determination of ion density or distribution of meteorite impacts. The data compression in this case results in the determination of certain statistical parameters such as the mean or variance, or certain points on a probability distribution curve.

1. Quantiles Representation

Further discussion beyond that given here on this topic may be found in Reference 15. A basic assumption involved in this method is that the time history of the measurement data is of no importance and that a probability density curve is what is primarily desired as a result of the experiment. Representation by quantiles is an effective means of compressing data under these assumptions. Basically, this method may be described as follows: a probability density function (x) is approximated by computing quantiles which represent a cumulative distribution function of the variable (x). A probability density function is shown in Figure 2-5. The variable, x, may typically be the number of particle counts per second and the curve for many experiments may have the normal distribution shown in Figure 2-5. If the particle counts are accumulated for a given period of time, say 1000 seconds, then the cumulative distribution function F(x), shown in Figure 2-6, may be approximated by the quantiles Q(n) where these quantiles represent the number of times (or frequency) particle counts up to some value, x_n, were received in the given time period, 1000 seconds. The value and number of the quantiles to be used is set and then the particle

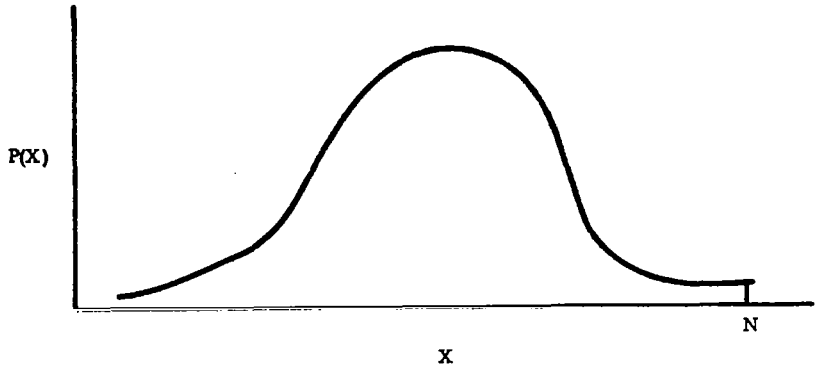


Figure 2- 5. Probability Density Function

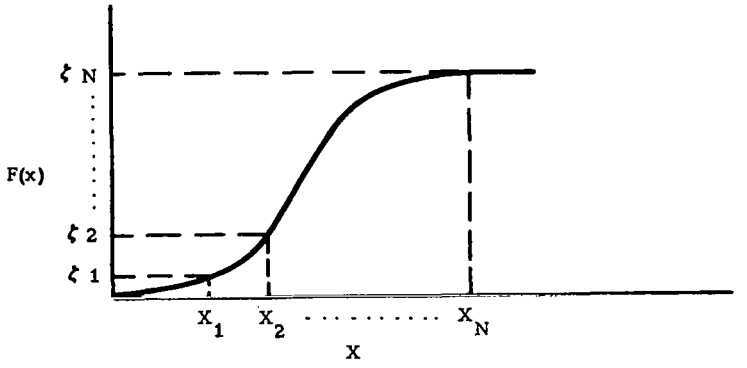


Figure 2-6. Cumulative Distribution Function

counts are accumulated for the time period. At the end of the time period the number of the particle counts of a given value received (starting from O and working towards the maximum N) are counted until the first quantile is reached, the value of X, resulting in this quantile is stored, and the process is continued until the desired number of quantiles are computed. It has been shown that four quantiles may be sufficient for a statistical representation.

Computation:

During Fixed Sampling Period:

Accept input data X
Update frequency of data X

At End of Sampling Period:

Add frequency of X sequentially starting from x = o.
If the cumulative frequency exceeds Q_1 store the value of X and use the next quantile Q_2.

Continue until all quantiles are computed.
Output the values of Q and associated values of X.

2. Representation by Moments

In addition to the statistical method described above, another statistical approach is to compute the moments directly. In general the p^{th} moment is given by:

$$M_p = \frac{1}{N} \sum_{i=1}^{N} (X_i - \bar{x})^p$$

The second moment (P = 2) is called the variance and the first moment the mean. The mean is given by:

$$M_1 = \bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

2.8.3.2.3 Compression by Complete Data Reduction

This method of data compression differs from the above two (2.8.3.2.1 and 2.8.3.2.2) in that it is not specified by a unique mathematical algorithm and its approach is to compute the desired end objective of an experiment on-board, based on the raw data from the sensors. Obviously the computational algorithms depend on the particular experiments and two applications of this approach will be given here:

1. Experiment: Composition Analysis

An experiment to analyze a soil sample may consist of bombarding the sample with alpha particles and measuring the returned scattered energy. This scattered energy spectrum may be analyzed to yield the elemental composition of the sample. A typical result from such an experiment is shown in Figure 2-7. This sample consisting of potassium, carbon, and oxygen may be completely analyzed on board rather than transmitting the scattered energy data to the ground. The procedure used may be of a least squares fit to reference curves stored on board, computing the slopes or breakpoints on the curves and determining the elements, or many other possible curve fitting methods. Some additional discussion beyond that here may be found in reference 10 on this experiment.

2. Experiment: Human Performance

The objective of this experiment is to measure samples of crew member performance on a set of tasks to detect any changes in his environment. The results of this experimentation will be compared with biomedical data to see what correlation exists and provide an evaluation of the effects of environment on human capability. A typical performance test unit may consist of a panel with colored warning lights and response buttons and arithmetic indicators. The experiment data may consist of responses to

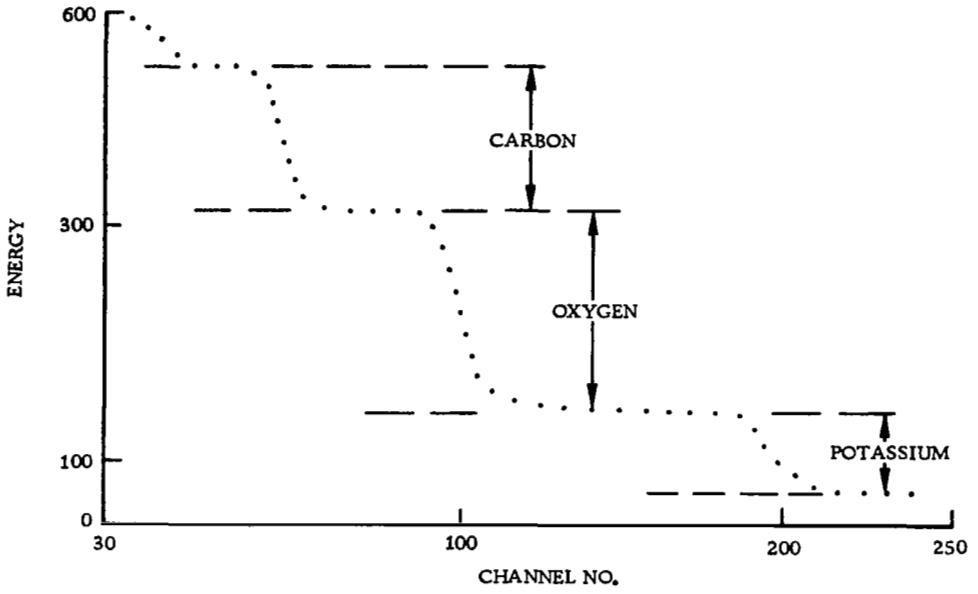


Figure 2-7. $K_2 CO_3$ Alpha Spectrum

warning lights and results from arithmetic manipulations. Some of this data may be reduced by the computer on board by computing the desired end results. A set of computations for reduction of such data may be described by:

a. Reduce data from warning light tests:

(1) Compute mean response time for each of 10 lights

(5 red, 5 green)

$$t^j_{r_m} = \frac{\sum_{i=1}^{n} t^j_{r_i}}{n}$$

where $t^j_{r_i}$ is the i^{th} measured response time of the j^{th} light. j : 1 to 10, n : 60

(2) Compute grand mean for red light

$$t_{r_{GMR}} = \frac{\sum_{J=1}^{5} t^j_{r_m}}{5}$$

(3) Compute grand mean for green lights

$$t_{r_{GMG}} = \frac{\sum_{J=6}^{10} t^{J}_{r_m}}{5}$$

(4) Search through red response times and determine minimum.

(5) Search through red response times and determine maximum.

(6) Search through green response times and determine minimum.

(7) Search through green response times and determine maximum.

(8) Compute mean response time for red/5 minute period

$$t^{R}_{r_m} = \frac{\sum_{k=1}^{50} t^{R}_{r_k}}{50}$$

where 10 measurements are assumed for each of the 5 red lights during a 5 minute period. If the test lasts 30 minutes, six of these response times need to be calculated.

(9) Compute mean response time for green/5 minute period

b. Reduce data from arithmetic tests:

(1) Compute percent correct/total period.

(2) Compute percent correct/each 5 minute period.

(3) Compute mean time to solve for total period.

$$t_{m_t} = \frac{\sum_{i=1}^{30} t_i}{30}$$

(4) Compute mean time to solve for each 5 minute period.

$$t_{m_5} = \frac{\sum_{i=1}^{10} t_i}{10}$$

(5) Compute minimum and maximum response time of total period.

(6) Compute minimum and maximum response time of each 5 minute period.

2.8.3.2.4 Sensor Sequencing and Scheduling

The scientific instrumentation consists of a variety of sensors as described previously (2.8.3.1). During any one phase such as the interplanetary cruise, it may be desirable to implement various portions of experiments based on other events or conditions, some of which may be predetermined and others adaptive. Thus, as conditions such as distances from planetary bodies change, scientific phenomena change such as solar activity. The instrumentation may be utilized in a manner to collect the optimum amount of information.

2.8.3.2.5 Sensor Pointing and Control

Some of the sensors will require pointing and control to obtain the desired information from them. The more obvious of these being the telescopic systems including the TV, Infrared, etc. The computer will be required to accept angular position signals from the sensors and based on inertial information compute appropriate pointing commands. Transformation matrices similar to those involved in navigation and control are required. A prime function of the computer will be to point the sensors at the planet in an optimum manner so as to prevent excessive overlap between scans, etc.

2.8.3.3 Computer Requirements for Experiments

The computer requirements will be broken down to two distinct phases for processing of information from the scientific instrumentation, interplanetary and Mars Orbital, since these are the two major differences in terms of experiment utilization.

These requirements are given in Tables 2-3 and 2-4. Storage, speed, and word length are given for each of the computational functions presented in section 2.8.3.2. The numerical values were obtained from a trial programming procedure using the data presented in section 2.8.3.2 and assumptions where necessary.

2.8.4 System Checkout

This function consists of various performance monitoring and self-test operations. In general, it is desired to achieve (a) malfunction detection, and (b) malfunction isolation. These items are achieved by two basic programs in the computer: (a) the status evaluation program, and (b) the status utilization program.

The status evaluation program is entered into first and three basic types of tests are performed on parameters monitored: (1) range evaluation (tolerance test), (2) rate evaluation, and (3) failure prediction; each parameter may be subject to one or more of these tests. Upon detection of a parameter out of tolerance in any test, the crew is advised of the result on the display panel. Then the out of tolerance condition results in an exit to a status utilization program. This program may read in additional parameters and perform further tests on these parameters in an attempt to isolate the malfunction. Upon completion of the utilization program the evaluation program is returned to complete the remainder of the monitoring program.

Some of the status monitoring data may also be transmitted to the ground. In this case, data compression (as described in the scientific instrumentation section) may be applied to reduce the transmission requirements.

A flow chart is shown in Figure 2-8 describing the status evaluation functions. The status evaluation routine in this drawing may be considered as the isolation routines.

Table 2-3. Interplanetary Experiment Computer Requirements

Function \ Requirements	Storage (Words) Instr.	Const.	Var.	Speed (Operations/Sec) Short	Long	Word Length (Bits)
1. Data Compression						
a. Debiasing	17	2	4	0.5	---	10
b. Difference Coding	19	0	100	7	---	10
c. Zero Order Polynomial Predictor	20	21	75	7.5	---	10
d. Zero Order Polynomial Interpolator	110	19	100	41	---	10
e. First Order Polynomial Predictor	73	19	100	27	0.5	10
f. First Order Polynomial Interpolator	175	19	100	64	0.5	10
g. Orthogonal Polynomial Coefficients	324	24	300	6361	522	12
h. Quantiles Computation	76	300	380	33	---	10
i. Moment Computation	57	5	150	30	1	10
j. Data Reduction	250	5	700	20	1	14
2. Sensor Sequencing and Scheduling	750	125	150	200	2	16
3. Sensor Pointing and Control	500	50	25	1000	400	16

Table 2-4. Mars Orbital Experiment Computer Requirements

Function \ Requirements	Storage (Words) Instr.	Const.	Var.	Speed (Operations/Sec) Short	Long	Word Length (Bits)
1. Data Compression						
a. Debiasing	17	5	10	340	---	10
b. Difference Coding	19	0	220	2900	---	10
c. Zero Order Polynomial Predictor	20	45	160	3000	---	10
d. Zero Order Polynomial Interpolator	110	41	210	16500	---	10
e. First Order Polynomial Predictor	73	41	210	10800	200	10
f. First Order Polynomial Interpolator	175	41	210	25600	200	10
g. Orthogonal Polynomial Coefficients	324	40	520	110,000	35,300	12
h. Quantiles Computation	76	630	800	12700	---	10
i. Moment Computation	57	11	315	12000	400	10
j. Data Reduction	750	5	900	1000	20	14
2. Sensor Sequencing and Scheduling	1000	175	200	500	5	16
3. Sensor Pointing and Control	1500	100	75	10000	4000	16

Note: Some of the storage requirements for these two phases are duplicated, and therefore they may not be summed up, e.g., total storage required for "Difference Coding" for the entire mission is 19 instructions.

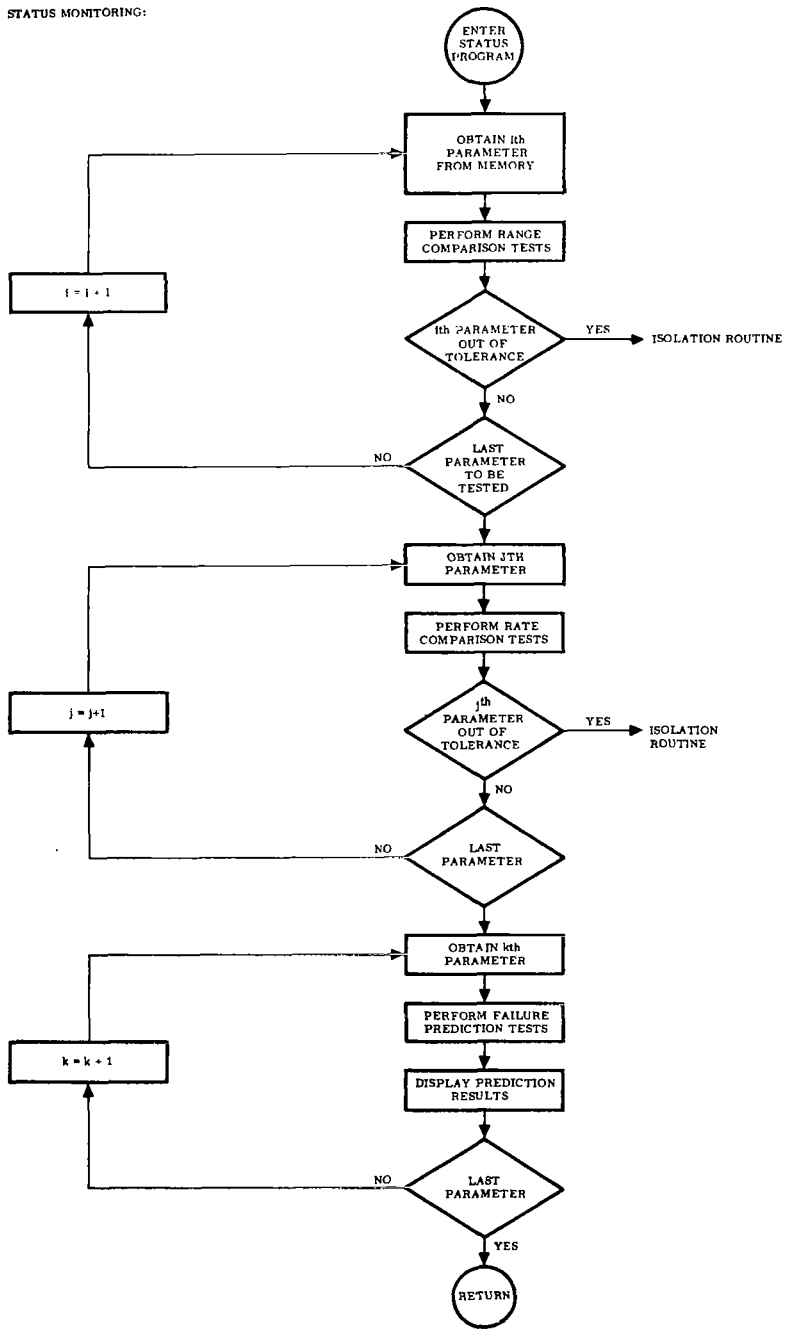


Figure 2-8. Flow Chart for Status Monitoring Routine

2.9 COMPUTER REQUIREMENTS

In the previous sections the computer functions have been presented and in this section the computer requirements shall be given. Before presenting the requirements, it may be worth while to discuss in general the methods used in obtaining the requirements. Basically the computer requirements are determined by investigating the equations and functional flow diagrams necessary for implementation of the system. Analysis of these identifies: (1) subfunctions for commonality, thereby resulting in possible subroutines, and (2) interdependence, i.e., the determination if partial results are required in later computations. A trial programming procedure is then used to obtain the requirements. Trial programming is coding without concern of address location or optimum coding. The requirements are then derived in terms of storage and speed.

The computer requirements for the multiprocessor study were first derived assuming a basic GP computer with a basic instruction repertoire was available; the requirements were then derived assuming certain features available such as indexing, multiple accumulators, etc. Appendix 1 contains a tabulation of the requirements obtained assuming a basic GP computer. The requirements are tabulated as storage (instructions, constants, and variables) in words, speed (short: Add, Subtract, etc. and long: Multiply, divide, etc. operations) in operations per second, and word length in bits where available. A detailed tabulation of the requirements for all the functions in every phase of the mission is given in this appendix. It should be remembered when reading this appendix that no special features such as indexing, multiple accumulators, indirecting, etc were assumed and no consideration was given to banking and computer word length i.e. double precision and half length requirements were not considered in determining the speed and storage requirements.

Tradeoffs were made on machine features by evaluating the effects on the requirements. These tradeoffs will be discussed in detail in Section IV, Paragraph 4.2.1. The requirements presented here in Table 2-5 are based on the following assumptions: an 18 bit word size computer with considerations of multiple precision operations and banking requirements, two accumulators are available, one index register is available, and a basic instruction repertoire is available. It should be noted that no provision is made for executive, self test, utility subroutines or Input/output requirements in the numbers given in Table 2-5. A rough estimate of the total requirements can be determined, if desired, by approximately a 20-25% increase in the figures given in Table 2-5. (This also holds true for Appendix 1.)

Storage requirements given below are the total number of words for instructions, constants, and variables; speed requirements are the total number of operations per second (equivalent short operations where it is assumed a long operation = 3 x short operation).

It should be noted for the purpose of presenting the requirements that the mission has been broken down into twenty phases. The requirements for the four basic functions described in paragraph 2.8 are given for each of these twenty mission phases.

Table 2-5. Computer Requirements by Mission Phase

Mission Phase	Storage (words)	Speed (Short ops/sec)
1. ATM. ASCENT		
a. Navigation and Guidance	664	1852
b. Telecommunication	600	1600
c. Scientific Experiments		
d. Status Monitor	960	1360
	2224	4812
2. EARTH ORBITAL		
a. Navigation and Guidance	6143	101202
b. Telecommunication	600	1600
c. Scientific Experiments		
d. Status Monitor	3560	8000
	10303	110802
3. TRANS MARS INJ.		
a. Navigation and Guidance	1994	51904
b. Telecommunication	600	1600
c. Scientific Experiments		
d. Status Monitor	1820	4050
	4414	57554
4. TRANS MARS COAST		
a. Navigation and Guidance	5026	65360
b. Telecommunication	2800	6500
c. Scientific Experiments	4753	10572
d. Status Monitor	3560	8000
	16139	90432
5. TRAJ. CORR.		
a. Navigation and Guidance	2479	133904
b. Telecommunication	2000	6500
c. Scientific Experiments	4753	10572
d. Status Monitor	1820	4050
	11052	155026
6. SPIN UP		
a. Navigation and Guidance	1060	63000
b. Telecommunication	2800	6500
c. Scientific Experiments	4753	10572
d. Status Monitor	1820	4050
	10433	84122

Table 2-5. (Cont)

Mission Phase	Storage (words)	Speed (Short ops/sec)
7. SPIN CRUISE		
a. Navigation and Guidance	5616	69160
b. Telecommunication	2800	6500
c. Scientific Experiments	4753	10572
d. Status Monitor	3560	8000
	16729	94232
8. DESPIN		
(Same as 6. SPIN UP)	10433	84122
9. MARS APPR. CORR.		
a. Navigation and Guidance	2799	133904
b. Telecommunication	2800	6500
c. Scientific Experiments	4753	10572
d. Status Monitor	1820	4050
	12172	155026
10. AEROBRAKING		
a. Navigation and Guidance	3400	42000
b. Telecommunication	2800	6500
c. Scientific Experiments	4753	10572
d. Status Monitor	1820	4050
	12773	63122
11. MARS ORBIT INJ.		
a. Navigation and Guidance	1329	39904
b. Telecommunication	2800	6500
c. Scientific Experiments	4753	10572
d. Status Monitor	1820	4050
	10702	61026
12. MARS ORBITAL		
a. Navigation and Guidance	6143	101202
b. Telecommunication	5730	15000
c. Scientific Experiments	6840	255000
d. Status Monitor	3560	8200
	22273	379402
13. TRANS EARTH INJ.		
a. Navigation and Guidance	1749	55704
b. Telecommunication	2800	6500
c. Scientific Experiments	4753	10572
d. Status Monitor	1820	4050
	11122	76826

Table 2-5. (Cont)

Mission Phase	Storage (words)	Speed (Short ops/sec)
14. TRANS EARTH COAST (Same as 4. TRANS MARS COAST)	16139	90432
15. TRAJ. CORR. (Same as 5. TRAJ. CORR.)	11052	155026
16. SPIN UP (Same as 6. SPIN UP)	10433	84122
17. SPIN CRUISE (Same as 7. SPIN CRUISE)	16729	94232
18. DESPIN (Same as 8. DESPIN)	10433	84122
19. EARTH APPR. CORR.		
a. Navigation and Guidance	3544	193904
b. Telecommunication	600	1600
c. Scientific Experiments		
d. Status Monitor	1820	4050
	5964	199554
20. EARTH RE-ENTRY		
a. Navigation and Guidance	6200	63360
b. Telecommunication	600	1600
c. Scientific Experiments		
d. Status Monitor	1820	4050
	8620	69010

Two graphs are given (Figure 2-9, Figure 2-10) showing the speed and storage requirements per phase. Each of these graphs consist of computations performed continuously (shown as solid lines) and computations that may be performed periodically (shown as dotted lines). It should be noted that speed requirements which are periodic are not additive - that is they will not require simultaneous computation of their periodic programs. These periodic computations are 1/2 hour every 3 days (Navigation and Guidance functions), 1/2 hour every 5 hours (scientific experiments), and on demand (checkout functions). It should be noted that only a portion of the above three functions can be broken down into such periodic portions. In addition, these periodic functions may be scheduled so as not to occur during short duration phases such as 5, 6, 8, 9, 10, 11, 13, 15, 16, and 18. This is why the speed and storage requirements on the graphs are lower in these phases than those given in Table 2-5. It should be noted that the total storage required for the other phases shown in Figure 2-9 corresponds exactly to that given in the table. (Note that the storage shown in dotted lines is required periodically only.)

However, this is not the case for the total speed required per phase as shown in Figure 2-10. In phases 4, 7, 14, and 17 when the periodic requirements exist, the speed requirements are actually greater than those given in the table. This is

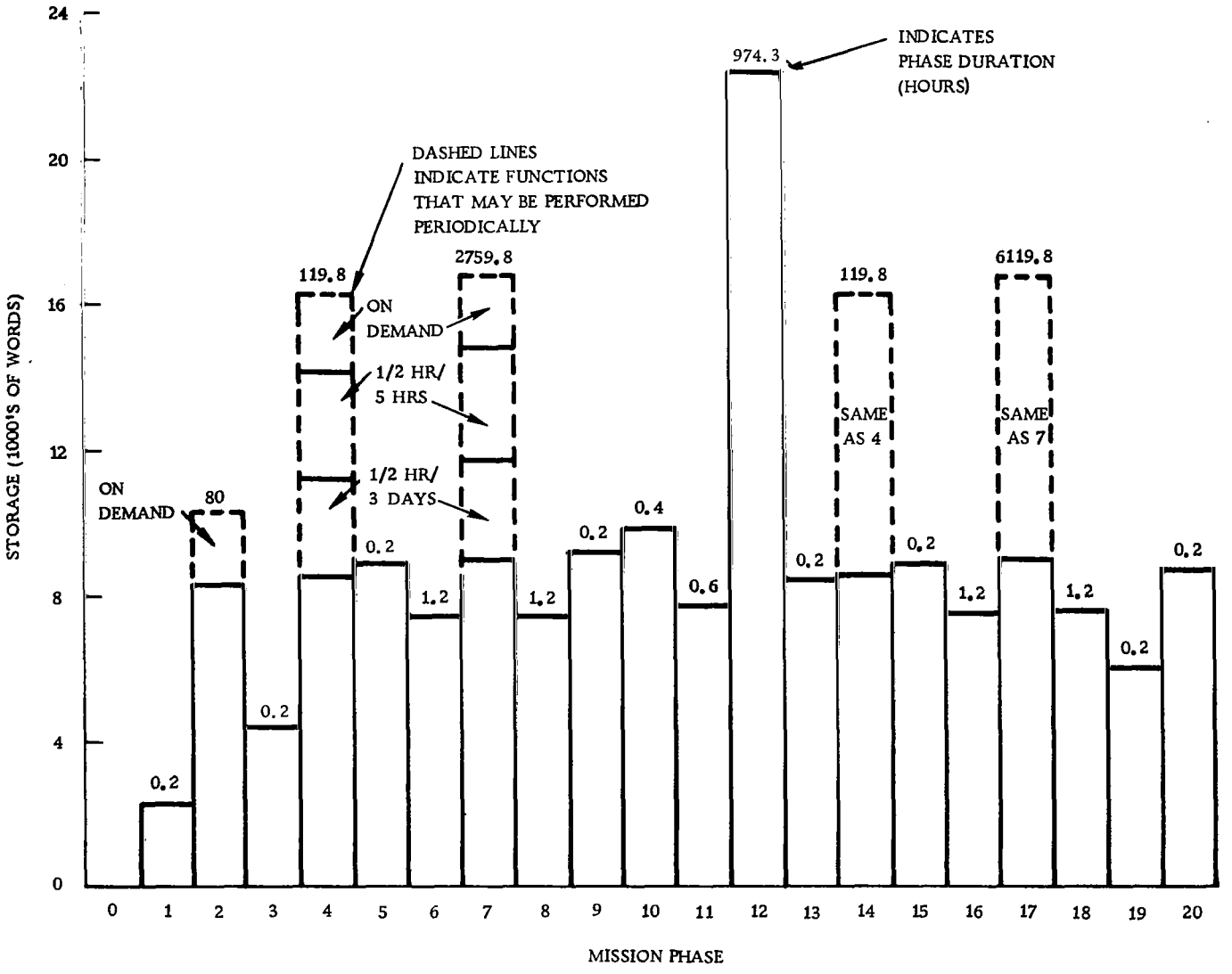


Figure 2-9. Computer Storage Requirements

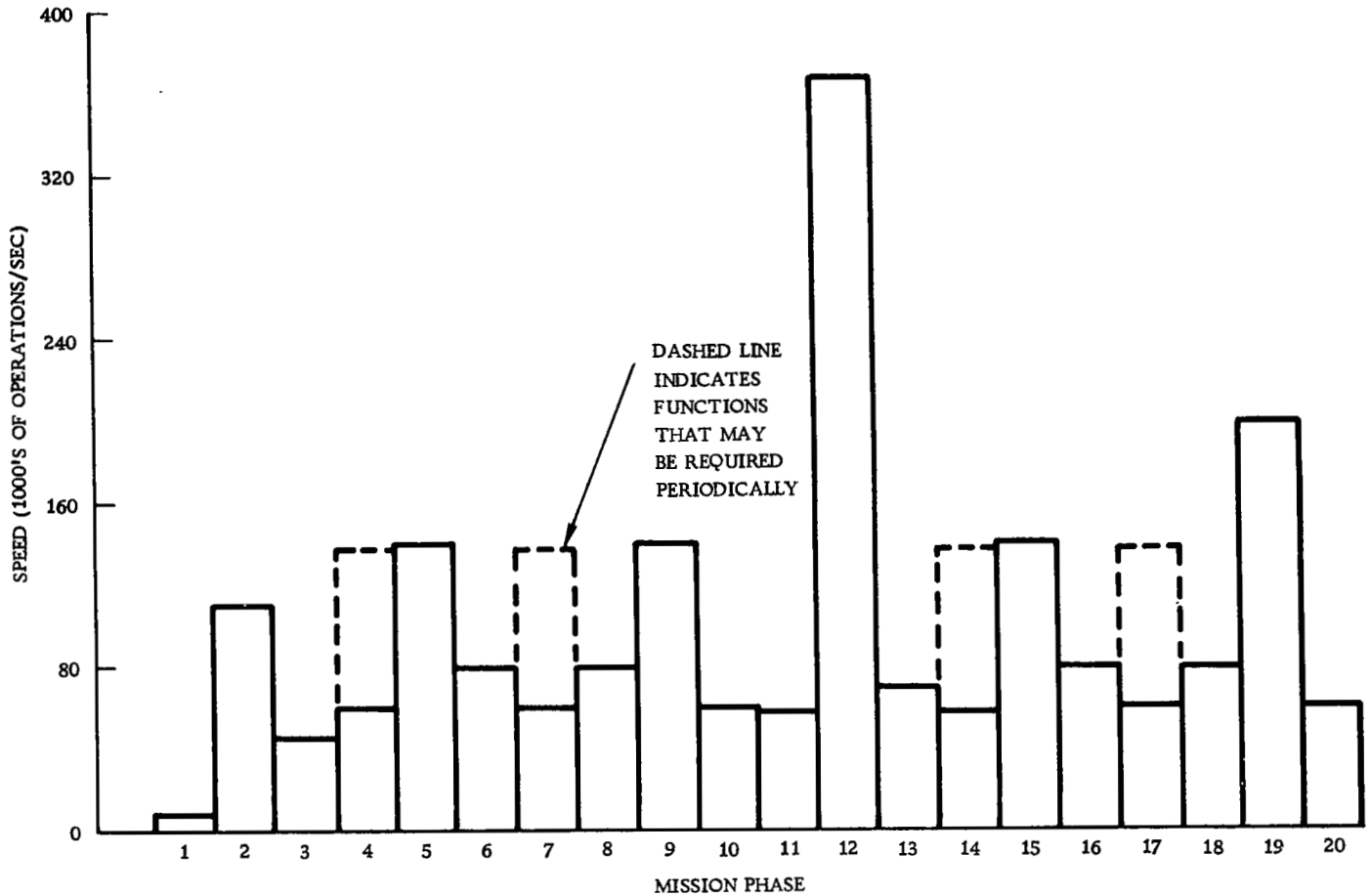


Figure 2-10. Computer Speed Requirements

due to the fact that the requirements for the functions were listed in the table as continuous requirements. If functions are performed periodically, in particular the scientific experiments data handling, there must be more data processed in a shorter period of time. This increase in processing speed may be varied by changing the on/off ratio of the function. The 1/2 hr/5 hrs for the experiment data processing gives a 1/10 ratio, which gives approximately an increase in speed by a factor of 10 over the speed requirement of this function if it were processed continuously.

These periodic requirements were pointed out since they may effect the design of a multiprocessing system. This may be seen from Figures 2-9 and 2-10 since it may now be possible to turn a portion of storage in a computer on and off periodically which may have the effect of reducing power consumption and also increasing reliability.

It is important to note here that the total storage requirement varies from phase to phase. The storage has not been cumulative added from phase to phase since it has been assumed that a bulk storage facility will be available for storing the programs required in each phase. If the computer were required to store the programs for the entire mission the total storage required per phase would increase considerably. The effects of this type of storage approach can be seen by examining the chart shown in Appendix A.

The above discussions have established the computer system speed and storage requirements. However, there are several other considerations that effect the computer requirements:

There are several phases in which the computations being carried out may be considered as "critical." The critical computations have two distinct considerations: failure detection and reconfiguration. Phases 10 and 20, Mars Aerobraking and Earth re-entry, contain critical computations, the navigation and guidance computations of these phases are critical in both failure detection and reconfiguration. The critical nature of these computations can be appreciated if one considers that as part of the navigation and guidance function, attitude commands are being computed and the vehicle may be near a temperature or acceleration limit, loss of attitude commands may possibly cause destruction of the vehicle. After investigating the navigation and guidance function for these entry phases, a total time of 5 seconds was defined as the maximum acceptable to detect a failure and also reconfigure the computation; reconfiguration is defined here as having the computational program with all the necessary values of the variables mechanized and being performed correctly in a computational facility after a detected failure.

It should be noted here that during phase 10, Mars aerobraking, the round trip communication time delay may be on the order of 30 minutes. This time delay makes it unfeasible to rely on earth based assistance during such a critical phase.

Another important point to note is that two reliability requirement constraints may be identified: Probability of Success and Availability. Probability of success is the appropriate parameter to consider as a reliability requirement during the critical phases of the mission while availability should be considered for the non-critical phases such as cruise and coast. Probability of success is not a very meaningful term in the non-critical phases; a computer failure has different effects on the mission depending on whether it occurred in critical or non-critical phases. Generally what one is interested in during the non-critical phases is what portion of that phase is the computer system operating correctly or availability of the computer system.

Phases 3, 5, 9, 11, 13, 15 and 19, which are basically trajectory corrections, may also be considered critical. However, the computations are critical only in terms of failure detection and once again the 5 second figure should be the maximum time allowed to detect a failure. Unlike the critical computations during the entry phases, loss of the computations during the above phases can simply result in shutting down the thrust motor with no catastrophic results, therefore, reconfiguration is not critical. The critical factor is to detect a failure and terminate the thrusting maneuver, after the failure is repaired a new trajectory calculation can be made and a new correction computed and applied. (This does not preclude the possibility of being able to reconfigure in 5 seconds also, since this requirement imposed by phases 10 and 20 may result in having this capability in these other critical phases also, particularly since phase 20 is the last phase of the mission.)

This Section has presented computer system requirements for the manned Mars mission. Based on these requirements various multiprocessing concepts shall be considered to evaluate their potential in meeting these requirements.

III. COMPONENT TECHNOLOGY

3.1 INTRODUCTION

In establishing the multiprocessor configurations and performing the tradeoffs to determine the most promising approach it is necessary to establish the technologies to be used in memories and circuits. The time period of interest for the Manned Mars Mission is 1980; however the technology time frame should be 1973-1975. This means that a usable technology must be in production by this time in order to allow for prototype construction and testing, and final design and construction of the computation system. The prototype system will of course need extensive reliability tests (greater than a year). Computer technology has had such a drastic change over the last six to eight years that it is hard to predict the most applicable technology at the end of the next six to eight years. Since it is desirable to choose a specific technology so that the study can proceed on a relatively concrete basis, a reasonable approach seems to be to examine technologies currently under development, use their functional characteristics for design, and extrapolate their physical characteristics to 1975 in order to perform tradeoffs. This is the approach being used.

3.2 CIRCUIT TECHNOLOGY

3.2.1 Introduction

In the semiconductor industry the trend is toward more complex array type structures in which hundreds of circuit functions are interconnected on a single chip. This approach provides lower power and higher reliability. The power is reduced primarily because interconnection capacitance is reduced. Reliability is primarily increased because there is an order of magnitude reduction in components and connections. The array type structures are currently being produced in limited quantities; however, it is expected that by 1975 the array type approach will be a common proven technology.

Presently there are two device approaches available which are compatible with array fabrication techniques: MOS, and bipolar IC's.

The circuit technology under primary consideration for this study is MOS (Metal-Oxide-Semiconductor). In particular, by the 1973-1975 time frame isolated MOS devices on an insulating substrate (heteroepitaxial technology) should have proven high reliability, low power and cost, and good radiation resistance. An example of this technology is the Silicon-on-Sapphire (SOS) circuitry presently being developed at Autonetics. As a result the sections of this report discussing circuit implementations will refer to the use of MOS-SOS chips. The densities and packaging used will be those assumed to be reliably feasible in 1975.

3.2.2 Advantages of MOS-SOS For Space

The SOS technology consists of a sapphire substrate with interconnected thin film silicon devices on one surface. The devices are fabricated in electrically-isolated islands of the silicon film. Array intraconnections are made by vacuum deposited aluminum or other metal films. The result is a fully integrated thin film circuit array on an insulating substrate employing single crystal silicon material and silicon integrated circuit batch fabrication processes. This technique thus possesses the electrical isolation and design flexibility of thin film-hybrid circuits which is the

essential attribute necessary to succeed in making large, high density functional devices. It also provides all the advantages associated with MOS technology. Figure 3-1 shows a cross-sectional view of a MOS-SOS field effect transistor.

For space applications MOS should prove to be the technology of the future for a number of reasons. Its power drain is significantly lower than that associated with bipolar circuits. The higher yields and much smaller size of MOS enable MOS chip complexities to be increased over that of bipolar. It is anticipated that the increase in density offers the potential of a greater reliability since fewer packages and intraconnections are necessary in a given system. (Data tends to point to the fact that the number of packages in a system and not the complexity of the packages have the dominant effect on system reliability.) Simpler processing and this same high density also gives MOS circuitry the potential for a large cost savings. The only present disadvantages associated with MOS circuitry compared to bipolar circuitry are its slower speed and lower radiation resistance. The speed difference should be somewhat overcome by the development of practical MOS-SOS complementary circuits. These circuits presently yield 10ns gate switching speeds in Autonetics' labs. However, in space applications the requirements for computation speed as defined in the previous chapter are not severe and in any case can be more reliably met by multiprocessor organizations such as those under study on this contract. Very high radiation resistance is at least not a requirement for manned space missions; however, it may be for other missions. Studies are presently being carried out to investigate such developments as new insulation layers in MOS devices. Replacement of SiO_2 by other insulators such as MgO offers the possibility, by 1975, of an increase in the radiation resistance of MOS devices by an order of magnitude or more. MgO MOS devices have been made in Autonetics' labs.

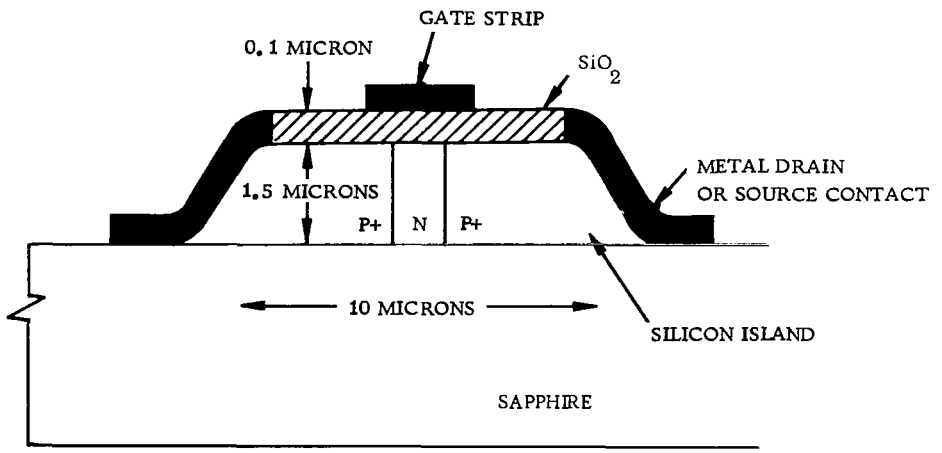


Figure 3-1. Cross-Section of P-Channel Junction Type MOS/SOS Transistor

As mentioned earlier, the fabrication method associated with MOS-SOS devices give it a number of attributes beyond those of bulk MOS devices. A few of these are listed:

1. The electrical isolation of all elements and interconnections on an insulator enables fabrication of devices with better electrical characteristics, with multilayer interconnections, and with the possibility for improved reliability. A reliability improvement is possible since less silicon surface area is available for failures, such as shorts through the SiO_2 layer.

2. Negligible parasitic lead capacitance and no substrate capacitance means that transient power can be reduced by a factor of two or more.

3. The transparency along with good thermal conducting properties of the sapphire substrate provide excellent opportunity for packaging innovations.

4. High tolerance to radiation (compared to bulk MOS) is expected due to the inherent device isolation and small device junction size.

There are a number of other points of interest in a discussion of the advantages of MOS-SOS for space applications; however, the above discussion should adequately substantiate the choice of a heteroepitaxial technology (MOS-SOS). In short, this technology should be well established by 1975 and should also offer reliability, power, and cost advantages over other technologies.

3.2.3 MOS/SOS Characteristics for 1973-1975.

As mentioned earlier, an accurate estimation of the physical properties of MOS/SOS in the 1973-1975 time frame is difficult to make. Such an estimation would most likely be conservative since actual processing breakthroughs cannot be anticipated. In any case to make estimations, Autonetics' past experience in its MOS and MOS/SOS pilot line construction of complex chips (800 FET's per 100 x 150 mil chip) and from Research and Development Lab work on MOS/SOS technology was used. The present MOS devices (FET's) are about .3 by .8 mils (larger devices are also used) and can be spaced approximately 1.2 mils center to center (present MOS/SOS devices cannot quite do this well); however actual complex circuits need so many lines and crossovers that the average densities are not nearly this high (800 FET's per 100 x 150 mils). Future multi-level crossover development and interconnection schemes along with smaller devices should enable circuits to be built with an average center to center spacing of 2 mils; however, this will require a considerable amount of MOS/SOS process improvement. In order to be slightly conservative an average of 2 mils center to center spacing will be used for 1975 MOS circuits. This gives approximately 5,500 FET's per 150 mils square. The actual chip size to produce good yields on complex circuits is not clear; however, 150 mils seems reasonable and will be chosen as a conservative estimate. The actual number of FET's probably lies in a range around 5,500, such as 4,000 to 6,500. Clearly, a few processing breakthroughs enabling yields to be increased would make larger chips available. For example, a 200 mil square chip (or larger) may well be usable in the 1975 time frame. Such a chip could contain 10,000 FET's. The 5,500 density number will be used to estimate the number of MOS/SOS chips necessary for implementation of any given candidate. In these estimates it is not critical as to whether the chip is 150 mils square or 200 mils square. The only fact of importance is that a small chip with 5,500 devices can be produced with reasonable yields.

Complementary MOS/SOS circuits presently show 5 to 10 ns unloaded gate switching speeds in the lab; as a result these devices in production in the 1975 time frame should be capable of operating at a five megacycle or better clock rate. The processors under consideration for this mission only require about a two megacycle clock; consequently, the MOS/SOS chips will easily be able to handle the speed requirements.

MOS chips are presently packaged in forty lead packs. This situation should be improved with future packaging methods. As an aid, for example, lines could be fanned out on the sapphire substrate so that larger packages could be used. Again, it is difficult to predict the development of the packaging technology, but it should certainly be reasonable to expect 100 to 150 pin packs for MOS/SOS chips in the 1975 time frame.

3.3 MEMORY TECHNOLOGY

Three main types of memory have been considered for the multiprocessing candidate systems. These are DRO core memory, NDRO magnetic memory, and NDRO semiconductor (MOS/SOS) memory.

All three approaches have been considered for the multiple computer and modular multiprocessor organizations. The NDRO magnetic memory has been chosen over the DRO core structure for the 1975 time frame. The reasons behind this choice along with a discussion of these two approaches are given in Section VI. In short, the NDRO magnetic structure offers increased reliability due to less sensitivity to transients, high quality control from a batch processed structure, and the ability to use many LSI circuits in its structure. (DRO structures in the 1975 time frame appear to require too much current to be amenable to the use of LSI circuitry.) This structure also dissipates less power than a DRO structure. A choice was not made between the NDRO magnetic and semiconductor structures. Both of the structures should be able to meet the reliability requirements in the 1975 time frame; however, the magnetic structure requires fewer processing developments in order to meet these requirements. (Its risk is lower.) However, either structure offers low risk. The semiconductor memory, on the other hand, should dissipate less power than the magnetic structure. Both of these structures are discussed in some depth in Section VI.

The distributed logic structure presented in Section IV, 4.3 uses MOS/SOS chips for memory and processing. This technology has been discussed in Section III, 3.2.

A bulk memory is also included in all three organizations. This memory will contain about 10^8 bits and will be used to store all programs for all phases and for buffering of telecommunications, TV, and other high rate input output. This memory is discussed in Appendix 2.

IV. MULTIPROCESSOR CANDIDATE ORGANIZATIONS

4.1 INTRODUCTION

Three candidate multiprocessor computer organizations were designed to implement the requirements set forth in section 2 using the technology base established in section 3. These candidates are: (1) Multi-Computer, (2) Modular Multiprocessor, and (3) Distributed Processor. The first two candidates have sufficient general commonalities to warrant their being presented in paragraph 4.2 with the commonalities and peculiarities of each discussed therein. The Distributed Processor will be discussed in paragraph 4.3.

4.2 MULTI-COMPUTER AND MODULAR MULTIPROCESSOR

4.2.1 General Organizational Considerations and Features

4.2.1.1 General Features

Many of the considerations were the same in the preliminary design of the Multiple Computer and the Modular Multiprocessor (hereafter referred to simply as Multiprocessor); therefore, a discussion of features common to both organizations and how the requirements effect the organization is given prior to a discussion of any of the candidate organizations. The features discussed in this section are for the most part arithmetic and control or processor section oriented. There are also a number of similar features in the memory and Input/output sections of these two candidates; however, it was felt that the considerations leading to these sections were sufficiently unique to warrant separate discussions for all the candidates.

An 18 bit instruction word shown in Figure 4-1 has been chosen. This format was chosen after tradeoffs on various other formats described below; these tradeoffs are reported in paragraph 4.2.1.2 and were performed by analyzing a number of representative programs. As an alternative, the format of the chosen 16 bit instruction word is shown in Figure 4-2; the 16 bit approach could be taken if further study were expended to determine (a) if a 32 bit double precision data word is sufficient to meet data accuracy requirements and (b) if not having indirect addressing and having a total of 5 instead of 9 Index/Bank registers did not reduce the programing efficiency greatly. The first 6 bits of these instructions are used for the op code. Operation code extensions for instructions that do not require full addresses, e.g., I/O instructions, accumulator to accumulator operations, and register transfers, give the facility for many more than 64 instructions.

The instructions will use a banking scheme so that it will only be necessary to have an address decrement in the instruction word. Programing studies carried out on various programs at Autonetics and also on this study have shown that a 7 bit address decrement provides little inefficiency penalties in terms of speed decreases or storage increases, and that the index/banking scheme using full length registers greatly reduces the banking problems.

The 18 bit instruction contains one bit, I, for indirect addressing. Both instructions contain B and T bits for the Index/Banking tags. One bit is used for the B tag and specifies one of two index/bank registers, the 18 bit instruction contains 3 bits for the T tag to specify either one or none of 7 registers while the 16 bit instruction uses

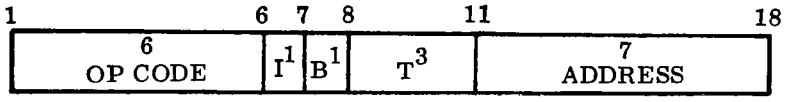


Figure 4-1. 18-bit Instruction Word

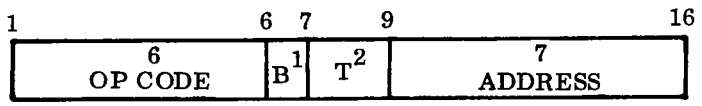


Figure 4-2. 16-bit Instruction Word

2 bits for the T tag to specify either one or none of 3 registers. This Index/banking scheme offers a certain amount of double indexing or banking. The index/banking schemes will be further discussed below. It should be mentioned that the indirect bit was considered to be of marginal value; the other 17 bits were firmly chosen based on an evaluation of the features. An indirect bit was considered next in terms of utility to complete the 18 bit word.

Various possibilities considered for the 18 and 16 bit instruction word formats are shown in Figure 4-3. Of course many other variations are possible; however these were considered the most promising possibilities. It should be noted that for all instructions for which it is applicable the 6 bits for the op code are broken down into 5 bits for the op code and 1 bit for an accumulator tag. As part of the programing analysis the use of the second accumulator was evaluated to determine if the use of the accumulators was symmetrical enough to include an accumulator tag. This simplifies the logic somewhat.

Two basic banking-indexing schemes were considered; the first scheme was selected using 1 and 3 bits. One scheme uses: one bit to specify one of two full-length registers and then two or three other bits to specify either one or none of three or seven other registers. The second scheme uses: four bits to specify any combination of five registers taken zero, one, or two at a time. The important point to notice about both of these structures is that there is no real distinction between bank registers and index registers since they are both full length (18 bits). Any of the registers can be added to the address decrement to generate a full length address or certain combinations of two registers can be added together (depending on which of the two schemes is chosen) and added to the address decrement to generate a full length address. These schemes have a number of advantages in terms of flexibility of use. For example one of the index/bank registers can be used to address a certain bank and can also be counted down and compared to a value. The schemes also offer the ability to double index. This means that a program can be set up (including index decrementing and comparing) without regard to the location of the program. Double indexing also makes the programer's task much easier where a number of index registers are needed, for example matrix manipulations. It is important to notice that full length bank registers means that there are no fixed bank boundaries. The only constraint is that a bank contains a maximum of 128 words. This means that it is not necessary for the programmer (or the assembler) to pack a number of programs or blocks of data into fixed size and location banks. (This would be necessary if bank registers containing only the upper 9 bits of an address were used.) Full length bank/index registers also means that banks of information are relocatable to any location in memory.

Two upper accumulators will be used. This decision was based on programing studies carried out at Autonetics on previous programs and also on this study. Some of these tradeoffs on two accumulators are given in paragraph 4.2.1.2. The two accumulators will be used referenced by an accumulator bit in the operation code in a number of instructions that access the memory. For accumulator to accumulator instructions a separate register instruction format utilizing operation code extension is used. This format enables the processor to carry out logical and arithmetic operations using the accumulators and the index/bank registers. As a result, the index/bank registers now used for hot storage plus addressing are increased from the 16 bits required for addressing to 18 bits. This provides for efficient use of the 9 index/bank registers since they must be connected to the adder for generating an address in any case. However it should be noted that the above use of the index/bank registers

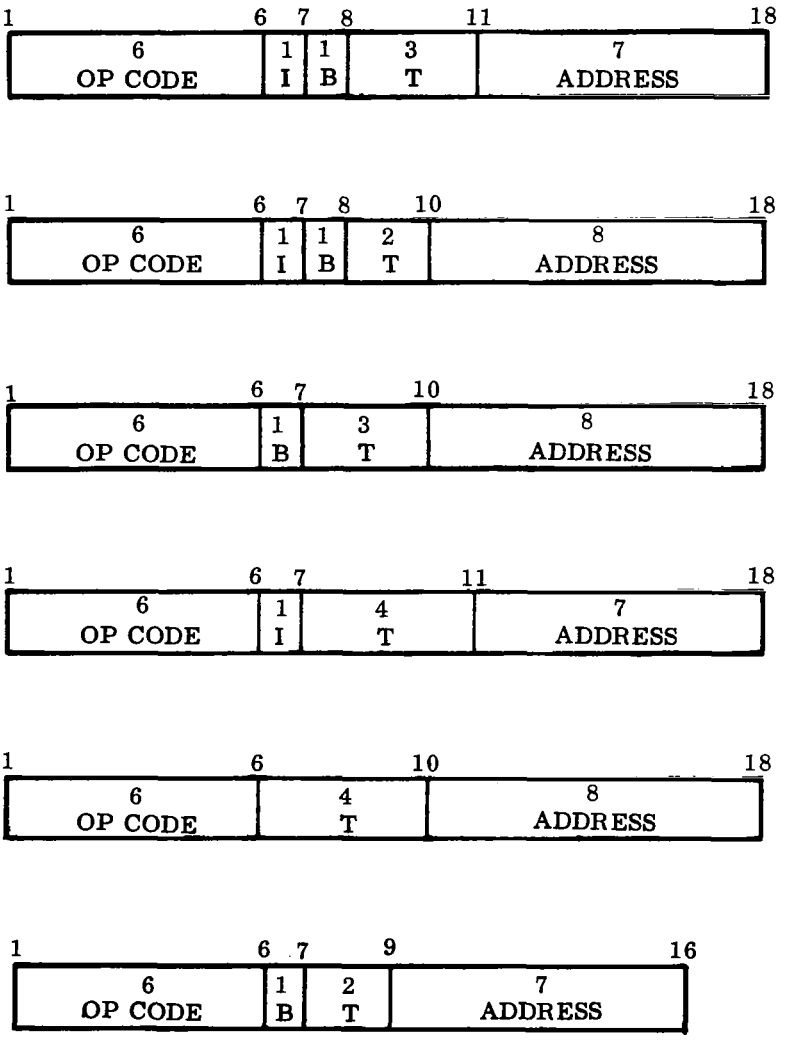


Figure 4-3. Instruction Word Formats

will not make them equivalent to accumulators (or like true general registers) since they cannot be used to carry out operations directly with operands from memory. This latter feature was evaluated to be of little use if two upper accumulators are available. Further discussion of the value of the above features is discussed in paragraph 4.2.1.2.

It is possible to include indirect addressing as either a bit in the instruction word or as one of the index registers' tag values. The former implementation enables indirect addressing and indexing whereas the latter does not. The former approach was selected as discussed in 4.2.1.2.

The indirect word format (the word picked up by the initial address) uses bits one and two to specify the indirect operation and bits 3 to 18 to specify an address. Bits one and two are interpreted in the following manner:

1:2 = 00 - end - direct address
01 - index with T_1 register
10 - indirect address
11 - index with T_1 and indirect

It should be noted that the above format not only allows multiple level indirecting and indexing but also allows simply indexing after indirecting. This latter feature is very useful when a reconfiguration has caused a block of information to be moved to a new location.

For a time consideration was also given to using one bit to specify fixed bank 0 (upper address bits equal to 0) and one bank register instead of two bank registers specified by this bit. The thought was that this would save hardware and also provide for easy subroutine linkage and bank/index constant storage; however, the multiprocessor organization imposes certain restrictions on accessing memories which will be discussed later and this made using a fixed bank impractical.

It is possible to include a masked mode of operation in the processor. The masked mode would be entered by an instruction that would set the masked flip-flop to one. All appropriate instructions executed while the flip-flop is "one" would use the second upper accumulator to mask the operands that come in from memory. For example a masked mode add could be executed as follows:

$$U_1 + U_2 \cdot (M) \rightarrow U_1 \qquad M = m+B_1$$

where

U_1, U_2 = accumulators
B_1 = index/bank register B_1
m = address decrement
(M) = contents of location M

The masked mode is useful if there are a reasonable number of operations with data that is packed with more than one character per word. This situation exists for some scientific experiments where 8 or 9 bit data seems to be sufficient. In these

situations the masked mode could be used to obtain half word operations. The present requirements study has not uncovered sufficient situations in which a masked mode would be useful to warrant its inclusion in the final processor specification. However future requirements studies should investigate in some depth the usefulness of such a mode especially in relation to one-half word data operations. If further information on the masked mode is desired reference 18 listed the instructions that could be executed in this mode.

Some floating point or double precision operations are certainly necessary for the computation system. The needs for these operations have been investigated to some extent, but a much more thorough investigation is necessary to explicitly determine the following:

1. Is floating point or double precision the best increased precision mode?

2. Should hardware (a double precision or floating point mode) or software be used?

3. If hardware is used, should a few double precision or floating point instructions be included in the single precision mode (or conversely)?

The preliminary precision investigations carried out in this study indicated that a floating point hardware mode using two word data would be well used in many navigation and scientific experiment operations due to both the need for a considerable amount of scaling and precision beyond 18 bits. As a result floating point instructions are listed and their operation briefly discussed in section 6.1.1. However it should again be mentioned that the above conclusions need further substantiation since it is not clear whether 30 bits of precision are sufficient. If not, a double precision mode with 36 bits of precision would be used. The two word floating point number will use a 30 bit mantissa and a 6 bit exponent although further study of this point is also necessary since a 29 bit mantissa and a 7 bit exponent may be a better solution. In any case the chosen floating point number provides for thirty bits of precision on data of magnitudes between 2^{-32} and 2^{32}. An additional reason for inclusion of floating point hardware was that by making a few simple additions to the adder (exponent operations must be inhibited from affecting the mantissas and conversely), no additional registers need to be added to the processor. A sizeable amount of control is necessary to carry out the operations, but, as mentioned earlier, a large amount of gating can be efficiently and reliably handled with MOS arrays.

A repeat mode has been included in the system. It is particularly useful for carrying out memory check sum tests and for moving blocks of data in main memory. This latter function is necessary during reconfigurations, at mission phase changes, and on a lesser scale during operational periods. In this mode the operand cycle of an instruction is executed on a succession of operands. The mode is entered by giving a repeat command (REP) that sets a flip-flop and loads a specified index register, T_7, with the number of operands to be processed. The next instruction is then executed in the repeat mode. Execution of operand cycles continues with T_7 counted down each cycle until it reaches zero. At this point the instruction is terminated and the next instruction accessed. Clearly this repeat mode will save a significant amount of time any time a list of data must be processed by one instruction. A good example of this time savings is in the execution of a check sum test on a program. (This test may be carried out periodically prior to the execution of some programs.) Without a repeat mode the check sum basically involves a two instruction loop of add and decrement and

test an index register. This is a 6 μs loop that is executed n times for an n instruction program. The same loop with a repeat mode requires only one instruction cycle of 2 μs followed by n operand cycles of 2 μs. This means a check sum loop execution time of 2 μs plus 2n μs compared to 6n μs. The instructions that can be executed in the repeat mode are given in Section VI, 6.1.1 along with a more detailed discussion of its implementation.

4.2.1.2 Evaluation of Features

This section discusses the results of the evaluation and the basis for the selection of the features presented in the preceding section and certain other features required in the preliminary design of the candidate organizations.

4.2.1.2.1 Multi-Accumulators and Indexing

The requirements presented in Appendix 1 for phase 12 of the mission (Mars orbital) were reevaluated with the ground rule that two accumulators and one index register were available. The effects on the requirements from these features is expected to carry over into other phases also. It should also be noted that these requirements did not take into account computer word length. The only effects examined were that of multi-accumulators and indexing. Table 4.1 presents the results from this evaluation.

Table 4-1. Speed and Storage Requirements for Phase 12, Mars Orbital

Requirements / Function	Storage		Speed	
	a	b	a	b
Navigation and Guidance	5625	4348	48949	46732
Telecommunication	6700	4700	13000	13000
Science Experiments	8940	8561	325715	252496
Status Monitor	5560	3560	8000	8000
Totals	26825	21169	395664	320228

a: no indexing, one accumulator
b: one index register, two accumulators (Requirements do not consider word length)

For Navigation and Guidance indexing accounted for most of the decrease in storage requirements (approximately 23% decrease) while the speed requirements had a net reduction (increase due to indexing and a decrease due to two accumulators) of approximately 6%. The Scientific Experiments function had a slight decrease in storage (approximately 4%) and a significant decrease in speed (approx. 23%). The effect of these features is presented in more detail in Reference 17. The results of this evaluation indicate that two accumulators and indexing are desired.

4.2.1.2.2 Word Length and Banking Size

The requirements for Mars orbital phase used above (from Appendix 1) were evaluated with the consideration of computer word length. Two word lengths were considered: 12 bits and 18 bits. The requirements were initially determined without taking into account word length. It was assumed that two accumulators and one index register were available in each case. The results of this evaluation are given in Table 4-2.

Table 4-2. Speed and Storage Requirements for Phase 12, Mars Orbital, With a 12 Bit and 18 Bit Word Length

Function \ Requirements	Storage (words) 12 bit	18 bit	*	Speed (ops/sec) 12 bit	18 bit	*
Navigation and Guidance	9373	6143	4348	140,196	101,202	46,732
Telecommunication	7730	5730	4700	26,000	15,000	13,000
Scientific Experiments	11194	6840	8561	272,421	245,000	252,496
Status Monitor	4260	3560	3560	8,400	8,200	8,000
Total	32557	22273	21169	447,017	369,402	320,228

*Indicates no consideration of word length.

Storage and speed requirements increased in the Navigation and Guidance function primarily due to the need for triple precision with a 12 bit word and double precision with an 18 bit word. The requirements for the Scientific Experiments function decreased somewhat with an 18 bit word primarily due to the ability to make use of half word storage and operations, these gains due to half word capabilities are eliminated with the 12 bit word.

Based on the above evaluations, the 18 bit word was selected since it provides programming ease and reasonable processing speeds while using little extra memory. This will be explained below. It should be noted that the reasons for the exact format of the instruction word have not been defined yet, the remainder of this evaluation section will complete the evaluation of the format. In addition, it should be mentioned that the 18 bit discussion here applies equally well to a 16 bit word discussion. The reasons for the exact word length selection, 16 or 18 bits, are given below in 4.2.1.2.3 of this section where an evaluation of numerous machine features are presented.

This discussion presents the reasons behind a choice of a word length in the vicinity of 18 bits rather than one in the vicinity of 12 bits. An 18 bit instruction word leaves room for the inclusion of 64 operation codes. This means that no double length instructions will be necessary since the 64 operation codes with extension provide sufficient instructions to take advantage of multiple accumulators and index/bank registers.

The advantage of this word length can best be understood by looking into the difficulties encountered with the 12 bit word. A 12 bit instruction word would require a double length instruction capability if indexing, multiple accumulators, indirect

addressing, etc., are to be included. From the discussion in 4.2.1.2.1 above it is seen that elimination of these features would result in approximately a 20% increase in storage. On the other hand inclusion of these features necessitates double length instructions which result in approximately a 30% increase in storage. This eliminates some of the advantages of the 12 bit word in attempting to reduce storage. The Navigation and Guidance data is greater than 24 bits and as a result would have to be operated upon in triple precision with a 12 bit word. This also increases storage.

The results presented above in Table 4-2. showed that a 12 bit word provided less than 1/8th decrease in the number of bits of storage and in fact gave a 1/3 increase in the number of words. This increase in the number of words can result in more circuits required for the 12 bit memory. The complexity of a 12 bit processor with double length instruction capability versus the complexity of an 18 bit processor with only single length instruction capability is about the same. In addition, the 12 bit processor would need to be faster. As a result, there are very little if any hardware gains by using a 12 bit word. An 18 bit word was therefore selected, thereby also providing a more flexible machine to the programmer (no triple precision operations or double length instructions to worry about). It should also be noted that a larger instruction word was not considered since not only is it not warranted by the data requirements but also a 7 bit bank causes no significant inefficiency penalties.

The choice of the 18 bit word leaves the possibility of using a 7 or 8 bit bank. A 7 bit bank was selected, some of the considerations involved in this selection are given below.

There is some increase in storage and execution time when going from a 256 word bank to a 128 word bank size. Past studies at Autonetics on some navigation and guidance routines indicated that a 128 word bank resulted in only a slight increase in inefficiency over a 256 word bank. In addition, in the program used in the evaluation here, there appeared to be little inefficiency in this bank size. One of the primary reasons there appeared to be little or no problems was due to the full length I/B registers since they did not result in rigid bank boundaries every certain 128 words. In particular, when indexing with these full length registers there is practically no effect due to the bank size. Most of the problems in going from a 256 word bank to a 128 word bank then arise from non optimized location of data. However, an optimum banked assembler or forcing the programmer to handle the data banking optimization could reduce this problem considerably. It is therefore recommended that a 7 bit bank be used.

4.2.1.2.3 Accumulator Tag, Register-Register Operations, Indirect Addressing, and Index/Bank Register Schemes

The above features were evaluated by investigating their usefullness in a number of programs which were considered as representative of the computational functions. These programs included: (1) Navigation and Guidance: Star Tracker Pointing, Body to Inertial and Locally Level to Inertial transformation Matrix, and Kalman Filter Computations, (2) Scientific Experiments: zero order polynomial predictor, orthogonal polynomial series, and quantiles computation, and (3) Status Monitoring. Details on these programs may be found in Section II, 2.8.

These programs were mechanized assuming all the above features were available. Then the programs were reevaluated with certain restrictions so as to obtain asnwers to: (1) if there is no accumulator bit in the instruction word, what

additional instructions are required?, (2) what instructions are used in register-accumulator and accumulator-accumulator operations?; (3) what is the effect of not having indirect addressing?; (4) what is the effect of the index/bank register schemes, i.e., any 5 I/B registers taken 0, 1, or 2 at a time, 1 B bit and 2 T bits giving any one of 2 registers taken with any one or none of 3 other registers, and 1 B bit and 3 T bits giving any one of two registers taken with any one or none of 7 other registers; and (5) what is the effect of not having double index/banking (single level only).

The answers to the above questions were used to aid in selecting between a 16 and 18 bit instruction word and also for the particular instruction word format (see paragraph 4.2.1.1 for the formats considered). A summary of the results will be given below, detailed discussions are given in Reference 18.

1. Accumulator Bit

   The following instructions used the accumulator bit in the routines programmed.

   Load
   Store
   Add
   Subtract
   Multiply
   Divide
   Compare
   Jump on Conditions
   Sum of Products - Multiply
   Jump on Minus or Zero
   Logical "And"

   A considerable number of instructions made use of the second accumulator. It is therefore recommended that some means of providing for identifying accumulator 1 or 2 be provided for those instructions that could use either accumulator.

2. Instructions between Registers and Instructions between Registers and Accumulators

   Instructions used (Register-Accumulator) or (Register)

   Store
   Load
   Subtract
   Add
   Multiply
   Shift
   Absolute Value
   Compliment

   Instructions used (Accumulator-Accumulator)

   Add
   Subtract
   Multiply
   Divide

Sum of Products - Multiply
Square
Load
Store

A considerable number of instructions made use of communication between the accumulators. Also many instructions were used between the registers and the accumulators although not as many as between the accumulators. It is recommended that full arithmetic and transfer instructions be provided between the accumulators themselves and also between the accumulators and registers. There was no apparent need for instructions between the registers themselves, therefore this feature is not recommended.

3. Indirect Addressing

Although limited use was made of indirect addressing in the programs evaluated, where it did occur (primarily for subroutine linkage) savings of between 6 and 8 percent in storage was achieved for some programs. The savings in timing were negligible however.

Indirect addressing is of considerable importance when it is desired to do list processing. This is due to the fact that the link addresses normally encountered in lists can be considered to be indirect addresses to the following words in the list. The need for list processing is not apparent in the present requirements. Therefore, the merits of indirect addressing due to list processing will depend on future requirements, if any, for list features.

It is recommended that indirect addressing be included since there are some present indications of its merits and the combination of its inherent flexibility of programming and potential to future requirements make it a promising feature.

4. Index/Bank Register Schemes

The preferred approach is to use 2 + 7 registers. The primary reason for this choice is the availability of nine registers versus only five with the others. In code sequences where multiple banks and indexes require more than five current registers, the housekeeping involved in storing the recovering register values went as high as 9% in storage and/or timing.

The approach using 5 registers with any combination 2 at a time shows some advantage in coding where five or less registers were used. The savings, however, are in the 3% range for storage and less than 1% in timing.

It should also be noted that the approach using 2 + 7 registers also provides additional facilities for temporary storage when not all 9 registers are used for indexing/banking. It is also worthwhile to mention that the coding which uses a large number of registers involves multi-level iterations and a large data base. Therefore in these programs the loss in efficiency due to less registers shows up more pronounced.

5. Double Index/Banking Capability

Without this capability some of the scientific experiements programs were somewhat lengthened, some of these programs had execution time increased by 50 percent. This capability is very useful in programs involving matrix operations. It is recommended that this capability be included.

4.2.1.3 Requirements and Organizational Considerations

There are a number of requirements that influence the computer design. These have been rated as reliability - 100, power - 10, flexibility - 4, and all others 1. Essentially this says that the computer organization should try to take advantage of various schemes to increase reliability and lower the power in light of the various computational requirements. For reliability there are two basic considerations, one is an availability requirement of 0.997 for the whole mission. This can essentially be interpreted as a level of satisfaction for the mission. In other words, if the computer system is operating 99.7% of the time the desired degree of success will be obtained. Another important consideration is a probability of success of 0.997 for critical mission phases. The computer system must be able to function during these critical phases in order for the mission to be completed. During these critical phases 5 seconds are allowed to reconfigure to a second operating system if the primary system fails. These numerical reliability values were obtained from an examination of the references cited in Section 2.1.

The above availability and probability of success requirements have a profound influence on the system. In particular in order to meet the probability of success during the critical mission phases some type of on-line back-up must be available. During non-critical phases average times of 1/2 hour can be allowed to get back on-line; as a result a repair replacement ability is sufficient.

One of the ways the importance of reliability and power affects the computer system is in terms of the amount of hardware that can be kept off during any phase. Data from Autonetics experience tends to point toward the fact that computers that are not on-line have reliabilities on the order of 3 to 10 times or more greater than the on-line modules; however this number has not been validated by a thorough analysis. In fact it is not even clear if the off-line modules should be turned off or if bias power should be maintained. The above dictates that equipment be kept off-line as much as possible. Therefore the computer systems are designed so that during long phases a good portion of the computing hardware can be turned off. This should not only increase reliability but also save a significant amount of power.

It should also be mentioned that there will be separate computer systems in the Mars Mission Module and in the Mars Lander. These two computer systems should be the same type since that will provide a simple sparing philosophy and will lower development and production costs. Sparing can be simplified since the Lander computer system must only be on-board the Lander and functioning correctly from Earth to Mars and while in the Mars area. On return, a Lander computer system can be pulled off and placed aboard the Mars Mission Module to function as spares for the return to Earth.

An interesting point to notice about the computational requirements is the variation of speed and storage throughout the various mission phases. In particular, during the long Trans-Mars and Trans-Earth phases the storage and speed requirements are relatively low and at about the same level. As a result it is desired to

design the computer system so that much of the computation speed and storage could be turned off during these phases. Figures 2-8 and 2-9 also show the amount of storage that is absolutely necessary to have operating continuously throughout the Trans-Mars and Trans-Earth phases. The rest of the memory system could be turned on and off periodically as shown in order to save power and increase reliability. During phase 12 (Mars Orbital) the maximum amount of computation resources will be in the system; as a result this phase dictates the maximum need for computational resources.

4.2.2 Multi Computer Organization

4.2.2.1 Organizational Considerations

The past paragraphs have presented the requirements for the Mars Mission computation system and also some explicit features of the processors within this computation system. The following paragraphs present the multiple computer organization and features that were developed from the above requirements.

It should first be noted that a single computer organization can immediately be thrown out since 5 seconds back-up in case of failure during critical mission phases could not be provided. In order to provide this type of back-up a second on-line computer carrying out the critical computations must be available. A single computer would also need a very high MTBF, and would be power and reliability consuming during low computational load phases. Another disadvantage is that this type of an organization is unable to flexibly meet unplanned variations in computational requirements by addition or subtraction of the number of modules in the system.

4.2.2.1.1 Duplex Computer Approach

A duplexed computer approach as shown in Figure 4-4 was next considered. This approach is discussed in some detail in Reference 17. Briefly it uses two computers each capable of carrying out all the computations for the most heavily loaded phase of the mission. The computations are carried out in both computers and the results sent to the output switch. During normal operation the primary computer's outputs are used and the secondary computer does output comparisons; however if the secondary computer discovers a discrepancy it can carry out a software self-ckeck and if it passes a lengthy self-check process it will take over outputing information.

The supposed advantage of this type of an approach is in the ease and completeness of failure detection; however the development of operational hardware and software self-tests for a multiple computer, to be described next, has been evaluated to be not much harder than the development of non-operational self-tests for one of the duplex computers. There are also quite a number of disadvantages to this approach. These are listed below:

1. Each computer must be able to handle all the computations. This means two large power consuming computers.

2. During long relatively low computation phases, Trans-Mars and Trans-Earth for example, there will be no chance to lower power and increase reliability by turning a good portion of the system off.

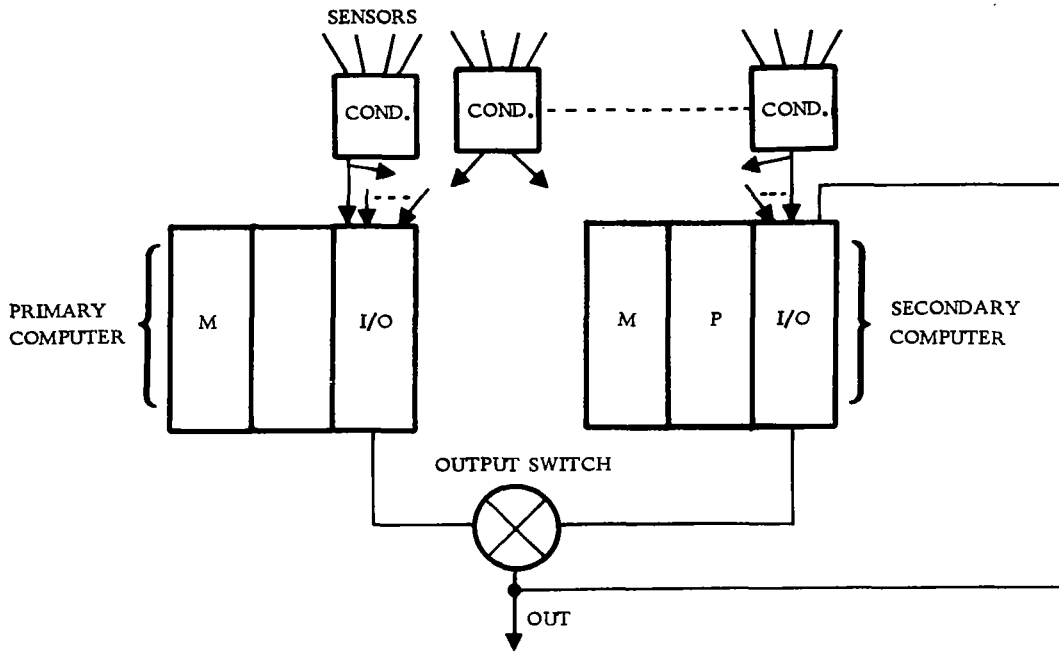


Figure 4-4. Duplexed Computer

3. After a failure of the primary computer there is no checking of the computation system; as a result a failure may occur and never get detected.

For the above reasons a duplexed computer approach was rejected.

4.2.2.1.2 Multi Computer Approach

1. General Considerations

The chosen multiple computer system will be a two computer approach where the computers operate separately each carrying out its own self check. At least two computers are needed in this system since there is a need to continue operation while one of the computers is in a state of repair or replacement. This computer approach, shown in Figure 4-5, was chosen because it is well suited to the system requirements as demonstrated by the discussion below. Clearly if the requirements were to be increased for other applicable missions additional computers could be added to the system.

A proper use of the above computer system during the critical and non-critical Non-Mars phases and the Mars-Orbital phases is discussed in a later paragraph. Very simply during Non-Mars non-critical phases only one computer will be in operation carrying out all the functions of the system. During the critical phases two computers will be on-line-the first one doing all of the system functions and the second doing a

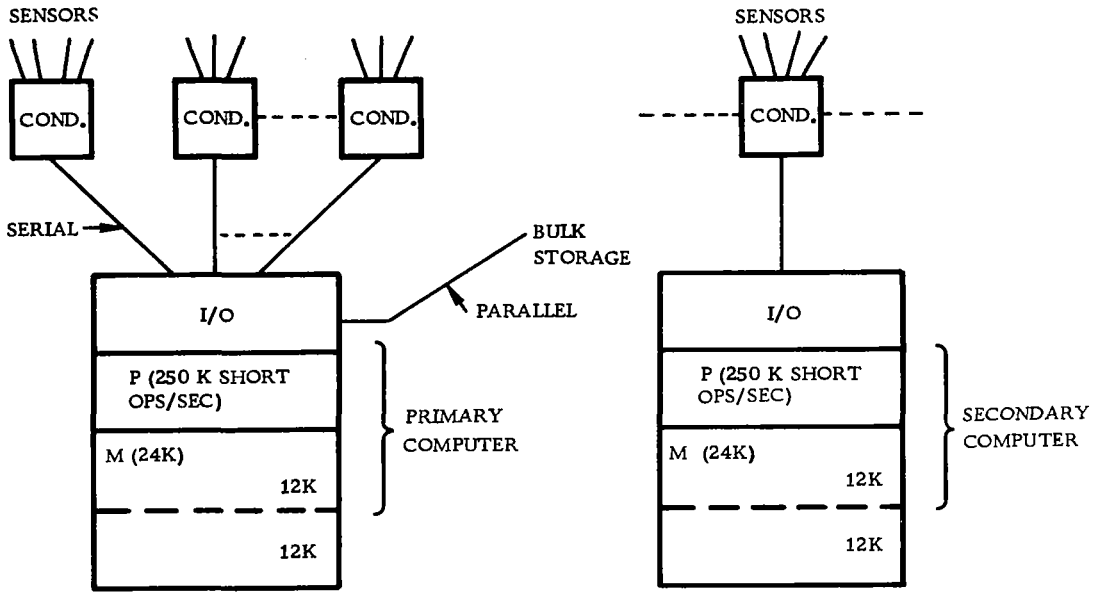


Figure 4-5. Two Computer Approach

redundant calculation of the critical information (navigation and guidance information). This method of operation enables a reconfiguration in less than 5 seconds. During Mars orbital operation two computers will be carrying out the computations. Reconfiguration in these phases is handled by a combination of repair replacement and switch-over to the non-failed computer.

The size, speed, and number of modules in a Multiple Computer approach is dictated by many factors. The most important of these for the Mars mission is of course reliability. For any given computational requirement the best reliability will be obtained by using the least number of computer modules (not less than two) as long as the speed requirements do not force the use of a less reliable memory and circuit technologies. This is clear by considering the fact that a lesser number of modules simply means less components for any given set of requirements. Another important influence on the size, speed, and number of modules in the system is the fact that off-line reliability is assumed to be much higher than on-line reliability. For the Mars Mission this can be interpreted to mean that it is desirable to turn off as many modules as possible during the Trans-Mars and Trans-Earth phases. As a result, the size of the modules should be adjusted so that during these phases the maximum amount of computation resources are turned off. A third influence on the

size of modules is the computational requirement for the highest computation rate phase, namely the Mars Orbital Phase. During this phase it is desirable to have all the modules in the system on and in use so that there will not be a lot of extra hardware in the system. Another factor not considered here that may have some influence on the size of modules is the requirements for the Mars Lander Module. These requirements have not been evaluated and as a result cannot be considered.

The above trade-off considerations dictated a processor size of approximately 250,000 short operations per second, and a memory per computer of 24,000 words. The memory is modular by 12,000 words as shown in Figure 4-5. The memory size was decided upon by examining Figure 2-8 in Section II. This figure shows that during the Trans-Mars and Trans-Earth phases, 4, 7, 14, 17, essentially 24,000 words are required if executive and I/O programs are included. This figure also showed that approximately 12,000 words of storage was needed to come in only intermittently. This was interpreted to mean that the best memory size for the Trans-Mars and Trans-Earth phases is 24,000 words with 12,000 word modules capable of individually having their power turned off.

Phase 12, Mars Orbital, was next analyzed for storage requirements. Two computers each with 24,000 words were determined to easily provide suffient storage since the actual requirements for this phase are approximately 30,000 words including I/O and executive programs. It was also determined that during the major portion of phase 12, one computer could have one of its 12K memory boards turned off thus allowing an increase in reliability and a decrease in power. This memory is also a convenient size for implementation with a thin-film NDRO memory in the 1973-1975 technology time frame. Using projected densities it enables getting a maximum amount of words per module and thus make best use of the drivers, receivers, and sense amplifiers.

The processor speed requirements for the Trans-Mars and Trans-Earth phases are relatively low in comparison to other shorter phases. Since circuit counts for processor implementations are not actually increased for reasonable increases in processor operating speed, it was decided to make one processor capable of handling the fastest non-Mars orbital phases. (This is true as long as the increase in operation speed does not require the use of a new circuit technology.) The processor, including execution of executive programs, was therefore made capable of handling 250,000 short operations per second. (See Figure 2-9 in Section II.) This also means that two processors will not only easily be able to handle the Phase 12 computation load but also will have extra computation power available during all phases so that they can catch up on the computations even if some reasonably long interruption occurs for a repair or replacement operation. In order to see that by 1980 a 250,000 operation per second processor is feasible to implement in MOS or MOS-SOS circuitry (or bipolar arrays), a calculation of the clock rate was carried out using 250,000 operations per second, 2 memory cycles per short operation and 4 clock pulses (bit times) per memory cycle. This calculation shows the need for a 2.0 microsecond memory read or write cycle for a NDRO

memory and a 2.0 megacycle clock for the processor. This certainly is reasonable for the 1975 technology time frame as expressed in Section III, 3-1.

The computer configurations for the multiple computer during the various mission phases are tabulated below. As mentioned above both the computers and the 12K memory modules within the computers are capable of separate turn-on and turn-off.

Phases 1, 2:	Computer 1, 1 memory module
Phases 3, 5, 6, 8, 9, 10, 11, 13, 15, 16, 18, 19, 20:	Computer 1, 1 memory module Computer 2, 1 memory module – active redundancy
Phases 4, 7, 14, 17:	Computer 1, 1 memory module on continuously, 2nd memory module on intermittently as shown on storage requirement graph – Figure 2-8.
Phase 12:	Computer 1, 1 memory module Computer 2, 2 memory modules

The distribution of the computations is relatively straight forward except for Phase 12. During this phase the functions are distributed as follows:

Computer 1	Telecommunications Status Monitoring Scientific Experiments (part) Update Minimal Nav. & Guid.
Computer 2	Navigation and Guidance Scientific Experiments (part)

In case of a failure during Phase 12 several actions may occur. If Computer 1 fails, Computer 2 proceeds and Computer 1 is repaired. A failure of Computer 2 will cause Computer 1 to enter into a minimal navigation routine which was continually updated by Computer 2, Computer 1 will also proceed with its other computations as normal.

If a spare computer or module is not available for repair, then the operative computer is reconfigured with a new program, this reconfiguration will contain:

Navigation and Guidance
Status/Monitoring
Telecommunications (reduced)
Scientific Experiments (reduced)

If Computer 2 failed under these conditions, Computer 1 will enter the minimal navigation and guidance routine until the reconfigured program is loaded with the normal navigation and guidance routine into Computer 1. Reconfiguration is further discussed in Paragraph 4.2.2.3.

Another consideration in the design is the power supply. A distributed supply has been chosen for this system. Distributed supplies appear to be the most reliable and easily implemented supplies for future time frames. This is partly due to the fact that a central supply needs large capacitors on each board of the computer system in order to supply constant power when transient switching is taking place. Distributing the supply by using a power supply per board eliminates these large unreliable capacitors. This distribution also enables each board to receive only the primary power level of the system. This combined with the above fact may mean that the distributed supply actually will have less components than a central supply system. The distribution of the power supply is made even more reasonable with the onset of the MOS and MOS/SOS circuitry, since these circuits allow microminiaturization due to their need for high voltages and low currents. Two other advantages of distributing the power supply are the ease of expanding the system and its power requirements and the ability to conveniently turn off all power to various sections of the system, for example a memory module. This power turn off can occur by the astronauts at phase changes or automatically after failures.

In order to implement this last feature a transistor switch will be used on the input to each power supply where appropriate. Such a switch for a supply is shown in Figure 4-6. In the actual implementation this switch may be a trans-switch (controlled by pulses) in order to provide isolation of system or battery ground and memory ground.

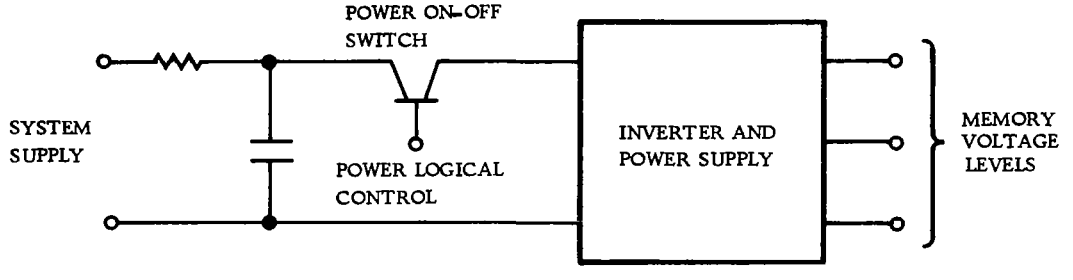


Figure 4-6. 12K Memory Board Supply and On-Off Switch

2. Processor, Memory and I/O Structure

From the above discussion and from the processor features given earlier a rough count can be obtained of the hardware necessary to implement the processor for one of the computers in this two computer candidate. This count as presented below was actually made with the aid of the processor design given in Section VI. Therefore a full understanding of the purpose of all the listed hardware can only be obtained by reading Chapter 6.

a. Two upper accumulators (36 bits)

b. One lower accumulator (18 bits)

c. One 15-bit program counter

d. Two 18-bit bank registers

e. Seven 18-bit index registers

f. One 6-bit instruction register and decoding for 64 op codes

g. One 18-bit memory register

h. One 4-bit tag register

i. One 5-bit shift register

j. One 18-bit parallel adder (a ripple carry adder is sufficient)

k. Bit time and mode clocks - 8 bits

l. Real Time clock (25 bits)

m. Fill clock (enables the processor to take up slack time in periodic programs with background programs) - 8 bits

n. Control Flip-flops - 14 bits

This gives a total of approximately 320 flip-flops for implementation of the processor. Using FET densities estimated for MOS/SOS in the 1975 time frame (approximately 5,500 FETS per 150 mils square), a rough approximation says this processor could be implemented on approximately two chips. Of course depending on the density and yield tradeoffs in this time frame, the chips may be slightly larger than 150 mils square.

The memory hardware estimate below is for a 24K 18 bit word NDRO magnetic memory. (For example today this memory would be fabricated from plated wire.) The complete memory is actually made up of 12K modules. A discussion leading to the decision to use an NDRO memory is given in section 6.1.2 along with block diagrams of memory systems. Section 6.1.2 also points out that this memory could be either a thin film magnetic approach or a semi-conductor approach. The circuit counts are for a magnetic memory assuming LSI circuits (Bipolar) can be used in the 1975 time frame.

1. Two 12K modules of 1,000 word by 216 bit lines.. Hardware per module:

   a. Word circuits - 16 LSI ckts.
   b. Bit drivers - 18 LSI ckts.
   c. Sense amplifiers - 18 LSI ckts.
   d. Decoders - 3 LSI ckts.

2. One Current Source per 24K

3. One LSI timing generator per 24K

4. One MOS/SOS or LSI chip for the data register (18 bits), the address register (15 bits), and read and write flip-flops.

The I/O section of the computer system is shown in Figure 4-5. Each computer has a central programmed controlled I/O section plus a number of connections to conditioners and high rate devices such as a bulk storage unit. The connections to the high rate devices are parallel and those to the conditioners are serial. The conditioners in turn are connected to a number of sensors and devices that handle both the inputting and outputting of data. There are a number of reasons for the present I/O structure. The trade-off was essentially carried out between the structure shown in Figure 4-5 (a central I/O unit of exactly the same form in each computer including standard format signals to a number of conditioners connected to these I/O units) and a completely centralized structure which would have the conditioning functions included in the I/O unit associated with each processor. The reasons for using the conditioner structure instead of the latter structure are given:

1. Typically a completely centralized I/O unit is used to get a more efficient use of hardware; however, in this system where reconfiguration is possible by disconnecting conditioners and connecting them to the second computer, a centralized I/O unit would have to have enough hardware to handle all the sensors. As a result a central I/O unit would not provide any hardware savings over the I/O unit using single copies of conditioners on-line.

2. The conditioner structure is also easily able to adapt to a change in sensors, addition of sensors, or improvements in the sensor design. All that is necessary is to add a conditioner or replace one that is already there; whereas in the completely centralized I/O structure there is a need to replace the complete I/O unit (the I/O unit will be just one MOS/SOS chip).

3. The conditioner I/O structure also provides ease of adapting the computer system to the Mars-Lander Module. This module will have significantly

different sensors from those on the Mars Mission Module. As a result the conditioner structure will provide the ability to use exactly the same basic computer with only the need to change the appropriate conditioners in this module.

As mentioned earlier a number of sensors must be handled in a strictly periodic fashion; as a result a program scheduler has been included in the design of this computer system. The scheduler is explained in paragraph 4.2.2.4. Basically the scheduler uses the processor real time clock to be sure that all periodic programs are handled at the proper rate and time. The background programs or nonperiodic programs are interleaved with those that are periodic. The calling of I/O variables can be handled in two ways: first, the programs can have a header associated with them that lists the I/O variables that must be called prior to program execution. When the program is brought into execution its I/O variables are then called by an I/O program. If processor waits for I/O variables can be limited, this is a very efficient method. The second way to handle I/O variables is to add two extra clocks or timers to the processor and use these in conjunction with the scheduler and Real Time Clock to call I/O variables early. This procedure has been programmed and is shown to yield a software overhead of 0.5% processor time per 100 programs per second. This extra software and hardware cost should not prove worthwhile in this system in light of the relatively small number of periodic programs that must be handled, and the low repetition rates of the periodic programs (20 times per second is the highest rate). The processor will also be given the ability to call I/O variables from a header and then leave the I/O unit to preempt memory cycles and thus take care of loading the data into memory. During this period the processor can execute a check sum on the fixed locations of the program and begin operating on the program until the need to use the data comes up. It should also be mentioned that the I/O units will receive interrupts and send these to the processor as necessary.

No need is seen at the present time for both of the computers to process a single job in parallel and thus have a need to exchange extensive amounts of data or program information. However, during certain phases of the computations a few words will be necessary to be exchanged between computers, such as, telecommunication interleaving control words and a few pieces of navigation and guidance data. As a result a serial channel from the I/O unit of the secondary computer to that of the primary computer will be included in the system. A rough I/O unit hardware count is given below. This only includes the section in the computer itself (not the conditioners). This count also assumes that the bulk storage unit has its own control unit with buffering on its outputs. The design of the I/O unit is similar to that for the multiprocessor; therefore, a description of the hardware is given in Section VI, 6.1.3

1. One buffer register (18 bits)
2. One assembly shift register (18 bits)
3. One memory register (18 bits)
4. Two memory address counters (15 bits each)
5. One 7-bit count register for controlling off-line transfers

6. One 4-bit count register for controlling on-line transfers

7. Seventeen control flip-flops.

This gives a total of 112 flip-flops. Using the same MOS/SOS densities as for the processor the I/O could be implemented on one MOS/SOS chip. In fact if high yields permit larger chips, the processor and I/O could be implemented together on two 200 mils square chips.

The explicit design of the conditioners can not be given here since the sensors, their characteristics, and their interface signals are not yet specified. However, a cursory treatment of I/O rates and sensors was given in Reference 17.

The redundant calculation of certain computations during critical phases has been discussed; however the output switching of the critical conditioners has not been made clear. Figure 4-7 shows the output switching for a primary computer and a secondary computer during a critical mission phase. These two computers can be assumed to be the primary and secondary computers in the Two Computer implementation, or the two separate sections of the Multiprocessor during a critical phase (see paragraph 4.2.3). Figure 4-8 shows how the logic levels for control of the output switch are generated within each computer. The switch is initialized with the primary computer in control. If a failure occurs in the primary computer, the secondary computer will take over and continue operating. If there is a failure in the secondary computer while the primary computer is still failed, all signals to the outputs will be turned off. The BITE circuitry is discussed in paragraph 4.2.2.2.

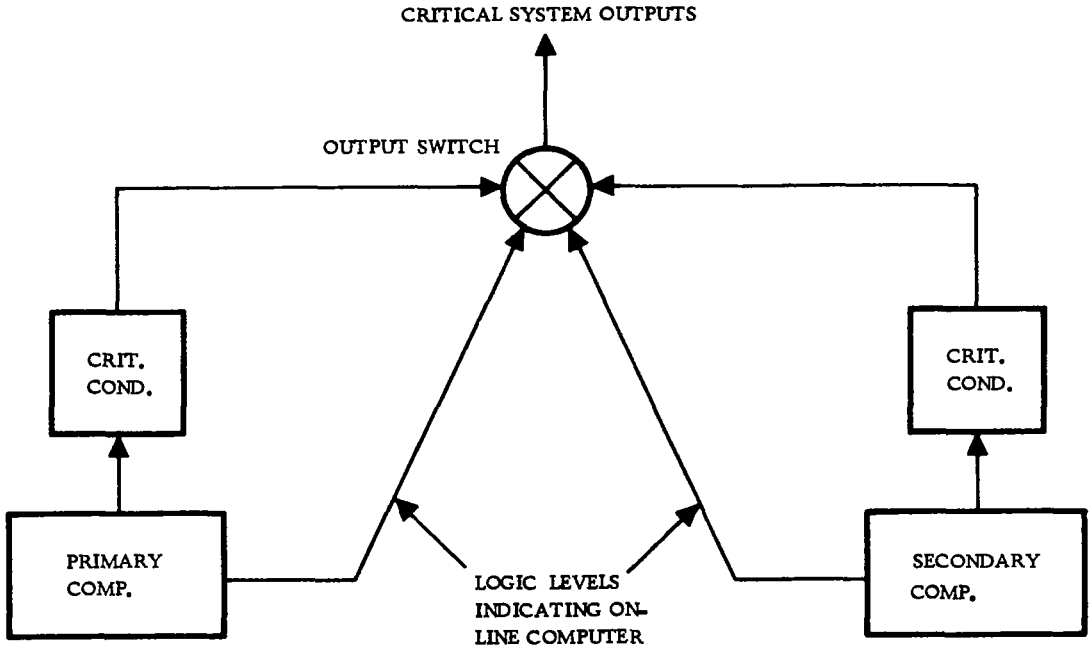


Figure 4-7. Output Switching of Critical Conditioners

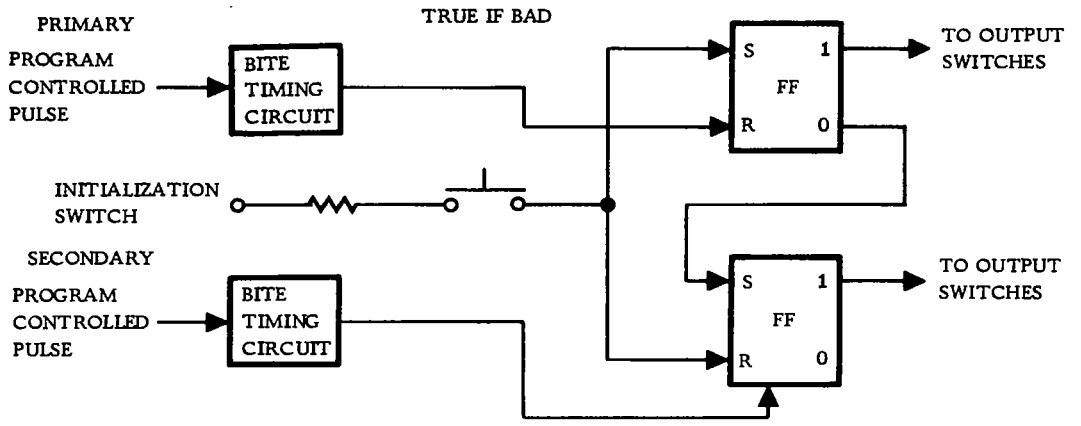


Figure 4-8. Logic Levels for Control of Critical Conditions

4.2.2.2 Failure Considerations and Reconfiguration

The ability to reconfigure the spaceborne multiprocessor in the event of equipment failures is required in order to meet the probability of mission success and availability goals. This section discusses the design of this capability for the Multiple Computer System.

The discussion is presented in the following five parts:

1. Basic Guidelines
2. Error Detection and Isolation Tests
3. External Status Reporting
4. Reconfiguration
5. Backup Equipment Assurance

The first part enumerates the important ground rules that are basic to the approach taken. The next two parts discuss the problem of performance assurance and its reporting which would signal the start of reconfiguration. The discussion of reconfiguration relates the actions required for reconfiguration as a function of the mission phase. The final part discusses testing of backup equipment to assure its readiness when reconfiguration is required.

It should be mentioned here that the approach discussed in Section IV for failure detection and isolation tests is based primarily on a software approach. This holds true for all three of the candidates. If solid failures are assumed the software approaches will be adequate. However, if intermittent type errors are considered to be of any significance (by this is meant errors that result in faulty conditions existing only for short periods of time, on the order of microseconds), then hardware failure detection may be necessary. To complete this topic hardware failure detection methods will also be considered for the selected candidate. It should also be noted that the considerations on reconfiguration were thought to be more difficult or worst case in some respects with software failure detection methods (hardware methods may offer greater ease of fault isolation) and therefore a broader spectrum of reconfiguration problems has been assessed.

4.2.2.2.1 Basic Guidelines

1. Equipment is less prone to failure when it is turned off, rather than when it is on. Therefore, where equipment is not needed by the computational requirements of a particular mission phase, it is desirable to turn it off.

2. Similarly, power is conserved by turning equipment off when not needed.

3. At worst, the time from the occurrence of an error to the time the system is reconfigured and properly functioning should not exceed 5 seconds. Generally this minimal time applies to the critical mission phases and can be much longer for non-critical phases.

4. Crew participation can be considered for the following functions:

   a. Reconfiguration during non-critical phases
   b. Turn-on and requests for checkout of idle standby equipment
   c. Replacement of a failed equipment with a spare, verification of the repair, and insertion of the equipment back into the system.

Within the framework of these basic guidelines the goal, for each candidate configuration, is to achieve 100 percent error detection capability and subsequent reconfiguration which maximizes the probability of mission success and the availability of the equipment. (Failure detection based on solid type failures will be primarily considered here.)

4.2.2.2.2 Error Detection and Isolation Tests

The following paragraphs describe the tests required to insure timely indications of the multiple computer system status during the mission.

1. Memory Check Sum

   The memory check sum routine simply adds the contents of fixed storage locations (instructions and constants) without regard to overflow and compares the result with the prestored correct response. The function of the test is to check for potential malfunctions in the computer memory and processor.

   The check sum routine could be written to add all of fixed storage at one time. This method was not chosen because of programming inefficiencies

which would result from having to keep track of which blocks in memory contain fixed information and which contain variable information. Instead a check sum routine would be built into each major programming segment and would be performed at the outset of the segment and possibly also at the conclusion, time permitting. Parameters such as the starting address, number of locations to be added, and expected check sum response are included as part of the program segment package. Initialization, execution of the check sum, and checking of the response would be handled by a utility routine. With indexing and the appropriate index test, decrement, and transfer instruction the check sum execution can be handled by a two instruction loop.

2. Arithmetic Section Functional Test

This test checks the performance of the arithmetic section logic circuits of the processor. No special test instructions are envisioned; therefore, no additional hardware would be designed into the system to perform this test. Patterns for exhaustively testing the arithmetic logic are prestored in memory and under program control act as stimuli to the logic. The responses of the logic are compared with prestored correct responses to determine the status.

Based on previous experience in writing this type of test, it is estimated that for this application the test would require 425 instructions and 75 constants and temporary storage locations. For a 4 μsec add time the test would run for about 2 msec. The degree of completeness, or the ability of this test to detect arithmetic section errors is expected to be high, say about 99 percent. Of course, proving this would require a thorough analysis which involves determining likely component failure modes and the ability of the test to detect the effects produced by the component failure modes.

The test is performed at a periodic rate. Its frequency would be adjusted to insure that the worst case reconfiguration time of 5 seconds during critical phases would be met.

3. Program Control Test

This test checks the ability of the computer to execute instructions in a legitimate operational sequence. Computer malfunctions which produce effects that are described by saying the computer is hung-up within an instruction, within a loop of random size, or wandering aimlessly through instruction sequences, would be detected. Malfunctions producing such effects can originate in the control logic of the processor, the memory, the clocking system, or the power supply.

Implementation of this test requires insertion of built-in test equipment (BITE) to mechanize a timing device. As an example, a digital timer would operate as follows: Under program control, a periodic square wave is set up and acts as input to the timer which consists of counters and associated logic. Tolerances are set on the duration of the "high" and "low" portions of each cycle of the square wave and on the period. The inability of the computer to provide this prescribed square wave, which would occur in the presence of a control error, would be detected by

wired-in logic associated with the counter and result in the setting of an error flip-flop indicating a computer failure. The period of the square wave and the associated tolerances would be determined to satisfy the worst case reconfiguration time requirement of 5 seconds.

From the programming point of view, periodically, an instruction has to be executed to effect the high portion of the wave, and a prescribed time later another instruction is executed to effect the low portion.

4. Input Signal Tests

Tests performed on input signals can detect failures due to errors in sensors, in data transmission, in input signal conditioning circuitry, or in transferring the signal through the input section of the computer to either the arithmetic section or the memory. Where tests are performed during normal operation of the system (on-line) the stimuli are not "canned" as they are in the case of arithmetic section tests since the sensors are not interrupted to provide prescribed input signals. In place of prescribed sensor values for testing purposes, the validity of these signals can be tested within the arithmetic section of the processor by a combination of the following techniques: reasonableness tests, dual redundant inputs, and BITE. Reasonableness tests use criteria such as the expected range and/or rate of the input parameter for error detection. Redundant inputs allow the disagreement between the inputs to provide error detections BITE in the form of input conditioner built-in stimuli under program control provides a backup of reasonableness tests and redundancy both for enhancing the error detection capability and for error isolation. The redundancy technique is the least desirable due to reliability and power considerations and would be used selectively, only if a study of the proposed reasonableness tests, BITE, and the criticality of the input signal indicate it is necessary.

Given that errors will be detected by the above mentioned techniques, the isolation problem is to determine if the input device, I/O conditioner, or computer is the error source. It is assumed that the input device cannot monitor its own status completely and will require computer participation for its status determination. It is further assumed that if digital transmission errors represent a significant problem, it would be handled by simple parity checking. A description of the detection and isolation process follows.

Included in the program segment requesting an input is the test required to verify it. If the input is acceptable normal operation continues. If the input is found to be in error, the error status is recorded in an assigned bit position of a status word in memory. (Assume one status word is reserved for each I/O conditioner thereby allowing reference in this description to "I/O conditioner status words".) Normal operation continues, even in this error case, except that the previous value of the input is used in the computations in place of the present value. At a prescribed point in the program, the executive looks at the contents of the I/O conditioner words. If this is the first cycle in which an error has been detected, the executive permits performance of at least one more input cycle. Note that the number of input cycles resulting in error reports should be greater

than one (1) since there is little likelihood that an error will occur at the start of a cycle. But, once having occurred, if it is a solid failure, it will be present throughout all subsequent input cycles and its effect will be truly represented by the I/O conditioner status words.

Next, consider the manner in which the I/O conditioner status words can be used to isolate the failure once the failure history is complete. Basically the process is closely coupled to the function of the failed circuitry. If the failure occurs in circuitry peculiar to a particular input, only that input signal will be affected and only one input will be flagged in one of the I/O conditioner status words. Such errors are either in the sensor, the transmission path between sensor and conditioner, or in the conditioner prior to the point where inputs are multiplexed. If failures are indicated in more than one input signal, the failed point must be in time-shared circuitry. This could be in the conditioner between the point where inputs are multiplexed and its output to the computer, the transmission path to the computer, or in the computer input circuitry. (An additional source could be a gross sensor error where the sensor provides more than one input signal and all have been affected. Such specific cases can be checked for by the executive program if the sensor cannot be depended upon to provide such information.) In the multiple computer configuration more than one conditioner is tied to the computer input unit; therefore, errors in the computer's input circuitry will affect most input signals.

Thus, it can be seen that the number of input signals and their relation to one another can provide a certain degree of isolation of the error. This degree of unambiguous isolation is related to the failure rates of the components within the isolable boxes that can be associated with each effect. If all inputs were bad, one would suspect the computer input unit; if the bad inputs were associated with one conditioner, one would suspect the conditioner first even though there is circuitry within the computer input associated only with that one conditioner, etc.

From the programming point of view, each input has associated with it certain parameters and tests employing those parameters. Tests on operational inputs are performed at the rate the operational program requires the inputs. Tests on non-operational inputs such as those supplied by BITE test signals are performed at a periodic rate. Detection of failures result in status notification by means of I/O conditioner status words in memory. The executive program interrogates these status words each cycle. A full cycle fault isolation routine is entered after the true failure history has been recorded in the status words. Isolation to a sensor, an I/O conditioner, of the computer input unit is achieved.

5. Output Signal Tests

In order to automatically detect errors in output signals, the loop on these signals must be closed. For this reason, all conditioner outputs are fed back to conditioner inputs and thereby made available for checking within the arithmetic section of the processor. As opposed to input signal verification by means of reasonableness tests, output signals are known at the time they are commanded. Therefore reasonableness tests are not required. All comparisons can be done digitally. Thus, for example, the output

voltage derived from a digital output word can be brought back into the conditioner, converted A to D, and the resulting digital input value compared with the original digital output value.

The programming requirements for output signals are similar to those for inputs. Associated with each output signal is a test which involves executing an input command for the I/O conditioner input channel reserved for the feedback of the output, and a comparison of input and output digital values. Test failure results in notification by means of I/O conditioner status words and a possible suspension of this output (note that for input errors past values were used while accumulating the failure history. The same, of course, cannot be done for output errors). The executive interrogates the status words each cycle. When the failure history is completed a full cycle fault isolation routine is entered and the error is isolated to either the computer output unit or to the I/O conditioner.

4.2.2.2.3 External Status Reporting

The status of the multiple computer system is continually reported to the space crew by means of control panel indicators. In the case of a failure they also provide the information to expedite off-line repair.

The isolable units are computers, I/O conditioners and input devices. As mentioned previously, the need for the computer to isolate input device errors is probably required in addition to its normal status monitoring of status signals generated by external devices.

The following tests have been described to monitor performance:

1. Memory Check Sum
2. Arithmetic Section Functional Test
3. Program Control Test
4. Input Signal Tests
5. Output Signal Tests

Failures detected by the first three tests imply isolation to the computer and, therefore, can be used to control a "computer fail light." The simplest implementation would be to have the program control test failure activate the light and to have failures in the memory check sum or arithmetic section test cause a failure in the program control test (as, for example, by executing a halt command in the event of a failure).

Input tests involve error isolation to either a computer, an I/O conditioner, or an input device. Output signal tests involve isolation to either a computer or an I/O conditioner. In these cases one method of failure notification is to have an "input or output fail light" and a status word display to indicate the computer input, computer output, I/O conditioner, or input device as the most likely error source. The executive program would control these indicators, and in cases where I/O conditioners or input devices have failed, but operation of the system can continue (this point will

be discussed in more detail in the section on reconfiguration that follows), the executive is responsible for terminating calculations involving the failed units.

4.2.2.2.4 Reconfiguration

This section discusses the task of reconfiguring the multiple computer system in the event of a failure. It is assumed that the failure has been detected, correctly isolated, and reported to the flight crew. Of course a spare must exist. If not, depending on the failed item, the mission may fail.

From the point of view of determining a reconfiguration plan or strategy, the mission can be divided into three types of phases: non-critical, critical, and Mars orbital. The plans for each of these phase types differ because of the speed of reconfiguration required or desired and because of the allowable status of the system during reconfiguration.

1. Non-Critical Phases

   During non-critical phases the primary system, comprised of the primary computer and associated I/O conditioners and I/O devices, is performing the required mission functions. The secondary system is turned off except for periodic intervals at which it performs self-checking functions. The composition of the secondary system is determined as follows: First, in anticipation of becoming active during critical or Mars orbital phases it contains the secondary computer and associated I/O conditioners and I/O devices required for the particular phase. Second, in anticipation of a failure in the primary system it can contain additional I/O conditioners and I/O devices to form a source of verified spare units.

   In the event of a failure in the primary computer, the astronaut would shut down the primary system. The secondary computer would assume its role. Physically, either the primary computer would be removed and replaced by the secondary computer, or the connector(s) to the primary computer would be disengaged and connected to the secondary computer. The power to the "new" primary system is then restored, the system is checked, and then the mission functions are resumed.

   In the event of a failure of an I/O conditioner in the primary system, either the entire primary system is shut down while repair is effected, or, provisions may exist to remove the I/O conditioner and allow the rest of the primary system to remain in operation. Physically, the failed I/O conditioner is replaced by a "like" I/O conditioner from the secondary system, the new conditioner is checked, and then its mission function is resumed.

   Failures in I/O devices may be repaired by a method similar to that described for I/O conditioners.

   The time required to affect repairs in the manner described above is a function of the accessibility and mounting of the units, and of the ability of an astronaut to perform physical repair actions in the environment of the Mars Mission Module. Extensive data on astronaut repair capability is a mission function of some current space programs and would be available for use in developing plans for the Mars landing mission. Present data

indicates that space maintenance is feasible in a zero G environment with or without a spacesuit. For evaluation purposes, as described in Section 5, it has been assumed that the space crew can perform repairs within 30 minutes.

Restoration of the secondary system after its equipment has been used to reconfigure the primary system will be discussed in the section dealing with backup equipment assurance.

2. Critical Phases

During the critical phases the primary system is performing both critical and non-critical mission functions and its outputs are controlling the vehicle. The secondary system is also turned on, performing the critical navigation and guidance functions in an active standby redundant mode. In addition, it is performing checking functions on spare I/O conditioners and devices. The time duration of the phase is comparatively short, never being longer than about 40 minutes.

Failures in the computer or certain of the I/O conditioners or devices of the primary system that affect the critical functions will require rapid reconfiguration. In this event, control of the vehicle is automatically passed to the secondary system by issuance of a logic level derived for the BITE circuitry associated with the program control error detection of the primary computer. This circuitry is functionally described in paragraph 4.2.2.1. Basically the essential event that occurs is activation of the secondary system outputs and deactivation of the primary system outputs. This concludes the reconfiguration for a critical failure in the primary system. The time to effect this reconfiguration is expected to be much less than the allowable maximum time of 5 seconds. The primary system is then shut down, either automatically, or manually by the astronaut.

Failures in non-critical I/O conditioners or devices of the primary system result in no immediate reconfiguration of hardware. Instead, the error, having been detected is reported to the flight crew for repair action to be performed at the conclusion of the critical phase, and the non-critical calculations that are affected are suspended. The rationale for this plan assumes that the primary system, without the capability of performing certain non-critical calculations, can still perform more mission functions than would be performed by the secondary system after an automatic switchover.

3. Mars Orbital Phase

The Mars Orbital Phase requires the maximum storage and speed capability to perform the mission functions. To accommodate this, the load is shared by the two computers and associated I/O conditioners and devices. The two systems shall continue to be referred to as the primary and secondary for consistency, even though in this phase neither system is devoted to a standby or backup role.

In normal operation the primary system is performing navigation and guidance and a part of the scientific experiment functions. The secondary system is performing status monitoring, telecommunications, the remainder

of the scientific experiments, and minimal navigation and guidance for backup purposes. By minimal is meant that portion of navigation and guidance which would be performed by the secondary system in the event of a loss of the full navigation and guidance function of the primary system, in order to facilitate subsequent restoration of full navigation and guidance during reconfiguration.

The manner in which reconfiguration is handled in this phase is contingent on the availability of a spare for the failed unit.

First, assume a spare is present. Consider the secondary system. A failure here is handled in a similar way as described for a non-critical phase failure of the primary system. The only difference is the source of the spare. It can either be obtained from spares stored, or, from an on-line configuration if spares are tacked onto the primary and/or secondary systems and periodically verified by test programs. Next, consider a primary system failure. As part of the normal operating program the primary computer is transmitting the latest updated navigation and guidance data to the secondary computer via the intercomputer communication link in anticipation of a primary system failure and the subsequent performance of the navigation and guidance function by the secondary system. As a practical matter, the secondary computer would store at least two sets of such data; one being the latest set and the others being the sets prior to it. Then when a failure of the navigation and guidance function of the primary system was signalled, the secondary system can start its minimal navigation and guidance function using the set of data most likely to be correct. Switch-over to the secondary system may be made automatic in order to assure the presence of a good set of navigation and guidance data in the memory of the secondary computer. If the primary system failure does not affect the navigation and guidance function, switchover need not be automatic and the reconfiguration of the primary system is handled similar to that for non-critical phases. Of course, if as part of the repair activity it were necessary to remove the navigation and guidance function, the secondary system would have to be told to assume that role.

Next, assume that a spare is not available. If a computer failed, the tasks of the operating computer would be reassigned to perform full navigation and guidance, reduced communications, full status monitoring, and reduced scientific experiments. Physically, aside from shutting off the failed system, reconfiguration would also entail connecting the appropriate I/O conditioners and associated I/O devices to the operating computer if they were not already linked there in a standby manner. If I/O conditioners fail, many reconfiguration possibilities exist depending on the commonality of I/O conditioners and the preference of performing certain tasks in lieu of others.

4.2.2.2.5 Backup Equipment Assurance

The basic functions assigned to backup equipment are to enable rapid reconfiguration during critical mission phases in order to enhance the probability of mission success, and to provide a source of verified spares generally during non-critical phases in order to increase system availability.

In order to assure the readiness of backup units they must be tested. It is undesirable to continually test them because of power considerations, and also because of reliability considerations if it is assumed that the failure rate of equipment varies directly with its usage. Therefore they would be tested periodically.

The configuration of the backup, or what has been called the secondary system, is predicated both on the anticipation of a failure in the primary system and on the expected role of the secondary system.

During the major part of non-critical phases the secondary system contains the secondary computer, and those available I/O conditioners and devices which can be inserted as spares into the primary system in the event of a failure of the primary system. The secondary system could consist of a complete duplication of the primary system if the duplicate set of equipments existed. Where there is commonality of equipments as may be true in the case of I/O conditioners, one of each type is all that is required. Where duplicate I/O devices either do not exist or it is not feasible to connect them to the secondary system, the associated I/O conditioners and secondary computer I/O unit can still be tested when a "loop back" capability is provided to route I/O conditioner outputs back to I/O conditioner inputs via a special test cable. Thus the nature of the backup configuration can be seen to present a certain degree of flexibility. No attempt to pin it down will be made during this study (for any of the candidates).

Checkout programs for the secondary system when it is in this "sparing" role would be organized similar to the operational checkout programs. The program control BITE would operate continuously. The arithmetic section functional test would be the same, but can be expanded if required. The memory check sum would be included with each test program segment and could be expanded to check sections of memory and processor controls not specifically needed by the test programs. The input/output test would depend on the I/O configuration. The entire testing could be initiated on-demand by the operator and be continually recycled. The operator can be assisted by having the primary system mark time between checkouts and indicate to him when to initiate the checkout of the secondary system. The secondary system can mark time to indicate when the required number of test cycles have been completed and when shutdown of the secondary system is to be performed.

In a prescribed period prior to entry to a critical phase the secondary system must contain at least those equipments necessary to enable the secondary system to take over the primary system role in the event of a primary system critical failure. This will allow the required rapid reconfiguration during critical phases.

In these time periods the checkout programs for critical equipment may be identical to those performed by the primary system for critical equipment. Checks of spare I/O conditioners and devices may be included as part of background calculations.

During the Mars Orbital phase, with the secondary system active, operational type testing would be employed. Spares can be tested both in the primary and secondary system if the required computing power is available.

Restoration of a full backup capability, during any portion of the mission, whether it be due to a failure in the primary system for which the secondary system supplies the spare, or whether it be due to a failure in the secondary system itself,

is accomplished manually by the flight crew. Performance of the repair actions is similar to that previously described for a primary system failure in a non-critical phase.

4.2.2.3 Software Considerations

The computational requirements of this mission were used to determine certain basic software design criteria. No matter which type of computer system is finally selected these criteria will still be valid.

1. Computational Characteristics

   There are a number of obvious features of the required computations that influence the software approach.

   The greatest difference between this mission's needs and that of most present-day projects is the wide variety of computational classes. Precise calculations for navigation and guidance, large data processing functions and others ranging in between these extremes, all with varying timing constraints, must be concurrently executed.

   There are certain processes that must be performed throughout the entire mission and others that will be done once. Additionally, the duration of the mission and the nature of program longevity imply that unanticipated computations will be added to the work load of the computer system.

   Another characteristic of these requirements is the wide range of computational loads between phases. In actuality, even within a given phase the load can vary quite radically.

2. Software Criteria

   In view of these characteristics a number of conclusions concerning the nature of the software can be made.

3. Dynamic Resources Allocation

   The processor, memory, and I/O resources of the computer cannot be pre-allocated to all of the functions to be performed during the mission. A means for providing some degree of dynamic usage of these resources must be employed.

   If the system were to be "hard-wired," the size of the computer would be prohibitive. By accepting the dynamic approach the sizing problem is reduced to consideration of the maximum needs at any one time during the mission, in this case, during the Mars orbital phase.

4. Flexibility

   In order to permit handling of unanticipated programs, and also to effectively process constantly changing program mixes, it is necessary to have an executive scheduling algorithm that can be externally controlled through program requests.

4.2.2.3.1 General

The functional design of the support programs, which are the Program Sequencer, the Reconfiguration Program, the Request Processor, the I/O Supervisor, and the Self-Test Program, was influenced by the following factors:

1. Computational requirements - The computational loading during different phases varies to such a degree that the computers' functional configuration must have flexibility. The periodic, or cyclic, nature of many computations must be maintained. The system must be capable of processing unanticipated programs.

2. Availability criteria - A positive means for detection of errors within the computer system is essential. Fault isolation of I/O errors must be performed. Upon failure, a means for smooth transition to a backup configuration must be available.

3. Programming flexibility - The software development must not be hindered by an excessive number of restrictions, since in-flight programming might be required. A positive means for program control must be incorporated.

Preliminary estimates of the cost of these support programs (detailed in 4.2.2.3.8) are less than: 3000 words and 10,000 ops/sec. Overhead in the computation programs for interfacing with the support software will be less than 1 percent.

4.2.2.3.2 Concepts of Program Design

1. Program Design Conventions

In order to permit efficient operation of the total software system, the following conventions are imposed on the design of computational programs:

a. All programs will be relocatable and will be permanently stored in mass storage.
b. External variables, which are all data used by more than one computational program and all data required for restart after reconfiguration, will be assigned fixed locations. For data used by more than one program, this provides for ease of relocating these programs particularly during reconfiguration between mission phases (if fixed locations were not assigned, moving one program could require going through other programs and recording new data location assignments). For data required for restart after reconfiguration, this facilitates inputting and outputting of such data when it is required in a number of different programs.
c. A computational program's internal variables will be segregated from its code and constants. Utility programs cannot have internal variables; the programs which call it must supply data areas and references.
d. Periodic programs will have a fixed execution time, and periods must be integer multiples of all higher frequency periods.

2. Scientific Experiments Execution

A special scheme for scheduling the execution of the scientific experiments support programs will be implemented. Attempting to

schedule them as periodic programs is too rigid in view of the following considerations:

a. A fixed frequency during a phase would be unrealistic. The rate at which data is input from a particular sensor would be alterable so that the associated data processing function can signal for an increase or a reduction in the sampling.

b. At uncertain unpredictable times the data from a sensor may be of nil value; for instance, on the dark side of a Mars orbit some TV reception might be worthless. This is also the case when a sensor malfunction is discovered.

c. In the Mars Orbit Phase the reconfiguration from the full computer system involves going to a reduced configuration. The associated reduction in the computational load is achieved by reducing the scientific experiment loading; the preferred method would be to continue most experiments with reduced data sampling rates.

In examining these programs, it can be seen that they really consist of several distinct parts (represented in Figure 4-9). The data is input from a sensor at some frequency and then stored in the mass memory. When a sufficient backlog has been accumulated the data reduction processing is performed. The end result is an output for either telemetry or console display. In order to achieve the objective of optimum efficiency along with flexibility these parts are considered as separate programs.

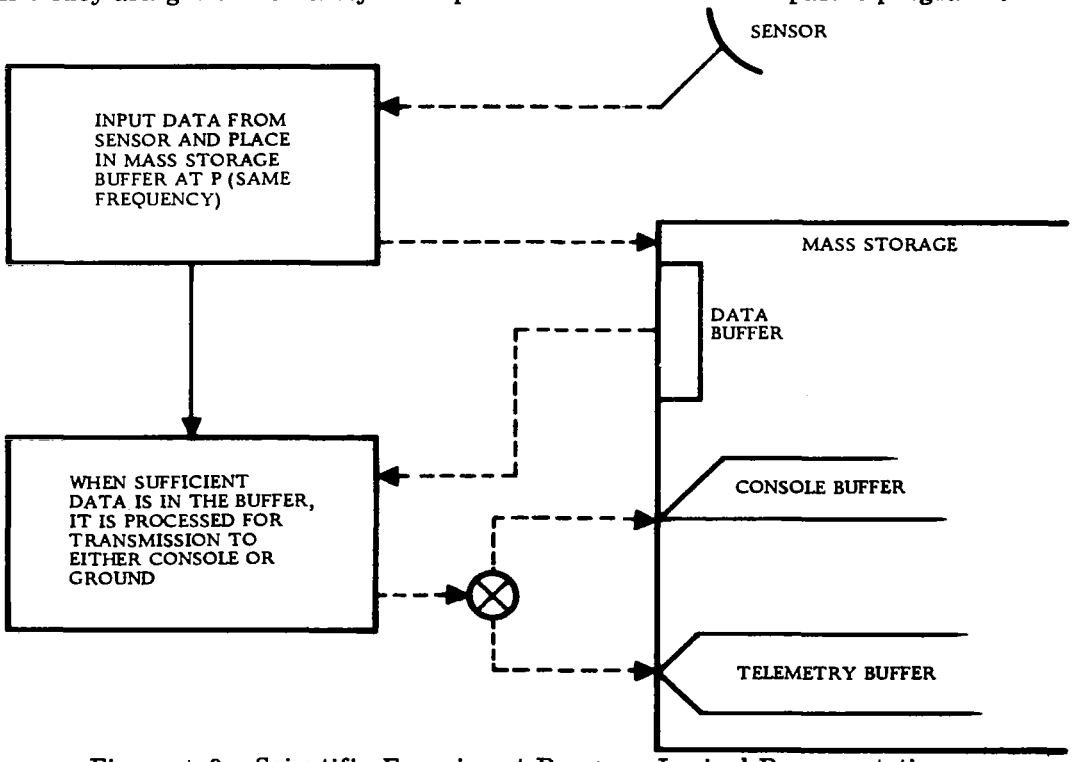


Figure 4-9. Scientific Experiment Program Logical Representation

The program for input of sensor data and buffering on the mass storage is executed at the highest potential frequency for the data sampling. The actual sample rate is controlled by setting a READ/NO READ flag under control of a frequency count. A NO READ setting would cause suppression of the buffering operation; not the sensor input operation; thus, reasonableness testing of the sensor data would be performed at a constant rate.

The data reduction program is not performed periodically, but as a request program when a sufficient backlog has been buffered. A special Data Buffer Monitor program is executed periodically to determine the status of the various buffers. When the loading level for a buffer is high enough, the appropriate flag in the Request Board (described in 4.2.2.3.5) is set. The measurement used for the i^{th} buffer is the following formula:

$$t_i = \frac{b_i - d_i}{r_i},$$

where

b_i = the size of the i^{th} buffer.

d_i = the load in the i^{th} buffer.

r_i = the loading rate for the i^{th} buffer.

thus

t_i = the time until the i^{th} buffer will overflow.

When this measure is less than a set limit, the associated data reduction program is requested. In order to prevent overflow, a priority is assigned to the program when the time-to-overflow is less than a second present limit.

Thus, when experimental data is sparse the computational support is infrequent, but when large data rates are encountered the same scheduling mechanism permits full processing capability.

4.2.2.3.3 Program Sequencer

The programs that are executed during any one phase are classified as follows:

1. Periodic - Execution is cyclic and each iteration occurring at a prescribed frequency.

2. Background - Execution is cyclic with no timing restraints.

3. Anticipated Request - Execution is on command; during execution certain subprograms may be periodic.

4. Unanticipated Request - Execution is on demand.

The basic problem is to insure that all periodic programs execute properly. The solution, which is dependent on the rules for programming periodic programs is to sequence within a fixed time-interval cycle; the length of the interval being equal to the highest frequency of the periodic programs. The periodic programs are ordered from most frequent to least frequent, (p_0, p_1, . . . , p_n).

At each cycle a counter associated with each program is decremented; when this counter, initially set to the programs' frequency, is zero the program should be executed. The programs are then executed in order of frequency. To permit time-interval cycling with low overhead, an interrupt system is employed.

As an example, Figure 4-10 shows a partial time history of the sequence of execution of the following.

Program	Frequency	Execution Time
a	1 sec.	0.25 sec.
b	2 sec.	0.25 sec.
c	4 sec.	0.5 sec.
d	4 sec.	0.25 sec.
e	8 sec.	1.0 sec.
f	8 sec.	0.25 sec.

Notice that program e could not finish execution before the end of the second interval. It was interrupted so that programs a and b could be properly sequenced and then was resumed. The shaded areas in Figure 4-10 are time gaps where no periodic computations are scheduled. This time is used for the background and request programs.

If no requests are being processed the background programs are executed in a cyclic sequence. The request programs are executed on a first-in - first-out basis with the ability to assign priorities. Figure 4-11 shows the organization of the Request Queue; Figure 4-12 lists the priorities and the queue alterations that are made to accommodate each.

The structure of entries in the Periodic Schedule, Request, and Background tables is given in Table 6-2 of Section VI. The process that occurs at each time-interval interrupt is illustrated in the executive flow diagram of Section VI, 6.3.

In order to maintain the frequency of periodic programs, "dummy" periodic programs corresponding in execution timing to the periodic computations of anticipated request programs are continuously executed. Also, within periodic programs there will be conditional program paths that may or may not be executed on any one cycle. Both of these situations create "dead time."

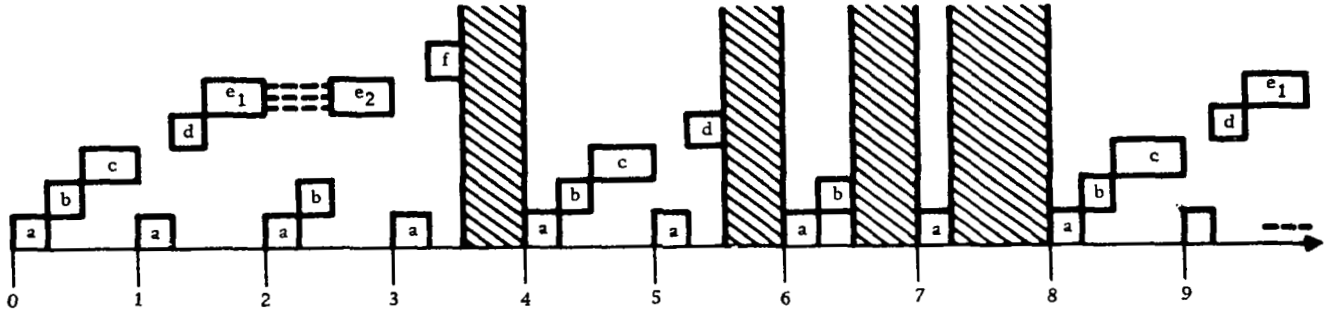


Figure 4-10. Sequence of Periodic Program Execution

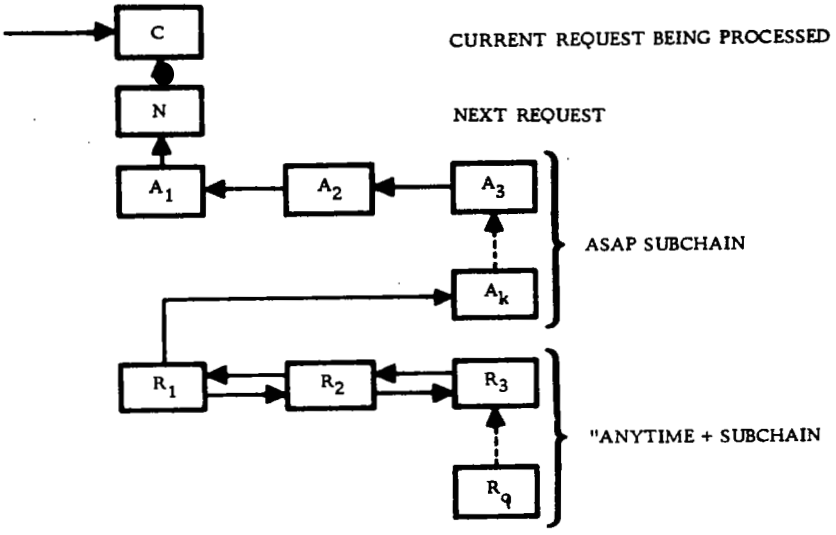


Figure 4-11. Queue Chain

PRIORITY OF X	ACTION WHEN ENCOUNTERED
NOW	$X \rightarrow C \rightarrow N \rightarrow A_1 \rightarrow \cdots \rightarrow A_k \rightarrow A_k + 1$
NEXT	$X \rightarrow N \rightarrow A_1 \rightarrow \cdots \rightarrow A_k \rightarrow A_k + 1$
ASAP	$X \rightarrow A_k + 1$
(UNSPECIFIED)	$X \rightarrow R_q + 1$

Figure 4-12. Priority Actions

Dead time can be executed as a delay, but this would lead to poor computer utilization. Therefore, a FILL function has been designed. This function is employed by setting the clock of a secondary interrupt system to the length of the dead time and transferring control to that portion of the executive which fills normal time gaps. The time-interval, or primary, interrupt system will override and reset this system whenever it is invoked.

Input operations can lead to some dead time also. This is of such short duration, however, that use of the FILL function is prohibited. Thus in most cases a simple delay must occur. For many periodic programs the first action is the input of parameters; to avoid time loss here, a special procedure is employed. This consists of executing a program specified "early I/O" code prior to performing executive housekeeping and program initialization functions.

Unanticipated request programs are sequenced as though they were anticipated, except that some computer reconfiguration is always involved. This subject is covered in the next discussion.

4.2.2.3.4 Reconfiguration Program

Reconfiguration will be required to handle mission phasing, failure recovery, and unanticipated requests in all phases. Following are the different means employed to reconfigure:

1. Phased Restart - This means is employed when a completely new program load is required; for instance, when a single computer must overlay a current phase with the next phase. This is accomplished by executing the loader as the highest priority request program and maintaining all periodic computations in the current phase until the new phases' periodic programs are loaded, along with any associated special purpose restart programs. At this time the new periodic programs begin executing and their restart programs are functioning properly, the remainder of the programs for the phase are loaded; background and anticipated request, which overlay the restart programs and any residual from the old phase. Thus a smooth transition from phase to phase is achieved.

2. Cold Start - This means is used when a computer is turned on. It operates essentially the same as a Phased Restart except that there are no "old" periodic programs to be executed.

3. Request Overlay - Whenever an unanticipated request is to be handled, the program must be loaded. This is accomplished by loading into unused core and overlaying lower priority anticipated request programs. If there is not enough room for this, a command from the console is required to either turn on the secondary computer, upgrade the priority, or cancel the request.

In order to permit these operations, the memory of the computer is organized in a particular manner as represented in Figure 4-13. Thus the load profile for each phase must group its programs according to classification. Figures 4-14 and 4-15 are general flow diagrams of these reconfiguration processes. Further details on the reconfiguration programs may be found in the executive flow diagram in Section VI, 6.3.

The loader is a program that uses load profile information residing in mass storage - the following tables are used:

1. Program Load Profile - Contains the name of the program, its location in mass storage, its size, check sum information, periodic frequency (if any), early I/O entry (if any), program entry, and pointers to the load profiles of any utility programs it uses.

2. Phase Load Profile - Contains the phase name, and pointers to the load profiles of the programs for the phase. This list of pointers is grouped so as to delineate periodic, background, and anticipated request classifications.

Since the code in the programs use register dependent addressing no address construction is required. Therefore, the loader merely fills in the appropriate Program Sequencer tables, determines a program origin, and inputs the program code from mass storage to main memory.

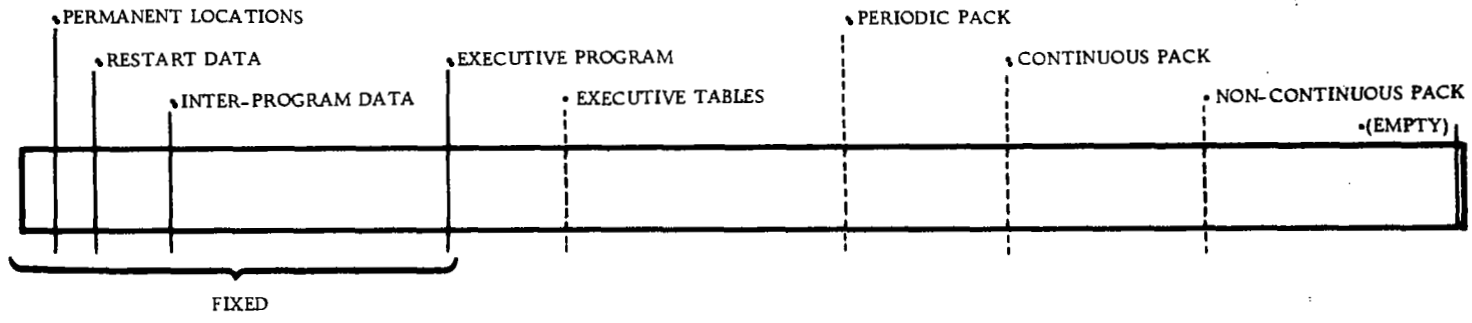


Figure 4-13. Memory Allocation

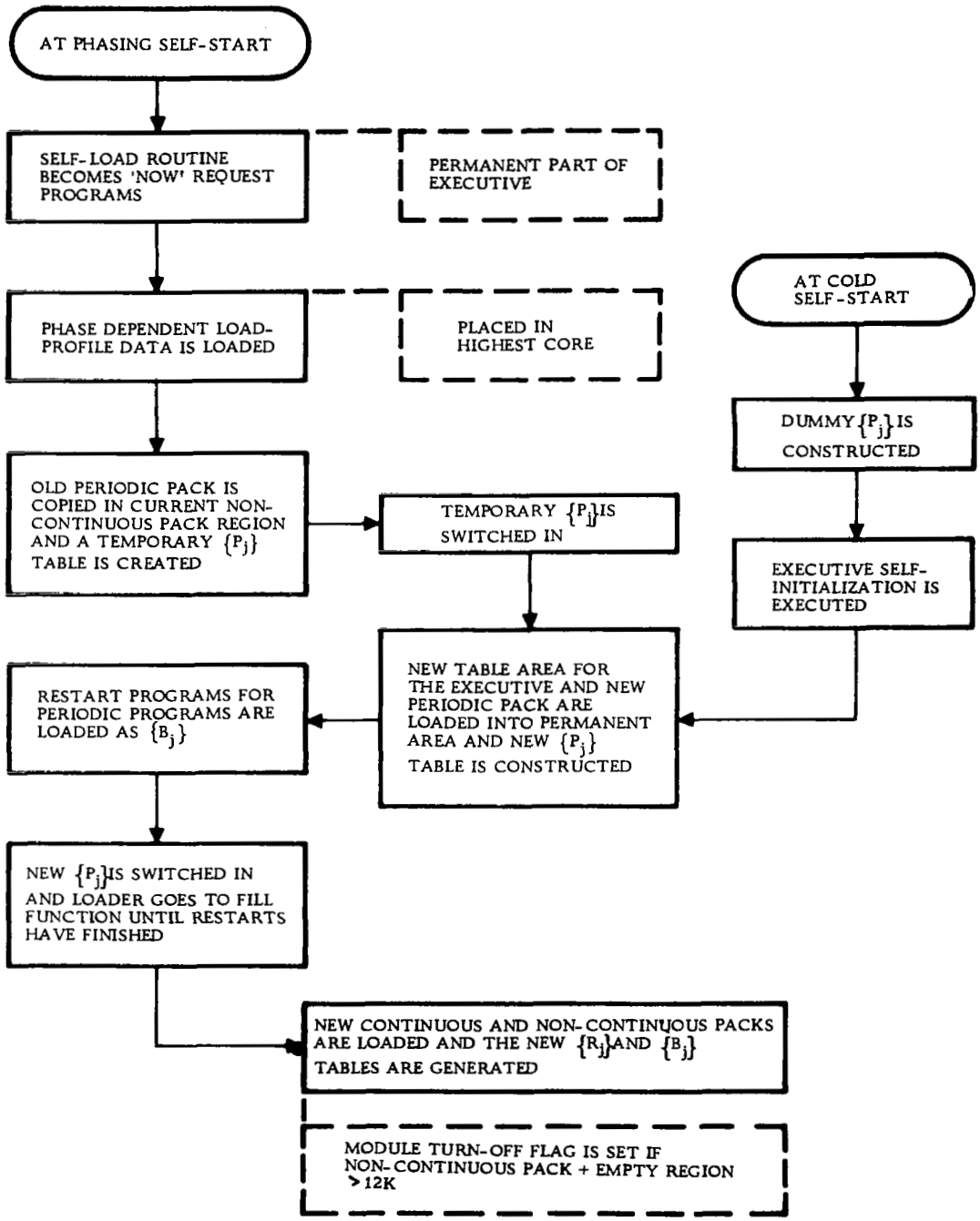


Figure 4-14. Reconfiguration Process

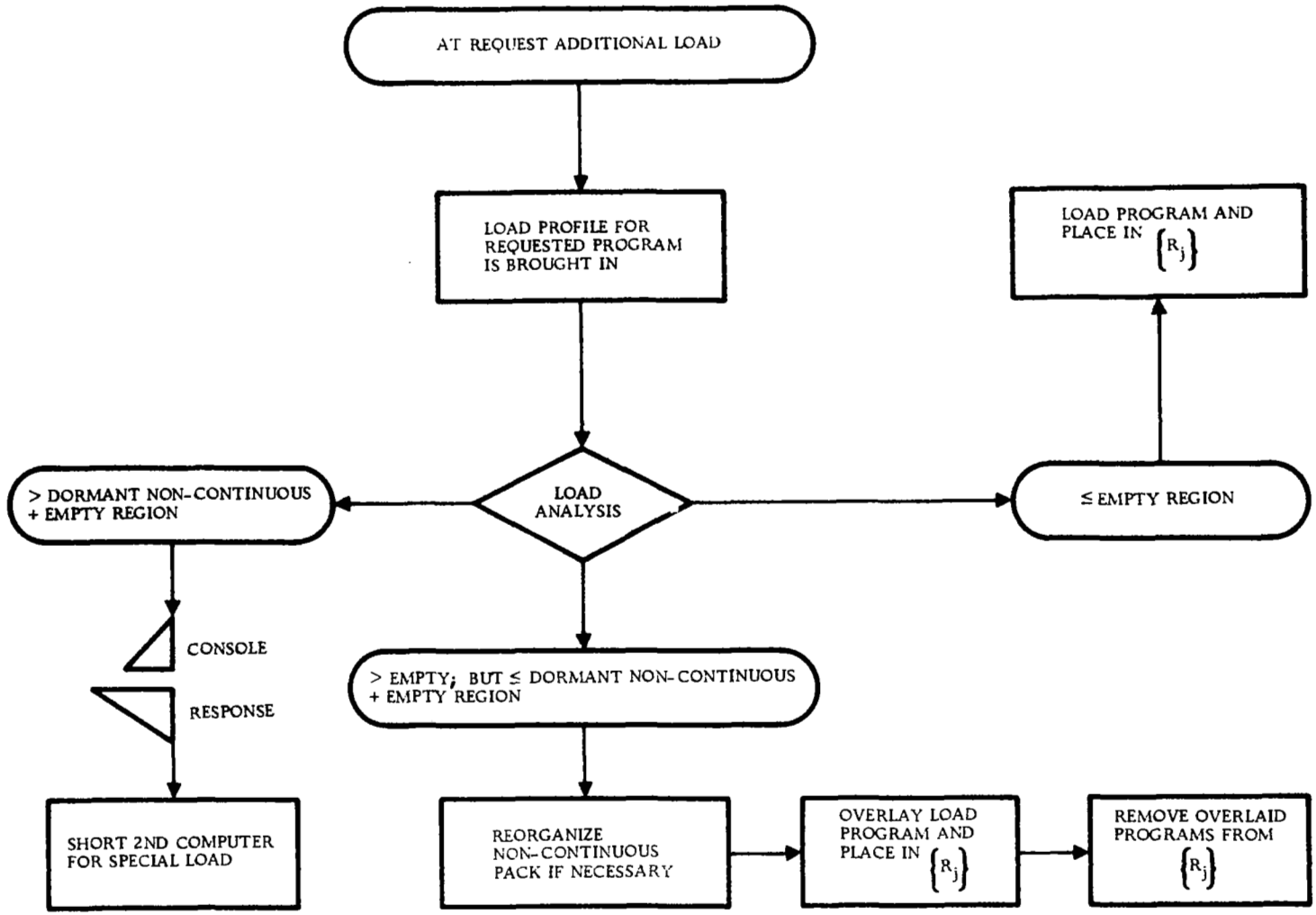


Figure 4-15. Reconfiguration Process

4.2.2.3.5 Request Processor

Requests for additional computations contained in anticipated or unanticipated request programs are made by setting a flag in a special Request Board table consisting of a string of byte flags corresponding to all possible requests; there is room in the flag to specify priority. The two primary sources for requests are the following:

1. Program generated requests - Whenever a computational program, on the basis of a test, determines that some special processing must be done, it merely sets the appropriate flag in the Request Board.

2. Console requests - A request can be generated via console message input. One of the request programs always in memory is the Console Message Processor, described below. A periodic program, the Console Interrogator, is continually checking to see if a message has been presented on the console. If so, the flag in the Request Board for the Console Message Processor is set with the highest priority.

The Console Message Processor verifies messages, checks their validity, and takes that appropriate action. On the mass storage there is a list of request program names cross-referenced to the Request Board. This list is used to determine which flags are to be set.

Before the Request Queue is processed by the Program Sequencer, another program, the Request Monitor, is executed. This program scans the Request Board and, if a flag has been set, alters the Request Queue and, when needed, initiates Request Overlay. The name of the request program is obtained from the cross-referenced list mentioned above.

4.2.2.3.6 Input/Output Supervisor

In order to provide a complete means of input and output with dynamic conditioner/sensor configurations, a standard system is provided.

The key feature is a Conditioner/Parameter Logic Table which represents the configuration of the I/O conditioners and the sensors, or parameter lines, via cross-referencing. When a sensor is connected or removed from a conditioner, or when a conditioner is added or deleted, this information must be transmitted to the I/O Supervisor (as a console message) so that this logic table can be updated.

In order to reduce overhead in exercising I/O, the following utility routines are provided:

1. GET - The computational program specifies the parameter desired, the number of inputs, the location for its storage, and reasonableness test information. This routine then uses the Conditioner/Parameter Logic Table to determine the source of the input, issues the appropriate I/O commands, performs reasonableness tests, and transmits the data to storage.

2. PUT - The computational program specifies the parameter being output, the number of outputs, and the location of the output data. This routine finds the proper output line, issues the appropriate I/O commands to transmit the data, and verifies the output.

4.2.2.3.7 Self-Test Program

A number of means are used to detect errors in the computer system. Regardless of what an actual malfunction is, the real purpose of the self-test process is to detect them and classify them as one of the following:

1. Computer failure - Any memory, processing unit, I/O unit, or critical I/O source malfunction is considered as a total computer error and a backup system must be implemented.

2. I/O failure - A malfunction in a single conditioner or a single I/O source that is performing non-critical functions causes that particular component to be blocked off. The remainder of the system continues performing in this reduced status.

There are a number of techniques used to detect errors. One is the Pulse Stream Test. A periodic program (executed at the highest frequency) alternately sets and resets a pulse flip-flop; this device is hardware monitored and, if the frequency is not maintained (within some tolerance), a computer failure signal is transmitted. Thus, logic sequencing malfunctions, closed loops, and program halts will be detected. Other self-test processes signal the discovery of errors by executing a halt command which causes a pulse failure.

Another technique is the use of check-sums. Since the programs are organized so that code and constants are blocked and check-sum data is available in the Program Sequencer tables, this check is made on a periodic basis on selected programs.

There is an Arithmetic Unit Test performed periodically to detect malfunctions in that processor's logic.

Input errors are detected on the basis of reasonableness tests, e.g., magnitude, value range, and parity. Output errors are detected by hardware feedback and a full comparison test. Conditioner errors are also detected by issuing test values through special hardware feedback loops from a periodic program; this test determines if the entire conditioner is good or bad.

At the beginning of each time-interval cycle a test is made of the Conditioner Status Words, which is where error notification is made by the tests just described. If any I/O errors have occurred, a flag is set and at the start of the next cycle a comprehensive test of all the status words is made; this is done to detect multiple errors.

If a single source error was detected, that sensor is blocked off. If multiple source errors in only one conditioner or a single periodic conditioner error was found, that conditioner is blocked off. If errors are detected in more than one conditioner, a halt is executed (indicating computer failure.

This covers the self test that is performed by software methods. Hardware methods may be used in addition to this. The hardware methods will simply require the monitoring on a periodic basis of failure notification signals set by the hardware failure detection equipment.

4.2.2.3.8 Cost of Support Software

The estimates in this section have been derived from a set of formulas involving the following parameters (the assumed value is included). The maximum phase is expected to be the Mars Orbital Phase (phase 12).

1. n_f number of periodic programs in phase

   $n_{f\,max} = 20$

2. n_r number of anticipated request programs in phase

   $n_{r\,max} = 30$

3. n_b number of background programs in phase

   $n_{b\,max} = 10$

4. n_u number of utility programs in phase

   $n_{u\,max} = 10$

5. n_c number of I/O conditioners

   $n_{c\,max} = 20$

6. n_s number of I/O sensors

   $n_{s\,max} = 100$

Another assumption is that all periodic support functions will be executed at 0.05 seconds. In the Mars Orbital Phase these estimates are for each computer. The table in Figure 4-16 gives the cost of each part of the support system.

Support Function	Storage		Speed (ops/sec.)	
	Formula	Max.		
●Program Sequencer		620	2000	
Periodic	$12n_f + 50$	(290)		
Request	$6n_r + 50$	(230)		
Background	$4n_b + 20$	(60)		
Linkage Support	$2n_u + 20$	(40)		
●Reconfiguration Program		520	*	
Phased Restart		(60)		
Cold Start		(10)		
Request Overlay		(50)		
●Request Processor		400	600	
●I/O Supervisor		610	*	
Cond./Parm. Logic	$n_c + 4n_s + 40$	(460)		
Status Check	$2n_c + 40$	(80)		
PUT, GET		(70)		
●Self-Test Program		685	6340	
Check Sum		(60)	(1500)	
Arith. Unit Test		(500)	(5000)	(At 0.1 frequency)
Pulse Stream		(5)	(40)	
I/O Tests	$n_c + 100$	(120)	(800)	
Totals		2835	8940	

Figure 4-16. Software Costs

4.2.3 Modular Multiprocessor

4.2.3 1 Organization

4.2.3.1.1 General Considerations

The computational and reliability requirements are the same as those given for the multiple computer. This of course means that the general processor features discussed earlier apply to the multiprocessor. It should also be noted here that the multiprocessor was the candidate chosen for further design; as a result a thorough discussion of its features and operation is given in chapter 6.

The multiprocessor has three 12K memory modules, two processor modules, and two I/O modules with full intercommunication between the memories and processors and between the memories and the input/output modules, these are the modules required to meet the maximum computation requirements. Again, since there is a need to continue operations while any one of the modules is in a state of repair (during critical phases), there must be at least two of each type of module. The Multiprocessor structure is shown in Figure 4-17. The ability to expand the system to four memory modules, three processor modules and three I/O modules is also included as shown by dotted lines in this figure. Further expansion would require addition of a separate multiprocessor system with an I/O link to the original system. This approach is taken due to the fact that expansion of the original system beyond 3-4-3 would require so many interconnections that the system may become impractical.

This organization offers the ability for any processor to use any memory module for either instruction or operand storage and likewise any input/output module (through communication to any memory module) for input/output operations to the desired sensors. Each module can interleave communications with any of the other modules in any sequence. It is possible for example for processor P_1 to be executing an instruction sequence from memory M_1 and receiving its operands from memory M_2 only. Essentially the modules can operate independently of and simultaneously with each other.

Communication may be restricted between modules by the use of lockout features that are incorporated in the design. These lockout features allow a processor to set a lockout in a selected memory to prevent any other processor from using that memory, the lockout feature in the input/output modules is also operated similarly. This feature allows the modular multiprocessor to operate essentially as a multicomputer during phases when critical computations are being performed in a redundant manner.

The operation of each of the modules is discussed in detail in Section VI. Briefly, each processor module functions independently of the other processor modules in a conventional manner. The processors request memory cycles according to their own internal timing, the granting of a memory cycle may take only several nanoseconds if the memory is free or up to several microseconds if other modules are granted access first. The memory modules communicate with all of the processor and I/O modules. These modules request memory cycles and are granted cycles by the scanning circuitry in each memory module. The scanner is a simple round robin type of scanner for choosing which processor or I/O module will receive the next memory cycle. It

operates as an asynchronous counter and is therefore capable of scanning all request lines in much less than 1 bit time. The I/O modules operate on instructions received via a memory module. They receive requests for I/O operations from the memories and contains scanning circuitry similiar to that in the memories to choose which memory request is honored. The I/O module also generates requests for memory cycles when it has information to transmitt to a memory. The sensors will normally be connected to the I/O through conditioners on serial links. However, the bulk storage device will be connected to each of the I/O modules by a parallel data link. The I/O module contains the capability for I/O operations with the bulk storage unit and any of the sensors simultaneously.

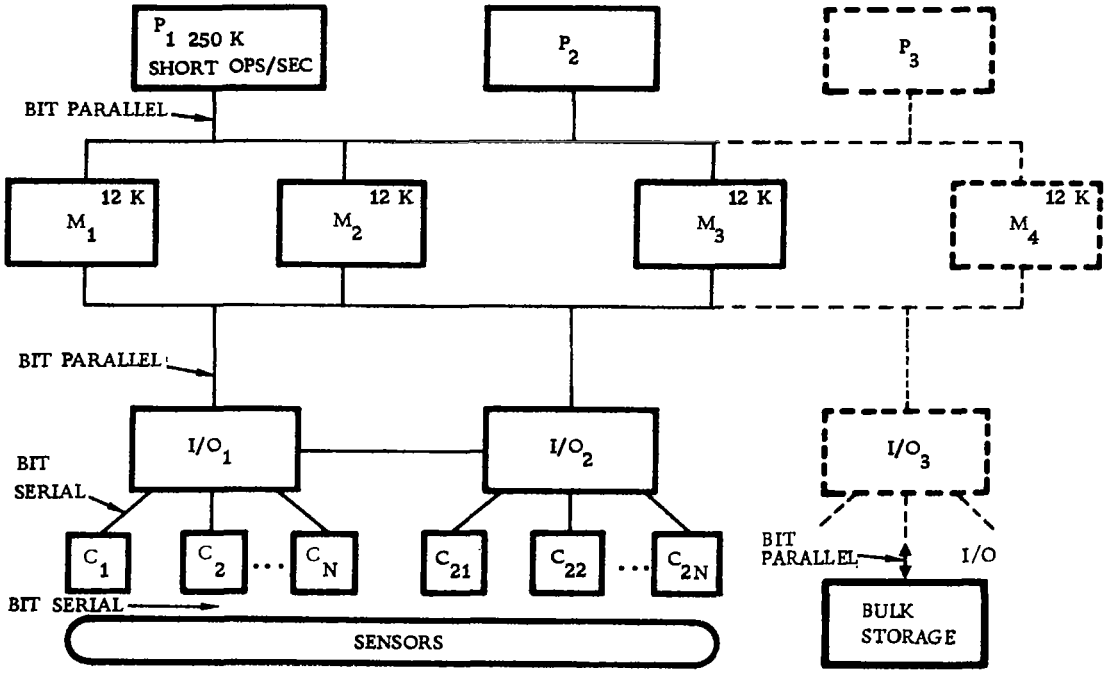


Figure 4-17. Multiprocessor

The proper use of this computer system during the various mission phases is discussed later in this section. Very simply, during Non-Mars non-critical phases, one processor, one memory, and one I/O module are in operation and the second memory comes into operation periodically. During the critical phases, the second processor and the third memory module plus the second I/O module are brought into operation. These latter modules lock out all requests from the primary processor (they are also locked out of the primary system). This means that the system operates in a manner very similar to two separate computers and, as a result, is able to guarantee reconfiguration within the five second time constraint. During Mars-Orbital operation, all modules in the system are in operation and all communication lines between the modules are allowed. Thus, full advantage can be taken of the multiprocessor structure.

The trade-offs involved in deciding the size, speed and numbers of each type of module in the system are essentially the same as those given for the two computer approach. The interesting point to notice is that the multiprocessor organization only has three 12K memory modules whereas the multiple computer has four 12K memory modules - two in each computer. The multiprocessor organization was able to save this 12K module due to its full inter-communication capability. During Non-Mars Orbital computations 20K is needed in a number of cases. This can be provided by two of the memory modules operating with one processor. The third memory module comes on and operates with the second processor during critical Non-Mars Orbital phases. In this way, all the Non-Mars Orbital memory requirements are easily satisfied. During the Mars Orbit, 30K of memory is adequate; as a result, three 12K modules will do the job in all phases. This was not possible in the multiple computer scheme since each processor did not have access to all the memory modules in the system, and, therefore, each processor had to be given two 12K modules.

The processor's speed requirements are also approximately the same as for the multiple computer case; as a result, MOS-SOS technology can again be used for the implementation of a 250,000 short operation per second processor.

The power supply for this candidate will be distributed to each board just as for the multiple computer.

The computer configurations for the multiprocessor during the various mission phases are tabulated below.

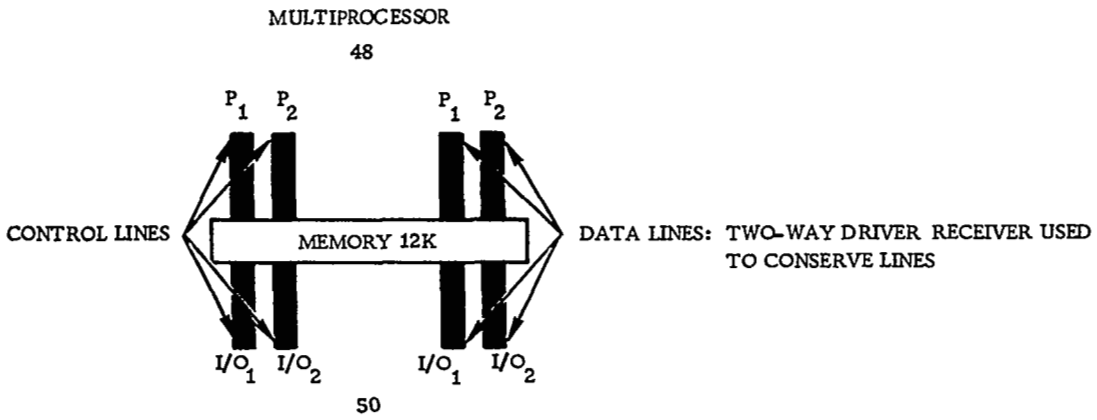
Phases 1, 2:	Processor one, I/O one, memory module one
Phases 3, 5, 6, 8, 9, 10, 11, 13, 15, 16, 18, 19, 20:	Processor one, I/O one, memory module one; processor two, I/O two, memory module two-active redundancy
Phases 4, 7, 14, 17:	Processor one, I/O one, memory one on continuously, memory three on intermittently.
Phase 12:	Processor one and two, I/O one and two, memory modules one, two and three.

4.2.3.1.2 Processor, Memory, and I/O Structure

Since the processor has all the features given earlier and also carries out 250,000 operations per second, it is basically the same as that given for the multiple computer. However, the need to communicate with three separate memories and two separate I/O units requires some extra flip-flops and gates and a fairly large number of extra lines. The extra flip-flops are the following: the program counter, and the nine index-bank registers are all increased to 16 bits. The control flip-flops are now increased to eighteen. This gives a total of approximately 330 flip-flops for this processor. This processor would also take one to two chips for its implementation in MOS/SOS circuitry. In order to get a rough feeling for the number of lines necessary for implementation of the multiprocessor an approximate line count from the memories to the processors and I/O units is presented in Figure 4-18. The specific interface lines in the count are discussed in chapter 6; however it is presented here in order to give a first cut comparison between the multiprocessor and multiple computer intercommunications. It should be noted from the figure that the multiprocessor requires almost three times as many interconnections as the multiple computer.

The use of a two-way driver/receiver on the data lines provides a good line saving; however as the number of modules in the Multiprocessor increases the number of lines necessary to provide full intercommunication becomes very large. Therefore, as mentioned earlier, intercommunication should be restricted if for any given mission the number of necessary processors gets beyond three and memories beyond four.

The memory hardware is again almost the same as that for the memory in the Multiple Computer candidate. However there are a few extra registers and interface circuitry chips along with a large number of extra lines. The primary difference is that there are three 12K modules, each one operating as a separate memory; whereas in the Multiple Computer there were two 12K modules per computer and both of these together acted like a 24K memory. This simply means that in the multiprocessor each 12K module must have not only the word and bit circuitry listed for the multiple computer 12K module but also the timing and registers listed for each 24K memory. This is not a significant increase in the hardware associated with a 12K module. The mutliprocessor memory modules must also have lockout hardware and a six bit scanner (2 extra bits provided for expandibility) to choose which processor or I/O unit receives the next memory cycle. This extra hardware amounts to approximately nine extra MOS/SOS chips. (With the development of bigger packages this could be reduced to four or five arrays.) The lockout hardware consists of a flip flop for each processor and I/O module and some gating to set these flip flops. The processors have an instruction that sends a signal to the appropriate memory in order to setup the lockout so that only this processor and a specified I/O unit can use the memory. This hardware is used to protect the memory modules from an errant processor during critical phases and also to provide undisturbed computing for periodic computations. In other words, during critical computations one processor operates with one of the memories and I/O units while the other processor operates with the other two memories and I/O unit. Each processor locks the other out of its memory (or memories) and I/O unit. This lockout enables each section of the multiprocessor to determine accurately whether it has failed or not. After a failure, reconfiguration can then be carried out easily within the 5 second maximum time. This lockout feature is also useful for periodic computations. When a processor is executing a periodic computation with a memory it does not want to be interrupted repeatedly by another processor since this would lengthen the periodic computation and possibly have an adverse effect on accuracies. Locking out the other processor during the short period while a periodic computation is in execution eliminates this possibility.



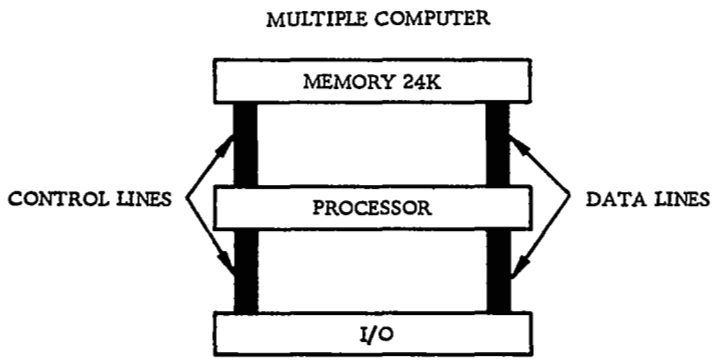
EACH MEMORY HAS APPROXIMATELY

M-P LINES 48

M-I/O LINES 50

TOTAL PER 12K 98

TOTAL PER SYSTEM 294 LINES



EACH COMPUTER HAS 49 LINES

TOTAL PER SYSTEM 98 LINES

Figure 4-18.. Approximate Line Count

The I/O structure for the multiprocessor is basically the same as that described for the multiple computer. The only difference is that the I/O modules are available to all processors through the memories. This can increase the wait for I/O variables; however the programs can be arranged to minimize queuing at the I/O units or memories. In particular during periodic computations a processor memory and I/O unit work together and lock the other processor out. The lockout hardware in the I/O is exactly the same as that in the memory and in fact is also used in the same manner during critical phases. The program scheduler using the Real Time Clock and call of I/O variables from the header of each program will be used here as explained for the multiple computer. The I/O registers are the same as those listed earlier except for the addition of three lockout flip-flops, a 3 bit memory request scanner, interface circuitry for up to four memories, a few control flip-flops, and the increase of the memory address counter to 16 bits. This gives a total of 150 flip-flops. One MOS/SOS chip will easily handle this. A diagram of the I/O unit hardware and a discussion of its operation is given in Section VI.

4.2.3.2 Failure Considerations and Reconfiguration

The introductory remarks and basic guidelines given in Paragraphs 4.2.2.2 and 4.2.2.2.1 for the multiple computer candidate apply equally well to this candidate, so they need not be repeated. Therefore the discussion will begin with error detection and isolation tests. Recall from the discussions on the multiple computer, that software approaches to failure detection are considered here. Since the multiprocessor was the selected candidate, a hardware approach will also be considered in Section VI. It should also be noted that due to considering the software approach first, many more problems associated with reconfiguration are uncovered.

4.2.3.2.1 Error Detection and Isolation Tests

The following paragraphs describe the tests required to insure timely indications of the multiprocessor status during the mission.

1. Processor—Memory Tests

The memory check sum, arithmetic section functional test, and program control test, as described for the multiple computer system, would also be the primary tests for detection of errors in memory or processor modules of the multiprocessor configuration. The nature of these tests need not be described here again.

In the multiprocessor configuration the requirement has been established to isolate errors to the processor or memory. The above tests provide this capability only to a limited extent. For example, a processor arithmetic error can be isolated to the processor by executing the arithmetic functional test twice, once from each of two memories. Normally the test would be executed the second time only upon failure of the first test. Similarly, a memory failure is isolable by a check sum where the memory is an operand source, not an instruction source, for two processors. Where a memory is an instruction source at the time of its failure the program control test will detect the error, as it will if the processor contains a control error.

The approach chosen to isolate errors between memories and processors (and between processors and I/O units too) generally takes advantage of the fact that isolation need not be instantaneous and that the space crew is available to perform procedures as required for isolation subsequent to error detection. The penalty of this

approach is that more equipment than otherwise necessary may be placed in a "down" condition at the time an error is detected and, of course, also that more crew participation is required. However, an analysis of the mission success and availability requirements shows that those requirements can be more than adequately met with this approach. The alternative is to add redundant hardware such as memory parity checking and arithmetic coding schemes.

In the section on reconfiguration the procedures for isolation of processor and memory failures will be discussed in some detail.

2. Input, Output Signal Tests

The basic nature of these tests for the multiprocessor is similar to that described for the multiple computer system. That is, reasonableness tests and BITE circuitry can be used to check input signals, and output signals are checked by looping them back into the computer. The results of the checks are recorded in I/O conditioner status words and these in turn are checked by the full cycle fault isolation routine to isolate the failure.

The new problem that arises in the multiprocessor is the ability to isolate errors between processors and I/O units. Generally, certain processor failures, and I/O unit failures, will result in the same conditioner status words. The isolation ambiguity is resolved by taking advantage of the built-in flexible communication paths between each of the processors and each of the I/O units. Thus, one processor can attempt to talk to two I/O units, or two processors can attempt to talk to the same I/O unit. The implementation of this test is dependent on the multiprocessor configuration at the time of the failure. As such it will be further discussed in the section dealing with reconfiguration.

4.2.3.2.2 External Status Reporting

The status of the on-line units of the system is continually reported by means of two sets of control panel indicators. Each set is controlled by a processor. In case of a failure the indicators either flag the failed module, or flag a trouble area which forms the basis for the initiation of further tests to isolate the failure. The isolable modules are memories, processors, I/O units, conditioners, and input devices.

Each set of indicators consists of the following:

computer fail light

processor - memory lamp

processor - I/O unit lamp

conditioner lamp

input device lamp

numeric readout

The computer fail light is controlled by the BITE circuitry for the program control test of the associated processor. Recall that the BITE circuitry is installed here because the computer cannot be relied on to actively report its own status. Errors reported by this lamp implicate either the associated processor or a memory. All other indicators represent the situation where the processor has been able to actively make a decision. These other indicators are meaningful only if the computer fail light is off. The indicators are tied to discrete outputs from the processor and are set by the program when certain errors are detected. As implied by their names, the fault may be in the processor-memory area, processor-I/O unit area, or more definitely, a conditioner or an input device. The numeric readout is associated with either of the 4 lamps that are lit, further specifying the failed unit or most likely failed unit, in a prescribed coded form.

Normally, failures detected by the program control test, arithmetic functional test, and memory check sum will be reported by either the computer fail light, or the processor-memory lamp and numeric readout. Failures involving input or output signals are reported by either the processor-I/O unit lamp, the conditioner lamp, or the input device lamp. The specific conditioner or input device is reported on the numeric readout.

Further details on how this failure notification system is used is given in the succeeding sections on reconfiguration and backup equipment assurance.

4. 2. 3. 2. 3 Reconfiguration

This section discusses the task of reconfiguring the multiprocessor after a failure has been detected and reported to the space crew. It will be shown that, as compared to the multiple computer system, the multiprocessor affords an inherently higher probability of mission success and higher system availability. The former is achieved by being able to withstand selected multiple failures during critical phases. The latter is achieved mainly by eliminating the need to include module replacement time as a part of reconfiguration time during non-critical phases for certain failures.

As in the case of the multiple computer system the reconfiguration plan is based on the type of phase in which the failure occurred; either non-critical, critical, or Mars orbital.

Figure 4-19 represents the general multiprocessor configuration. The nomenclature given therein will be used throughout the discussion.

1. Non-Critical Phases

During non-critical phases the primary system consists of M1, P1, I/O1, $C1_1$, . . . , $C1_N$ and M3 as required for the performance of non-continuous functions. The secondary system consists of M2, P2, I/O2, $C2_1$, . . . , $C2_M$. The primary system is performing the required mission functions; the secondary system is normally turned off except for periodic intervals when it is turned on and checked. Generally, the configuration of the secondary system is based on the anticipation of a failure in the primary system, and its subsequent role as a checkout device and as a source of verified modules that will become a part of the primary system during reconfiguration.

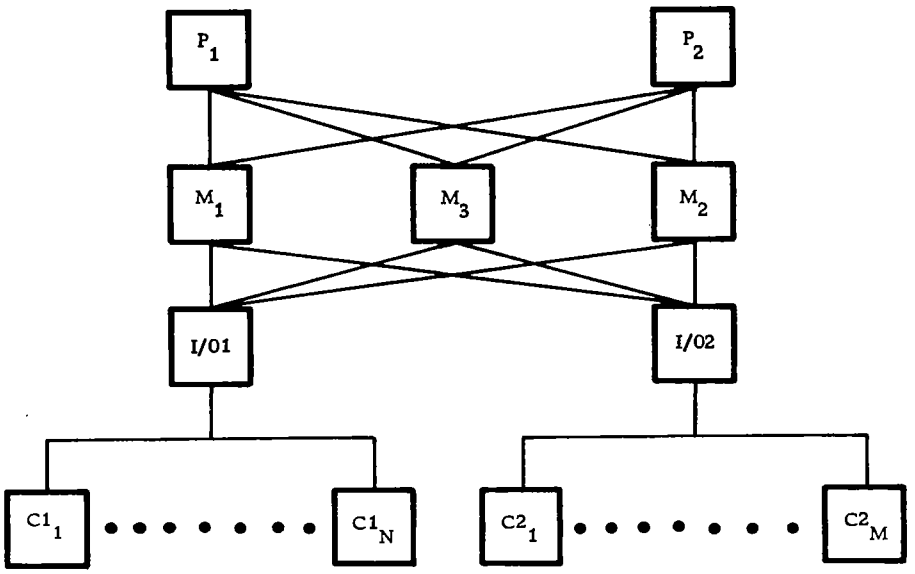


Figure 4-19. General Multiprocessor Configuration

The reconfiguration process starts with a failure notification by the primary system. As stated in the previous section, failure notification consists of a computer fail light controlled by BITE circuitry, processor-memory, processor-I/O, conditioner, and input device fail lamps controlled by discrete outputs from the processor, and a readout specifying the most likely unit containing the error.

There are two reconfiguration procedures; one if the error is in either a memory, processor, or I/O unit, and the other if the error is in a conditioner or input device.

The procedure for memory, processor, or I/O unit failures uses the secondary system as a checkout device to test the primary system and isolate the failure. Where the primary system has isolated the error the procedure provides corroboration. More often however, the primary system will be unable to isolate its memory-processor-I/O unit failures. The procedure is as follows: First, the power to the primary processor P1 is turned off. The secondary system is turned on and checks itself with the same tests it has been periodically performing when there was no error in the system. (The tests are described in the section on backup assurance.) Next it starts checking the primary system. During these tests all instructions to be executed by P2 are read out of M2. M1 is checked by loading it with selected bit patterns and verify the loading. Similarly for M3. In the worst case, all locations of M1 and M3 are accessed. An M1 or M3 failure is isolated by these tests and reported via the control panel readout reserved for P2's reports. This report would confirm the primary system's failure report wherein either the computer fail light came on or the P-M light came on with the readout specifying either M1 or M3 as the likely error source. If both M1 and M3 are found to be correct, checkout automatically continues

with P2-M2 checking I/O1. The nature of the error which would cause I/O1 to be suspect probably requires the performance of only a select few input/output operations. Typically the BITE inputs from the conditioners would be accessed and several outputs would be tried. This test removes the ambiguity of errors reported by the primary system processor-I/O lamp, i. e., if the test fails, I/O1 is the failure source; if the test passes, P1 is the failure source.

In the case of failures in either conditioners or input devices, the primary system isolates the failure (by means of the conditioner status words and full cycle fault isolation routine) and reports the failed unit by means of the conditioner or input device fail lights and the readout. In this case the secondary system is not used for fault isolation although it will probably be turned on to check the replacement prior to its insertion in the primary system. As a precautionary measure, the astronaut may request certain prescribed readouts from the primary system to insure that the conditioner involved in sending the failure readout is correct and has not, through its own failure, implicated another conditioner.

Having isolated the failure by the above procedures, the next step is to reconfigure the system around the failed unit. Because of the flexible communications several possibilities exist for processor-memory-I/O unit failures. Since the secondary system is active and functioning in cases of those failures, the present plan calls for using the P2-M2 combination in the reconfigured primary system. I/O2 is used only if I/O1 has failed, and then it must be connected to the conditioners associated with I/O1. Conditioner or input device failures also require physical replacement to effect reconfiguration. Thus, for processor or memory failures, reconfiguration is effected without physically replacing the failed module, while for other failures physical replacement is required. Replacement of processor or memory modules is performed off-line, after reconfiguration has been accomplished. This represents an improvement in system availability over the multiple computer system. Table 4-3 summarizes the reconfiguration procedure.

Table 4-3. Non-Critical Phase Reconfiguration Summary

Primary System Prior to Failure	Failed Unit	Failure Detected By	Failure Isolated By	Reconfigured Primary System
M1,M3,P1,I/O1, $C1_1, \ldots, C1_N$	M1	Primary	Secondary	M2,P2,M3,I/O1,$C1_1, \ldots, C1_N$
	M3	Primary	Secondary	M2,P2,M1,I/O1,$C1_1, \ldots, C1_N$
	P1	Primary	Secondary	M2,P2,M3,I/O1,$C1_1, \ldots, C1_N$
	I/O1	Primary	Secondary	M2,P2,M3,I/O2,$C1_1, \ldots, C1_N$
	$C1_j$	Primary	Primary	M2,P2,M3,I/O1,$C1_1, \ldots C2_k, \ldots C1_N$

2. Critical Phases

The reconfiguration process for the multiprocessor system is similar in its basic concept to that previously defined for the multiple computer system. However, because of the flexibility of inter-module communications and the ability to isolate failures to the lower level with confidence, the multi-processor can withstand certain multiple failures during critical phases. This will result in a higher probability of mission success for this candidate.

Referring to Figure 4-19, the primary system, consisting of M1, M3, P1, I/O1, $C1_1, \ldots, C1_N$, and associated input/output devices is performing the mission functions, both critical and non-critical. The secondary system, consisting of M2, P2, I/O2, $C2_1, \ldots, C2_M$ and associated input/output devices, is in an active standby redundant mode. Failures in the primary system are detected by the primary system, and where the failure involves units performing critical functions, an automatic switchover to the secondary system for control of the vehicle is initiated. This constitutes reconfiguration and is accomplished in much less than the allowable 5 seconds. Noncritical failures in the primary system result in a suspension of associated computations, not in an automatic switchover to the secondary system. Failures in the secondary system result only in a failure notification. No reconfiguration is necessary to sustain the critical functions.

Now then, after the first critical failure the operating system can be informed of the failed system's status as reported on the failed system's failure notification lights. The operating system can then perform isolation checks on the failed system, if required, as part of its background calculations and report the total system status. (The memory storage requirements for the isolation programs should be readily available considering the "relatively low" storage requirements for the critical navigation and guidance function.) If the failed unit were a processor, the operating system can bypass subsequent failures in I/O units, conditioners, and input devices by reassigning tasks. If the failure was in a memory it is conceivable that the primary and active redundant configuration can be reinstituted in anticipation of the next failure. For an I/O unit failure, subsequent failures in a memory or processor may be tolerable. Similar possibilities exist for a critical conditioner failure.

One additional consideration is worthy of mention in the critical phase configuration of the multiprocessor; that is the possibility of a single failure bringing down the entire system. The possibility of such an effect is considered negligible because of the memory lockout feature incorporated in the design. Basically, prior to the inception of the critical phase, P1 is locked out of M2, and P2 is locked out of M1 and M3. The lockout feature is described in more detail in Paragraph 4. 2. 3. 1.

As has been stated, reconfiguration during this phase is automatic, being accomplished well within the allowable 5 seconds. Failed units are repaired by replacement during non-critical phases in the manner previously described.

3. Mars Orbital Phase

During the Mars orbital phase the entire multiprocessor system is on; taking fullest advantage of the configuration in the performance of the required computational tasks.

During this phase there are no critical computations. However, to shorten reconfiguration time in the event of a loss of the navigation and guidance function, a minimum navigation function is performed in a standby redundant mode.

A basic conflict exists in the design of the reconfiguration system for this phase. On the one hand it is undesirable to restrict the inherent flexibility of the communications between processor and memory modules, while on the other hand if such restrictions do not exist there is the possibility that failures in these units will result in simultaneous processor failure notifications, implying that a known starting point for subsequent failure isolation does not exist. Several examples of this effect will now be presented.

First, suppose a memory fails, say M1. If at the time of the failure M1 is acting as a source of instructions for one processor, say P1, then P1 is likely to fail its program control test. While P1's BITE timer is marking time to recognize the error, P1 could conceivably write into a good memory, either M3 or M2, and contaminate it. Or, P1 could lock P2 out of M3 or M2. Then P2 may subsequently indicate a failure while attempting to operate with M3 or M2. In the computer's time frame these two failures might be far apart, i. e., many computations may occur between the first and second failure indication. However, in the time frame of the astronaut, the failures may appear to occur simultaneously. The result is that the reconfiguration procedure does not have a known good starting point and requires a certain amount of trial and error.

As another example, if at the time of M1's failure it is acting as an instruction source, not only for P1, but also for P2, both processors will fail their program control tests and simultaneous failure notifications will result.

Similar to these previous examples, if P1 were to fail with a control type error it would be detected by its program control test. While the program control timer is marking time prior to signalling the error, it is difficult to predict P1's actions. It is possible that it can contaminate the system.

Thus, as opposed to the non-critical and critical phases, the reconfiguration procedure for the Mars orbital phase can involve trial and error, since a known good starting point for reconfiguration may not exist.

Next, what reconfiguration may entail will be examined if the known good starting point is not available. Remember, this situation arises only for certain cases of memory or processor failure.

First, the entire system is shut down; all computations are suspended. Next, using one processor-memory pair, say P1-M1, a checkout program is loaded into M1 from bulk storage. This program would be similar to that used for checkout in non-critical phases. Then, P1-M1 proceeds to check itself. If P1 or M1 contains a failure, the computer fail lamp will be lit. In this case, the error is in P1 or M1 and the procedure is restarted, this time however, loading the checkout program into the P2-M2 pair. The P2-M2 system will pass its self-check (assuming single errors). If P1 or M1 does not contain a failure, it can be used as a checkout device for M3, M2, and P2, similar to operation in non-critical phases. That is, the rest of the system, except P2, is turned on, and checked starting with M3 then M2. If both M3 and M2 are correct, the error is assumed to be in P2.

Once the error is isolated to the processor or memory, the operational program is reloaded, and computations resume at a reduced level, depending on whether the failure was in a processor or memory. The navigation and guidance function must start over, from an initialization routine, since the previous values have been lost. It is estimated that it will take at most 1/2 hour to compute accurate data. Concurrently, the failed module is replaced with a spare, if the spare is available, and the full mission functions are eventually resumed. If the spare is not available, the mission functions are reassigned, some being suspended or reduced.

Reconfiguration time for this type of failure is mainly a function of the time to resume the navigation and guidance function and the time to remove and replace the failed unit. These actions are not sequential, but rather overlap one another. Astronaut participation does not appear to be excessive. He is required to control power to the units, to call for the loading of processor-memory pairs, to interpret the failure notification lamps, and finally to remove and replace the failed module. It is assumed that reconfiguration time here is of the order of magnitude of 30 minutes.

Next, the types of communication restrictions that might be applied to reduce the necessity of the above procedure will be discussed. Essentially the restrictions tend to assure having a known good memory-processor pair at the start. Further, referring to Figure 4-19, they would be applied such that if P1 reports a failure, M2 and P2 are known to be good, and if P2 reports a failure, M1 and P1 are known to be good. The good memory-processor pair is then used to isolate the failed module.

The first feature is intended to make it difficult for a processor to write into a memory, thereby tending to reduce the possibility of system contamination by a failed processor or failed memory that affects a processor. This is done by requiring the processor to execute a specific sequence of commands to enable its writing into any one of the memories. These special commands, which in the simplest case would consist of one Enable Write command containing a particular memory number, would be associated with hardware in the write control circuitry of the processor. Thus when a STORE command into a memory is executed, a write control would be issued only if writing into that memory were enabled. In line with the basic approach to the problem, this feature does not impose much of a constraint on the P1-M1 pair or on the P2-M2 pair since the enable write sequence need only be executed once at the start. For P1-M2, P1-M3, P2-M1, P2-M3, each time the writing was enabled, and after writing occurred, it would be followed by a Disable Write command, such that future writing again requires execution of the write enable sequence. Note that this technique is extremely flexible since it is under stored program control.

The next feature is a programming constraint, and involves no additional hardware. It requires that both processors should not be simultaneously executing instructions from the same memory. This feature guards against the possibility of a failure in one memory causing program control failures in both processors. From the programming point of view this can be accomplished in several ways. By one method, all instructions executed by a particular processor would be relocated to a particular memory prior to execution. By a second method, prior to using a particular memory as an instruction source the processor would first seek permission of the other processor. This can be done by having each processor store a particular pattern in a known memory location when it is using the memory as an instruction source, and erasing that pattern when it is finished with the memory. Alternately the processor using the memory can lock out the other processor. A combination of these methods is the likely solution, with emphasis on those requiring the least computing power to perform the function.

A third and final feature represents another programming constraint. Given that P1-M1 or P2-M2 will be the eventual base upon which reconfiguration and error isolation procedures are built, one would tend to restrict the number of write requests from P1 to M2 and from P2 to M1. Basically the reason for this is that failures occur randomly and therefore would occur as P1 is writing into M2 or as P2 is writing into M1. Further, when a processor appears to be berserk, either due to its own error or a memory error, there is less of a chance of accidentally executing a write enable sequence and/or subsequent instructions causing writing.

The extent to which these three features are used should be determined by further study. It appears that the first and third features are most likely, and the second feature less likely or toned down.

Up to here only memory-processor failures have been considered. Failures in the input-output area in the Mars orbital phase will now be discussed.

Failures involving an ambiguity of either a processor or I/O unit can be resolved by the full cycle fault isolation routine if the processor is normally communicating with I/O devices through both I/O units. Then, as is the case for conditioner failures, the group of apparent failed input and/or output signals, can be associated with either a particular I/O unit, or, if all appear to be bad, with the processor. Actually, where both processors have equal facility in communicating through both I/O devices, I/O unit failures will result in similar failure reports by both processors and no further isolation activity is needed. Similarly a failure report from only one processor would implicate the processor.

If each processor is not normally communicating with one of the I/O units, it can attempt to perform selected input and output operations through that channel to resolve the ambiguity. The result can be substantiated by requesting the other processor to perform a similar operation, although this may not be necessary.

In the event of this type of a failure, be it the processor or I/O unit, tasks would be reassigned. The navigation and guidance function is preserved, either at full strength or in a minimal manner depending on the failed unit. The full computational capability is restored only after the failed unit has been replaced.

Finally, conditioner or input device failures are isolated by the full cycle fault isolation routine. Communications involving the failed unit are suspended until a replacement is inserted into the system.

4.2.3.2.4 Backup Equipment Assurance

In order to assure the ability of backup equipment to take over its role in the system as required, it will be periodically tested.

Referring to Figure 4-19, the backup system during non-critical phases consists of M2, P2, I/O2, $C2_1, \ldots, C2_M$. Normally its configuration consists of at least that equipment to perform the navigation and guidance function during critical phases in the event of primary system failure, or to assume a full computational load during the Mars orbital phase.

The tests on the backup equipment are initially disjoint from the primary equipment. That is, prior to the start of testing the primary processor P1 locks the

backup out of M1, M3, and I/O1. The backup tests itself by performing navigation and guidance computations, memory check sums, and arithmetic functional tests. These tests would be similar to that performed by the backup computer in the multiple computer system. At the conclusion of this self-check, the backup reports successful completion via its readout. The astronaut then tells the primary system to initiate interface checks. P2 is allowed to test its ability to communicate with M1, M3, and I/O1, and P1 tests its ability to communicate with M2 and I/O2. These tests are expected to be short and simple since only the interface circuitry is being checked. Typically, for example, the processor to memory test might assure the ability to transfer ones and zeros on the data lines for both read and write requests, and the ability to address several selected memory locations. The results of the interface tests are reported via the respective processor readouts. If no error is detected, the secondary system is returned to the idle state. If an error is detected it would be assumed that the backup unit involved is incorrect so as not to suspend primary system functions. The suspected backup unit is replaced and the test is repeated. If it fails again, the primary system's unit is replaced. This concludes the procedure for backup equipment assurance during non-critical phases.

During critical phases, the backup system is on-line performing at least the navigation and guidance function in active standby redundancy. The lockout feature is used to separate processor, memory, and I/O units associated with primary and secondary systems. This serves to assure that single failures will not bring the entire system down (as can occur during the Mars orbital phase). The backup system tests itself while performing its operational function. Where errors are detected, the primary system can be used to isolate the failed unit on request from the astronaut. As in the discussion presented for primary system reconfiguration, this testing would allow the ability to continue operation in the event of additional failures during the critical phase.

During the Mars orbital phase, the term backup equipment does not really apply, although there is a minimum navigation and guidance backup function. Thus in this phase checks are done as part of the operational program.

4. 2. 3. 3 Software Considerations

4. 2. 3. 3. 1 General

The functional design of the support programs for this configuration is essentially the same as that for the multi-computer described in Paragraph 4. 2. 2. 3. Therefore, this section will not describe the full design but will cover the differences between the two.

The primary factor that creates differences in the software design is the functional configuration during the Mars Orbital Phase when all modules are interconnected. This differs from the previous design where the two computers were both functioning but were independent from each other.

The costs for the support programs (detailed in Paragraph 4. 2. 3. 3. 8) are less than: 3000 words and 10000 cps/sec. Overhead in the computational programs will be between 2 - 6% in time and storage.

Section VI contains a detailed design of the overall executive and reference should be made there for further information on the concepts introduced in this chapter.

4. 2. 3. 3. 2 Concepts of Program Design

The same conventions apply for this design, and scientific experiments are executed as before (see 4. 2. 2. 3. 2).

4. 2. 3. 3. 3 Program Sequencer

The same scheme of processing periodic programs on a time-interval interrupt schedule and filling in with priority-ordered request and background computations is used (see Section 4. 2. 2. 3. 3). However, during the Mars Orbital Phase a few modifications are necessary:

1. The periodic programs are grouped into two packages, $\bar{p}_1$ and P_2, with separate schedules. Two processor-module groups P_1 - M_1 and P_2 - M_2, are assigned to process these packages. When P_1 is operating on $\bar{p}_1$ in M_1, a logic block is set to prevent P_2 from executing in M_1 in order to insure proper timing through $\bar{p}_1$: the same is true for P_2 - M_2.

2. The request queue is in M_3 and is accessed by both processors when they are free to do so. Therefore, some additional logic is required to avoid interference.

   When a processor begins scanning the request queue, it sets a bypass to prevent the other from scanning at the same time. This should involve a delay of only a few machine instructions and is reset when it is safe to do so.

   A flag is set in the request queue entry for the NOW request program when a processor begins executing it. Thus, when the other processor examines this entry it will know not to duplicate and will pick up the next entry for execution.

3. The background programs are also divided into two groups, one for each processor-module group, so that when no requests are being filled dual non-interferring processing can be done.

4. 2. 3. 3. 4 Reconfiguration Program

During the critical and non-critical phases, reconfiguration to handle mission phasing failure recovery and unanticipated requests is accomplished the same as before with one additional task involved. The backup computations are loaded for P_2 - M_2, and P_1 - M_1 - M_3 - I/O_1 is mutually blocked off from P_2 - M_2 - I/O_2: this involves the setting of logic flags.

In phasing to Mars Orbital from phase II, the P_2 - M_2 - I/O_2 logic blocking must be removed while the periodic computations are picked up. The means for doing this is presented in the executive flow diagram in section 6. 3.

A special backup load profile is available for the Mars Orbital Phase which has only one periodic package and fewer computational programs (reduced scientific experiments and communications). This is used if one processor and/or two memories fail, since in these cases only one processor-module group can be utilized.

If M_3 during this phase contains only request programs, the failure of one memory module will allow reconfiguration to the original load profile. The only loss will be in the additional time involved in handling the former M_3 programs as unanticipated programs.

4.2.3.3.5 Request Processor

(See 4.2.2.3.5)

4.2.3.3.6 I/O Supervisor

(See 4.2.2.3.6)

4.2.3.3.7 Self-Test Program

Although there is a considerable increase in this area, primarily in fault isolation, the impact is on the makeup of the backup configurations and not the primary system.

A new area is the failure notification process. The tests performed are the same as before (see 4.2.2.3.7).

4.2.3.3.8 Cost of Support Software

There is some increase in cost over the multi-computer configuration's design (see 4.2.2.3.8). The primary one being the overhead caused in the computational programs due to using an ENABLE/DISABLE accessing scheme.

The support costs that increase are:

1. Request scheduling	20 words	170 ops/sec.
2. Linkage support	6 words	40 ops/sec.
3. Reconfiguration	120 words	*

These additions (146 words and 210 ops/sec.) make a grand total of 2981 words and 9150 ops/sec.

It should be mentioned here that, since the multiprocessor candidate was selected for further investigation, a detailed design of the overall executive may be found in Section VI.

4.3 DISTRIBUTED PROCESSOR

This section covers the preliminary design of the distributed processor candidate. A description of an analysis of parallelism within computations shall be given prior to the organizational description, fault detection, and software considerations.

4.3.1 Parallelism

Two types of parallelism were considered: Natural Parallelism and Applied Parallelism. These two types were defined as follows.

1. Natural Parallelism - The property of having the capability for carrying out a number of groups of operations on distinct data bases or on the same data base simultaneously and independently.

2. Applied Parallelism - A number of groups of exactly the same operations on distinct data bases or on the same data base simultaneously.

Basically the nature of most of the distributed logic organization machines conceived of to date may be broken down to one of these two classes or a combination of the two. The Solomon type of distributed processor is primarily an "Applied Parallel" machine while the Holland type may be considered a "Natural Parallel" machine. References 20-25 cover a fairly broad spectrum in the type of distributed machines designed this far. The Holland type distributed processors (local control) can handle natural parallelism very easily; however, although applied parallelism is also handled just like natural parallelism, the price of local control seems high when applied to a large number of "Applied Parallel" problems. The Solomon type distributed processor (global control) is designed to easily handle "Applied Parallel" problems, but it is not well suited to the handling of "Natural Parallel" problems. In fact, to enable Solomon type computers to handle even two "Natural Parallel" computations, the ability to interleave control signals would need to be instituted.

This distinction in parallelism led to investigating the computations to determine the effectiveness of the two types of parallelism in a distributed machine with application to the manned Mars mission. The computation task for the manned Mars mission as defined by the requirements in paragraph 2.8 was investigated and the two types of parallelism were considered for the indiviudal tasks. A description of how this was accomplished is given below. This section is concerned with determining the effectiveness of parallelism. How it is actually implemented will be presented in paragraph 4.3.2.

The simple example given in Figures 4-20 to 4-22 is the computation of $a/x + b/x + cy = z$. Figure 4-20 illustrates the sequential steps of computation on a single computer (S), while the numbers above each circle indicate the time required to compute the term in the circle. It may be noticed that the a/x and b/x term fit the definition of applied parallelism, therefore, they are computed as in Figure 4-21 using applied parallelism. The term A/S in the drawing is the ratio of time required on the Applied machine to that required on the Single machine for the total computation. As shown in this example, it takes 2/3 as long on the applied machine for the computation to be performed, also the degree of applied parallelism required in this computation is defined as 2 and occurs only during the portion of the computation where a/x and b/x are computed.

It may also be seen that the term c/y need not necessarily be computed after a/x and b/x in Figure 4-21, that is the capability for computing cy in parallel with a/x and b/x would lead one to the computation flow of Figure 4-22. This capability exists if natural parallelism is available. However, to say that this is now a combination of applied and natural parallelism is rather meaningless according to the definitions, since Figure 4-22 could be implemented by natural parallelism alone. That is, three groups of operations could exist simultaneously, two of which are the same mathematically. However, this type of an approach can lead to serious inefficiencies when one considers the mechanization of such natural parallelism on a

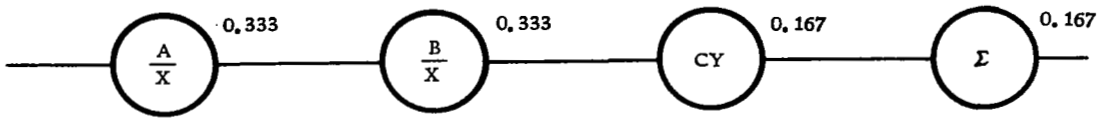


Figure 4-20. Sequential Steps in Computation

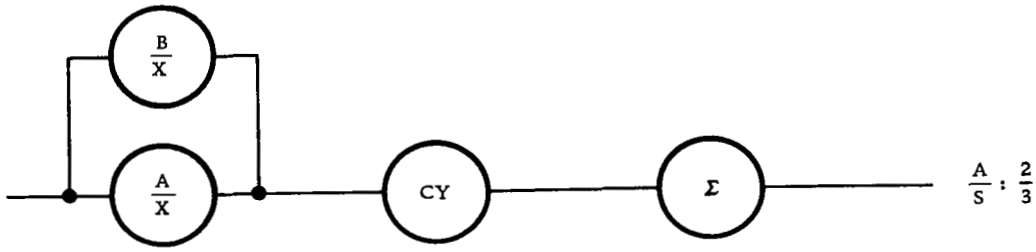


Figure 4-21. Applied Parallelism in the Computation

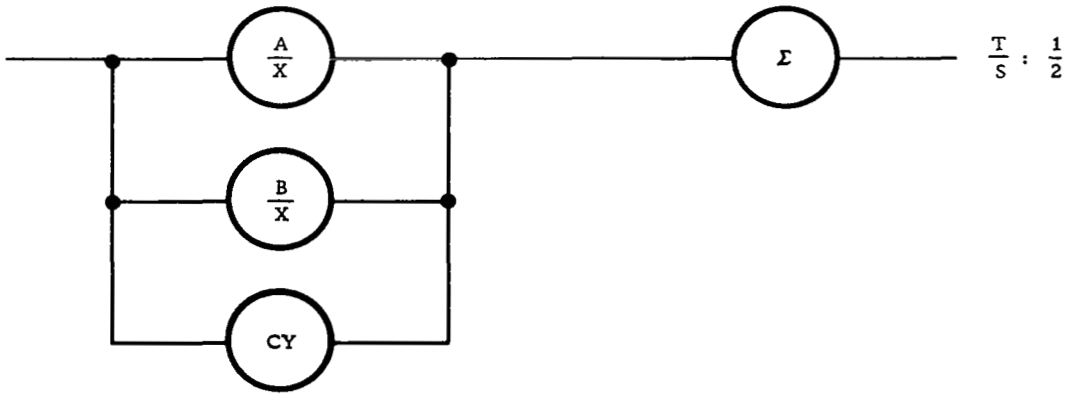


Figure 4-22. Applied and Natural Parallelism in the Computation

distributed machine since a global or central control function may be utilized to control processors performing the same operation simultaneously while the natural parallelism characterized by distinct operations simultaneously requires local control to be given to distributed processors. (Further discussion on this implementation will be given in section 4.3.2.)

It is for these reasons that the term "Total Parallelism" is introduced. This term indicates the combination of applied parallelism with natural parallelism as previously described in arriving at Figure 4-22. Basically, this term simply implies using applied parallelism where possible in the computation task and then natural parallelism where possible after that. The total parallelism then results in the ratio T/S = 1/2 for Figure 4-22 and in addition to the applied parallelism used here the degree of natural parallelism used is 2.

The computational tasks for phase 12, Mars Orbital, were analyzed to determine the ratios A/S and T/S; Table 4-4 lists the results from the analysis. It should be noted that the functions listed in Table 4-4 correspond exactly to those identified in paragraph 2.8, e.g. the scientific experiment functions correspond directly to those listed in Table 2-3. The quantity $\%_S$ indicates the time required on the single computer, A/S is the reduction due to Applied Parallelism and the $\%_{a/s}$, which is the time required on an Applied parallel computer, is given for all the sub functions such as "Orbit Determination." Also, the overall A/S for each sub function is given besides each "Total". Likewise the results with Total parallelism are given in the last two columns. The totals for all the functions are given at the end of the table and are repeated here:

$$\frac{A}{S} = 0.135$$

$$\frac{T}{S} = 0.025$$

This shows that Applied parallelism gives approximately a 7 to 1 reduction in computation time and also providing natural parallelism in addition results in a 40 to 1 reduction in computation time.

It is interesting to note here the differences in the ratios between the functions. For example, 1.1.3 gives an A/S of 0.001 and 1.3.4 of 0.00134 both these functions involve the manipulation of large matrices, thereby making use of applied parallelism; many other functions have numbers much higher such as 0.5 since they do not lend themselves to parallelism.

Therefore, considering these different ratios along with their relative requirement (percent of time) as was done in Table 4-4 gives a good indication of the effectiveness of parallelism when considering an overall problem for this space mission.

It should also be pointed out here that this subject of parallelism within computations has been studied to some extent in Reference 23. The investigations listed in that report were of parallelism within individual types of functions such as an NXN Matrix Inversion, etc.

Table 4-4. Reductions in Computation Time Due to Parallelism

Function	% s	A/S	$\%_{A/S}$	T/S	$\%_{T/S}$
1. Nav. & Guid.					
1.1 Att. Ref.					
1.1.1	1.31	0.531		0.531	
1.1.2	1.56	0.5		0.5	
1.1.3	0.28	0.001		0.001	
1.1.4	1.48	0.323		0.156	
1.1.5	1.51	0.046		0.046 →	
1.1.6	2.8	0.31		0.31 →	
1.1.7	2.6	0.384		0.384 →	
	11.54		3.895		1.937
Total		0.336		0.167	
1.2 Ldmk. Tkr.					
1.2.1	2.9	0.325		0.216	
1.2.2	---	---		---	
1.2.3	0.003	0.5		0.5	
1.2.4	1.1	0.04		0.035	
1.2.5	0.03	0.25		0.2	
1.2.6	0.003	0.5		0.5	
1.2.7	0.005	0.5		0.5	
1.2.8	6.1	0.41		0.41 →	
	10.141		3.497		2.500
Total		0.345		0.246	
1.3 Orbit Determ.					
1.3.1	0.016	0.3		0.25	
1.3.2	0.024	0.3		0.25 →	
1.3.3	0.0067	0.45		0.42	
1.3.4	4.65	0.00134		0.00134 →	
1.3.5	0.016	0.5		0.5 →	
	4.713		0.0298		0.0208
Total		0.0063		0.0044	
1.4 Orbit Integration					
1.4.1	0.002	0.66		0.630	
1.4.2	0.05	0.80		0.690	
1.4.3	0.028	0.41		0.223	
1.4.4	0.002	0.056		0.056	
1.4.5	0.009	0.585		0.284	
1.4.6	0.002	0.35		0.350	
1.4.7	0.26	0.5		0.400	
	0.353		0.189		0.119
Total		0.535		0.338	
1.0 Nav. & Guid. Total	26.75	0.285	7.61	0.093	2.500

→ (indicates functions combined to give total parallelism)

Table 4-4. Reductions in Computation Time Due to Parallelism (Cont)

Function	% s	A/S	$\%_{A/S}$	T/S	$\%_{T/S}$
2.0 Tele-Comm. Total	4.1	0.25	1.02	0.15	0.617
3.0 Sci. Exp.					
3.1 Data Comp.					
3.1.1	0.0235	0.167		0.167	
3.1.2	0.329	0.017		0.017	
3.1.3	0.395	0.0106		0.0106	
3.1.4	0.855	0.0175		0.0175	
3.1.5	1.0	0.0175		0.0175	
3.1.6	2.4	0.0175		0.0175	
3.1.7	51.5	0.05		0.033	→
3.1.8	2.4	0.00323		0.00323	
3.1.9	1.88	0.01		0.0094	
3.1.10	0.27	0.06		0.03	
	61.06		2.668		1.695
Total		0.0437		0.0275	
3.2 Sequencing & Scheduling Total	0.141	0.50	0.0705	0.25	0.0353
3.3 Pointing & Control Total	6.0	0.30	1.8	0.25	1.5
3.0 Sci. Exp. Total	67.201	0.067	4.538	0.0252	1.695
4.0 Sys. Check-out Total	2.3	0.15	0.345	0.12	0.275
GRAND TOTAL	100	0.135	13.513	0.025	2.500

→ (indicates functions combined to give total parallelism)

Thus far nothing has been mentioned with regards to the degree of each kind of parallelism required to achieve these reductions. In fact, these reduction ratios assumed all the parallelism that could be made use of in the problem was available. The computations were gone over to assess the problem of a finite degree of parallelism and the results are shown in Figures 4-23 and 4-24. It is seen that the maximum gain due to applied parallelism (1/0.135) is approached rather quickly and in fact, for this particular set of computational tasks no further gain was possible after a degree of 1331. This curve shows that a reasonable degree of applied parallelism that may be utilized is somewhere between 12 and 25. Beyond 25, the gains do not increase very much. It should be noted that this curve is a succession of steps since there is a certain gain for a degree of 1 and no more gain until a degree of 2 is utilized, etc. Another note is that this curve assumes no natural parallelism is available, all gains are simply applied.

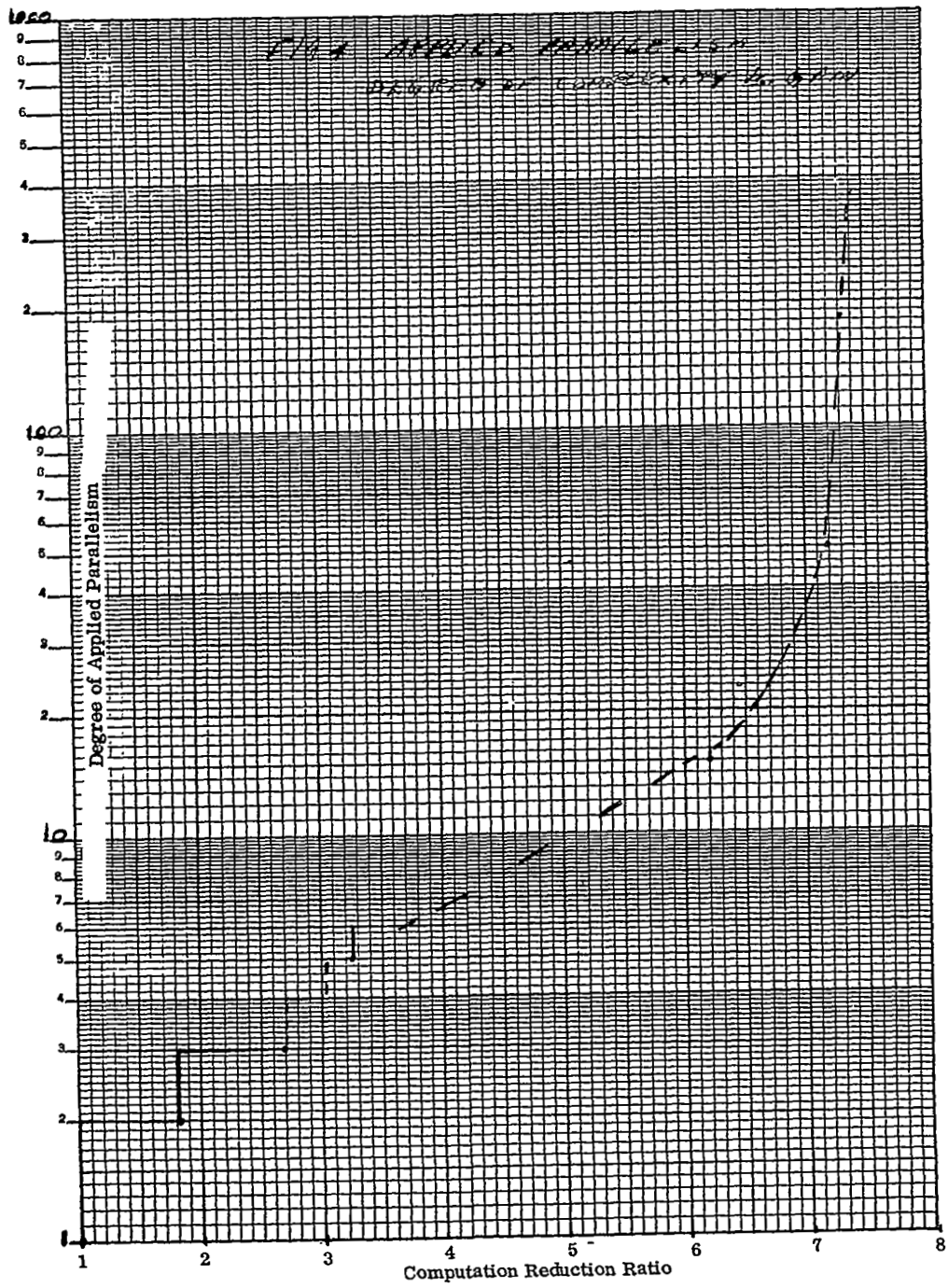


Figure 4-23. Applied Parallelism - Degree of Complexity Vs Gain

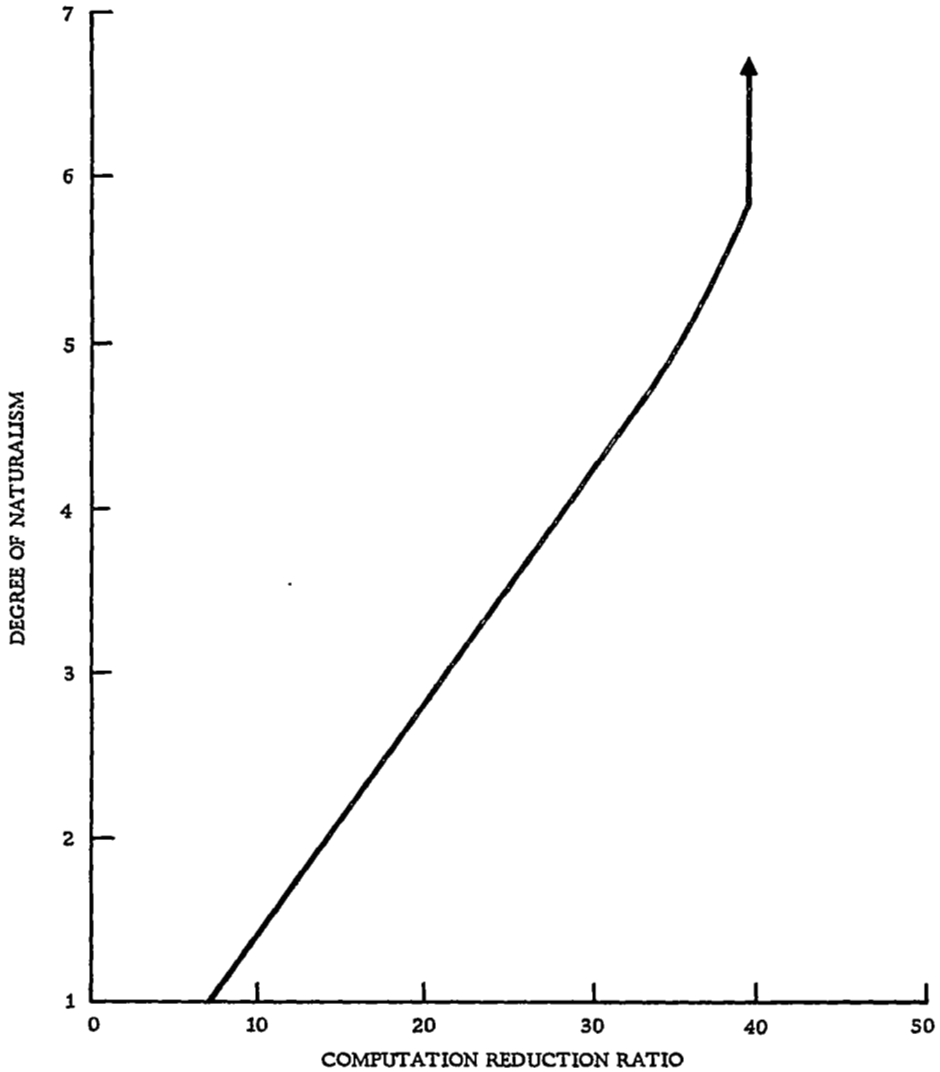


Figure 4-24. Natural Parallelism Curve

The natural parallelism curve is shown in Figure 4-24. This curve was derived assuming all the applied parallelism that could be utilized was available. This was done to single out the effects of adding natural parallelism on top of applied parallelism. The natural parallelism problem resembles that of a PERT routine. A maximum of 45 naturally parallel groups were conceived of in analyzing the computational tasks. However, all of these could not be effectively utilized. As examples, consider task 1.3 Orbit determination; this has a degree of 3 in terms of natural parallelism: 1.3.1, 1.3.3, and a group consisting of 1.3.2, 1.3.4, and 1.3.5 may be computed in parallel due to natural parallelism giving a degree of parallelism of 3. The worst case group is task 1.2.8, the time required for the three prior groups forming task 1.3 "Orbit Determination" is less than the computation time required for the group consisting of 1.2.8. Therefore, there is no gain in splitting the computations in 1.3 into a degree of 3. Another example may be given within a small function, consider 1.3.1. There is also a gain within this function due to natural parallelism and once again the gain is not sufficient to be used when considering the group 1.2.8.

Following a procedure such as this, one may arrive at the maximum degree of natural parallelism that may be utilized, of course, more may be used as described above, however, no gain results, this maximum degree was 6 for the total computational task. The points between 1 and 6 are difficult to determine since there are many possibilities of combining the given computations within a natural group to achieve the maximum reduction ratio with that number of groups. This curve approaches a straight line as the assignment of tasks is optimized.

From the results of this analysis, a natural degree of parallelism of 6 results as the optimum case. However, there are many problems which it appears will exist when combining tasks as described above into natural groups. Indeed, the communications between natural groups may reach very high rates due to chopping a given small function such as for example 1.1.1 in half and splitting it between two natural groups. Therefore, the optimum value of 6 may not be practically achievable. To determine what may be achieved would take an extensive additional effort and is not warranted at this time. It is estimated that a feasible number for the degree of natural parallelism for the total computational task may be on the order of 12 to 20; this does not include overhead functions such as executive program, self test, etc.

4.3.2 Distributed Processor Organization

4.3.2.1 General Considerations

This section describes a new distributed logic structure that is capable of carrying out computations while taking advantage of both natural and applied parallelism. In other words, the machine is capable of operating under local or global control. This should enable the structure to obtain high hardware utilizations while reducing instruction and data storage. The interesting property of this structure is that it was not designed to solve a particular type of computational problem as distributed logic structures have been in the past; but instead, it was designed as a general purpose computer to take advantage of the new MOS or MOS/SOS technologies and to provide very high reliability due to the use of many levels of graceful degradation.

The mission requirements given earlier in the report generally apply to this structure. In particular the same computational speeds must be met and the existence of critical computations means that there must be an on-line backup. Equipment should also still be kept off-line as much as possible to increase reliability. However, the amount of separate main memory required and the downtime due to repairs have no real applicability here. The latter point is not applicable since spares will be kept as fixed modules within a single computation structure. The different interpretations of the requirements will become clearer as the structure is explained.

The following description of the distributed processor gives a fairly complete conceptual description of the machine. Further developments of the explicit features of the machine will depend on a good amount of programming effort and software development. This will certainly result in new hardware trade-offs. As a result of the uniqueness of this design and of the lack of programming experience with distributed logic structures in general, a number of features such as the length of the instruction word or the number of operations will not be given as they were for the multiple computer and multiprocessor. However, the description is conceptually complete including a hardware and reliability estimation so that it may be compared to the other two candidate organizations.

4.3.2.2 Global and Local Control Structures

Two basic types of distributed processing were investigated. One type used global control, and was typified by the Solomon machine. The other type used local control as for example, the Holland machine. The Solomon machine uses many cells or processing elements executing the same operation in parallel or not executing the operations. A common control unit and common addressing of the cell memories is used. This machine is described in depth in the literature. This type of a structure takes good advantage of applied parallelism; however, it is not able to take advantage of the natural parallelism in the computations. In fact it was designed explicitly to be able to handle problems with a good amount of applied parallelism, such as solutions of partial differential equations or very large matrices. As a result, when this type of a structure is applied to general purpose problems it has low hardware utilization due to the fact that a good portion of the time a cell is not executing an instruction that is sent on the communication lines. This type of structure also does not take advantage of the graceful degradation ability inherent in a distributed structure since failures within the relatively complex control unit and memory could bring the whole system down.

The Holland machine uses many cells executing different operations in parallel. Any cell can operate as a controller, an accumulator or a storage unit. Paths to operands are then built from the controller cell to storage cells and then back to a cell designated as an accumulator in order to carry out the operations. This path building necessity extremely complicates the programming of the Holland type machine and also makes the reconfiguration problem after a failure very difficult. There have been a number of attempts to solve the path building problem in order to take advantage of the local control features of this type of a machine; however, an adequate solution has not been found. This means that this type of a structure also has low hardware utilization due to problems of paths crossing and of optimal programming. The structure also has the additional programming problem of a relatively small instruction set. The Holland machine, and a number of variations

to try to solve the path building problem are described adequately in the literature. Another disadvantage of the Holland type machine for general purpose computation is the fact that even though it is able to execute either naturally parallel or applied parallel computations, it is obviously inefficient if a large number of applied parallel computations must be carried out. These computations must be operated upon as if they were naturally parallel and as a result extra instruction and constant storage is necessary along with more programming effort to lay out the solutions.

4.3.2.3 Autonetics Distributed Processor

4.3.2.3.1 General Organization

The chosen distributed logic structure uses a number of groups of cells each carrying out a task in order to handle the computational requirements. This structure is shown in Figure 4-25. Each group of cells will actually carry out a complete task (such as a navigation and guidance problem), a number of tasks, or even a part of a task dpending upon how many computations are necessary to do a given problem. The primary consideration used in dividing programs amongst the groups is to limit the inter-communication amongst groups as much as possible. In this way the inter-group bus can be used primarily for communication to I/O variables and to one of the groups operating as the Executive. Within a group one of the cells operates as a controller and provides commands for the others. The individual cells can either accept these commands or execute commands from their own memory. In this way, both local and global control can be carried out simultaneously within a group. At the same time all the groups can be operating in parallel. This type of operation enables both the natural and applied parallelism inherent in the general purpose spaceborne problem to be efficiently carried out. It should also be noted that there is no main memory in the system. All instructions, contents and variables are stored within cells. Since it is also necessary for this system to reconfigure within five seconds during critical phases, a second inter-group bus in included in the structure. This is not shown in Figure 4-25.

Paragraph 4.3.3 discusses proper use of this computer system during the critical, non-critical, and Mars orbital phases. Very simply, during non-Mars, non-critical phases, approximately 12 groups with three more groups coming on and off periodically will be used to carry out the computations. There will also be four groups handling executive functions along with a number of spares connected to the inter-group bus but not operating. During critical phases, the system will be divided up into two sections. The first section will contain 12 operating and four executive function groups and will carry out all the computation of the system. The second section will use the redundant inter-group bus along with two computational groups and four executive function groups. Again, there will be a number of spares off-line (but connected to the inter-group busses) that will be available to either section. A lock-out feature of the same type as described for the Multiprocessor will be used in each group switch connecting the inter-cell bus to the inter-group bus. This will enable the system to operate in a manner similar to two separate computers and as a result be able to guarantee reconfiguration within the five second time constraint. During Mars orbital operation the system will use 20 groups doing the computations, four executive function groups, and a number of spares off-line but connected to the inter-group bus. From the above, we can see that reconfiguration after any group failure is simply a matter of detecting which has failed and switching

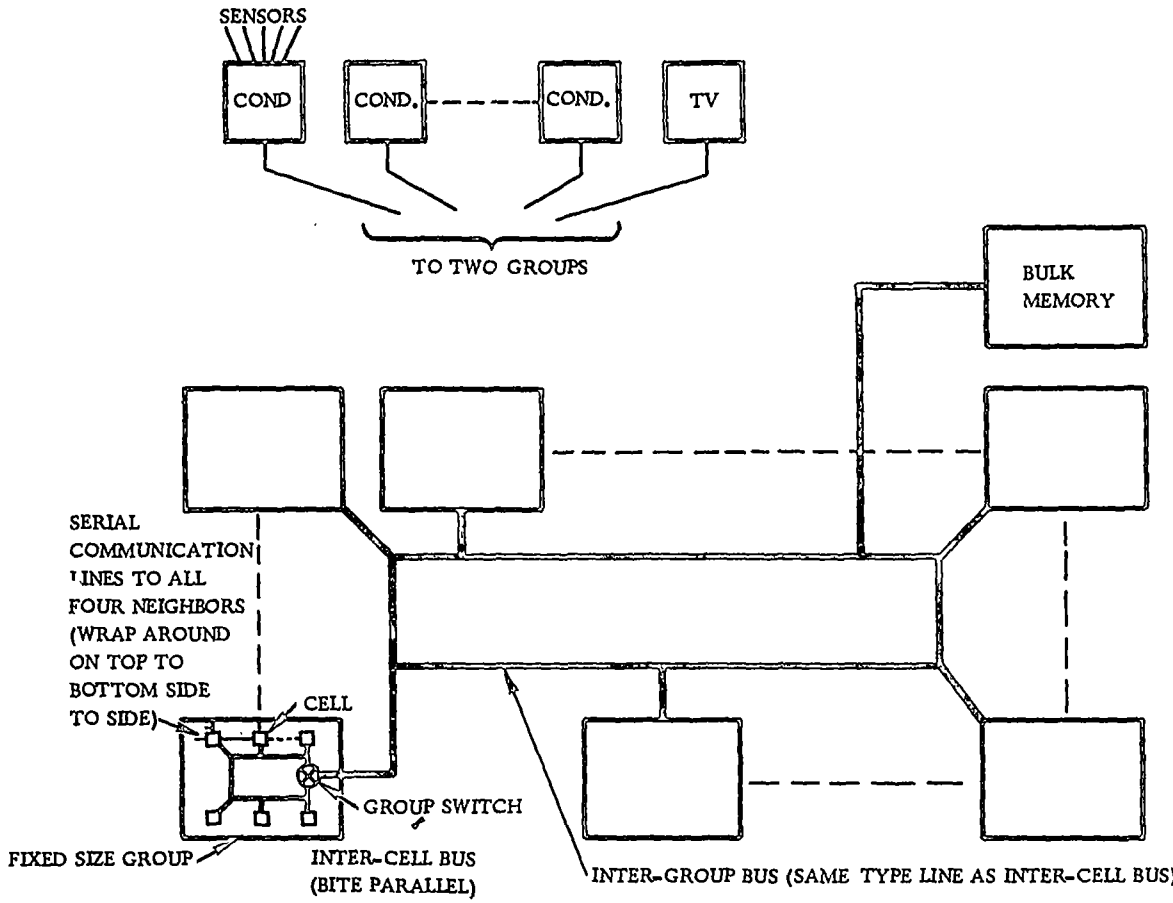


Figure 4-25. Distributed Processor

over to a spare group. No replacement will be necessary since all spares are on-line. This type of sparing philosophy obviously enhances the availability of the system. It is possible here since adding extra groups only means an extra connection to the inter-group bus and as a result will not increase the connections or size of the distributed processing system. On the other hand, this type of a sparing philosophy was not used in the Multiple Computer or Multiprocessor since a large increase in connections would have been necessary thus making the approach impractical.

In order to take best advantage of the MOS technology and to get a flexible general purpose structure, a cell was chosen to be one MOS SOS chip. In the 1973-1975 time frame this chip has been estimated to contain about 5,500 FET's on a 0.15 inch square chip. This organization would actually need a slightly larger chip (0.2 inch-square) so that about 10,000 FET's would be available. This is certainly reasonable if yields can be increased or if discretionary wiring can be used to connect redundant registers. With a large chip one-half of the chip could contain 32 18 bit registers for memory and control and the other half could be used for logic. This organization is then able to provide a good amount of processing power in a single cell. It also limits the intercommunications amongst cells since each cell only communicates to its four neighbors in a serial manner and in a bite parallel manner to the inter-cell bus. A picture of a cell is shown in Figure 4-26. This figure will be explained in depth later in this section. The limited intercommunication and the small size of the cells means that a complete distributive logic computer system of 625 cells could be included on a 8" x 8" two layer board. This structure providing on the order of 16,000 to 20,000 words of control, instruction, and data storage should be sufficient for the Mars Lander Mission.

The number of cells in a group was chosen to best suit the applied and natural parallelism inherent in the Mars Lander Mission computations. The development of the curves to determine the number of cells in a group is given in section 4.3.1. An analysis of these results along with an estimation of the executive and I/O functions suggested that a structure containing approximately 25 groups of 25 cells per group would be a good solution. The function of these groups as executive or I/O processors is given in paragraph 4.3.4. The operation within a cell will be carried out in both byte parallel and serial manners at speeds up to two megacycle clock rates. The intercommunications on both the intercell and intergroup bus can be carried out at least at one megacycle per bite rate. However, for the Mars Lander Mission, it appears that these cells could operate at a much slower rate than two megacycles and that the intercell and intergroup buses could also operate at less than one megacycle per byte communication rate. The exact speed to be used should be determined in future study of this candidate. As mentioned earlier, the tasks will be divided up among the groups so that intercommunication from group to group will be limited as much as possible. This will enable the intergroup bus to be used for executive monitoring and I/O communication. This division of tasks among groups will also enable the power to be turned off to certain groups during a number of phases. This, of course, increases reliability and lowers power dissipation. It should be noted that the storage in this structure is volatile; however, this should not be much of a disadvantage since the primary power supply will be backed up and will be of high reliability. Even if a power failure should occur, bringing the system back up only involves reloading information from the bulk storage unit. A volatility discussion was given for the semiconductor memory in Section VI. This discussion also demonstrated that volatility did not present any severe problems.

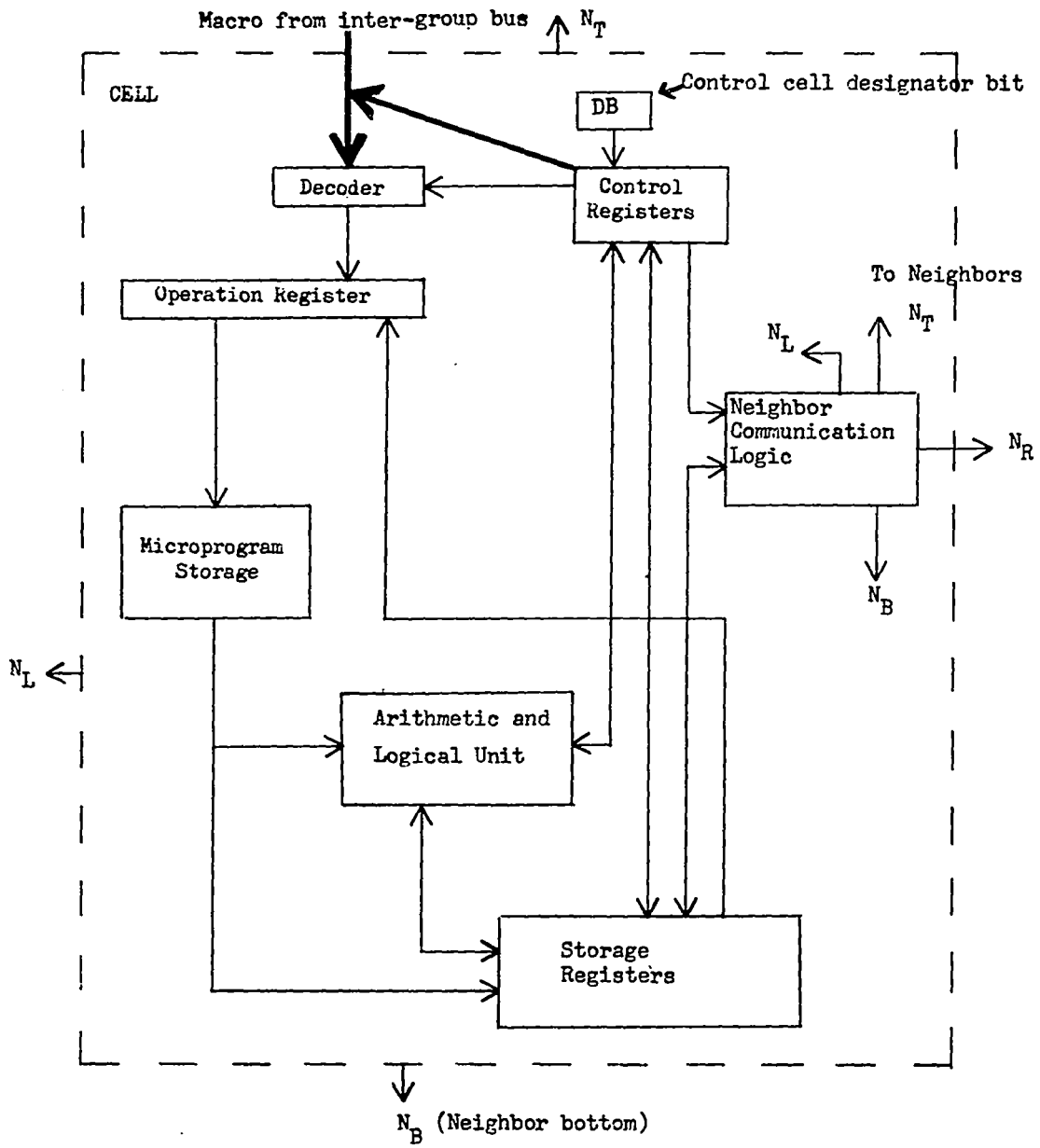


Figure 4-26. Distributed Processor Cell

4.3.2.3.2 Explicit Features

Each cell operates as the controller of the group, an operating cell, or a storage cell. The controller cell provides the global control for a group by placing macro instructions on the inter-cell bus. These macro instructions have yet to be defined. However, they will be instructions such as matrix inversion, sum check, sine, etc. The operation cells receive these macros from the controller or from their own storage registers, decode them, and then use them to read out a sequence of operations from the micro-program storage contained in a cell. See Figure 4-26. The sequence of instructions from the micro-program storage cause storage registers and control registers to be added, exchanged, or transferred to neighbors. Again referring to Figure 4-26 this means that the control registers allow a macro from the inter-cell bus to load the operation register or they allow a macro from a cell storage register to load the operation register. This operation is then carried out and the next operation obtained in the same manner. For some computations, a cell may need more storage than is available within itself. It can then use one of its neighbors as a storage cell. The neighbor communication logic is then used to obtain information necessary to the computation being carried out in a given cell. All the storage within a cell is addressable and can be used in arithmetic or logical operation. The controller cell, denoted by the designator bit, controls the use of the inter-cell bus by the operating cells and also controls the group switch shown in Figure 4-25. The group switch contains a small amount of decoding logic and a flip-flop register. If the inter-group bus contains a command, this command is let through onto the inter-cell bus immediately after the present transmission on the bus. If the command is for this group, it will be recognized and accepted by the controller cell. This group will then remain connected to the inter-group bus until the transmission is completed. If the command is not for this group, the group will be immediately disconnected from the bus. The command from the executive group contains a task name that can be recognized by the controlling cell. Each group switch (there is a switch for each of the two inter-group buses that connect to an intercell bus) contains a lock-out register. This register can be set by the executive group and operates in exactly the same manner as the lock-out register in the memory and I/O units of the multiprocessor. In other words, during critical phases the register is set so that any given group will only accept commands and communicate over one of the two inter-group buses. This, of course, enables isolation of failures so that reconfiguration can be carried out within the five second time constraint. The above description should demonstrate that the ability of this system, or of a group in particular, to operate with both global and local control means that good hardware utilization can be obtained while using a minimum amount of storage (it takes advantage of both applied and natural parallelism).

The executive uses a number of groups to control the two inter-group buses, to handle I/O, to handle communication with the bulk storage unit, to hold global data so that any group may use it, to handle data communication from group to group, to send out macros to load the system, and to allocate I/O time on the inter-group bus. The operation of the executive is described in more detail in the following two sections. Information over the inter-group bus is bite parallel. The number of bits in a byte should be determined in future study of this candidate. However, a rough approximation says that 9 bit bytes at one magacycle per bite rate would certainly be adequate. However, for any particular system, making the bus less parallel means that less drive must be provided by the cell. This, of course, results in lower power and higher reliability due to less connections and drivers. It should also be mentioned that the groups can use the inter-group bus only when sampled and allocated time by the executive. Information on this bus is tagged with control bits, names of tasks, data addresses and data itself. An example of possible word formats is given in Paragraph 4.3.4.

It should be noted that the groups are of fixed size. At first this may seem to provide a restriction on the ability to allocate tasks among the various groups. However, further investigation shows that fixed size groups actually alleviate many of the programmer's problems in optimizing tasks to groups. In particular after a number of reconfigurations due to failures, tasks that were optimized for one group size might not be able to fit into either smaller or larger groups in an optimum manner For example, a number of small tasks placed in a large group would provide many communication problems. It is also clear that the executive would have many problems in trying to shuffle cells from group to group and in matching tasks to varying size groups after a number of failures. As a result the need for reconfiguration flexibility and the need to provide the programmer with reasonably small groups of cells in which to optimize the program points toward a system structure as has been shown. These points are discussed to some extent again in Paragraph 4.3.4.

4.3.3 Failure Considerations and Reconfiguration

The introductory remarks and basic guidelines given in Sections 4.2.2.2 for the multiple computer candidate apply equally well to this candidate, so they need not be repeated. Therefore the discussion will begin with error detection and isolation tests.

4.3.3.1 Error Detection and Isolation Tests

The tentative conclusion for detecting errors in the distributed processor system is to use a group testing scheme. By this method all cells of a group are checked at the same time, rather than checking the individual cells within a group at different times. Further, at checkout time, the entire group is devoted to the checkout and does not participate in the operational problem. Because the distributed processor is a relatively new computer approach, a brief discussion of some other checkout approaches which were considered will be presented, followed by a description of group testing.

One set of testing schemes emphasized the use of tests that did not depend on the existence of self-test programs. Instead testing would be carried out using operational data.

By the first of these methods, error detection hardware would be built into each cell. No special testing mode would be required. Checks would be performed continually along with the operational problem. Parity bits would be generated and checked as the primary means of detecting data transfer errors within the cell and between cells. Outputs would be fed back and checked. Inputs would involve redundant receivers. Control circuitry would probably be redundant and checked for disagreement. Arithmetic logic could either be redundant or use check bits. The immediate disadvantage of this scheme lies in the large amount of redundant hardware required (probably more than double).

A second approach uses active redundant cells in a multiplexed manner. Referring to Figure 4-27, a cell group containing 20 operational cells and 5 test cells is depicted. The operational data flow paths between cells are not shown, only those for testing are shown. Cell T1 is responsible for testing cells C1, C4, C5, and C17; T2 for testing cells C2, C6, C7, and C11; etc. Periodically, during normal operation, four test time slots are reserved so that each test cell may check its adjacent cells. All test cells act in parallel, testing one operational cell at each test time slot. The test action at any test time slot consists of checking the results of the previous test, and if OK, setting up the test of the next cell by loading its contents into the test cell.

Upon resumption of the operational program, the test cell performs the same operational calculations as the cell under test. As an example, assume that T1 is now checking C5 and is to test C1 next. Operationally T1 is performing the same problem as C5. At testing time the results generated by T1 are compared with those generated by C5. If disagreement exists the error is reported to the executive processor to suspend group operation. If no disagreement occurs, T1 is loaded with the contents of C1 and will redundantly perform C1's calculations during the next operational cycle. Some of the pitfalls of this approach are as follows: First, there is a loss of flexibility of operational communication paths between cells since one of the four paths is used only for testing. Second, the ability to provide the test cell the same inputs as the cell under test during the active redundant test phase may pose a severe programming constraint. Third, the three operational communication paths are not checked. Fourth, errors in circuitry peculiar to the execution of a particular macro are detected only if that macro is being executed while the cell is tested. Therefore there could be an excessive delay (greater than 5 seconds) between the occurrence of an error and its detection. Fifth, some redundant hardware is probably required for disagreement detection. Finally, the symmetry and efficiency of test cell utilization is geometry dependent. For example, in Figure 4-27 each test cell uses all four of its inter-cell communication paths and 5 test cells test 20 operational cells. In a 4 x 4 test cell matrix the symmetric approach is to have 4 test cells checking 12 operational cells, with each test cell checking 3 operational cells.

A third approach reduces the redundant hardware by employing time-redundancy. By this method a sequence of program steps is performed by the group, the tasks of the cells within the group are interchanged, the program steps are repeated, and the results of the two executions are compared. The obvious disadvantage of this approach is the reduction in operating speed by a factor greater than two.

A second set of testing schemes differ from the above three approaches in that testing is performed by executing self-test programs. These approaches are more in-line with the testing philosophy of the multiple computer and multiprocessor candidates.

The first approach involves a "floating test-cell" concept. Here, one cell in each group contains a test program. In operation, all other cells perform the operational problem while this cell tests itself. After a prescribed number of program steps, the testing task of the successfully tested cell is exchanged with the operating task of another cell within the group, that other cell then testing itself during the next program sequence. Thus the testing is multiplexed, with all cells eventually executing the self-test, except possibly the group controlling cell. The biggest drawback of this method is the storage limitations of the individual cell. That is, it is highly unlikely that a comprehensive self-test program can be held in the storage registers of an individual cell. Note that a cell is in fact a processor and requires the same attention to its individual controls and registers as does the processors of say the multiple computer or multiprocessor candidates. Unlike those candidates, however, the cell doesn't have the memory capacity for test program storage. One might consider adding hardware to reduce the software test storage requirements, in effect providing a compromise between the error detection hardware testing approach previously mentioned and this all software approach. Similarly, additional cells within the group could be assigned solely to hold the checkout problem and conduct tests on each of the operating cells in turn. Each of these latter approaches may be possible, but have been disregarded in favor of the approach which conceptually appears to be able to do the job in the simplest manner and at a reasonably low cost in redundant hardware and time. This is the group testing approach which will now be described.

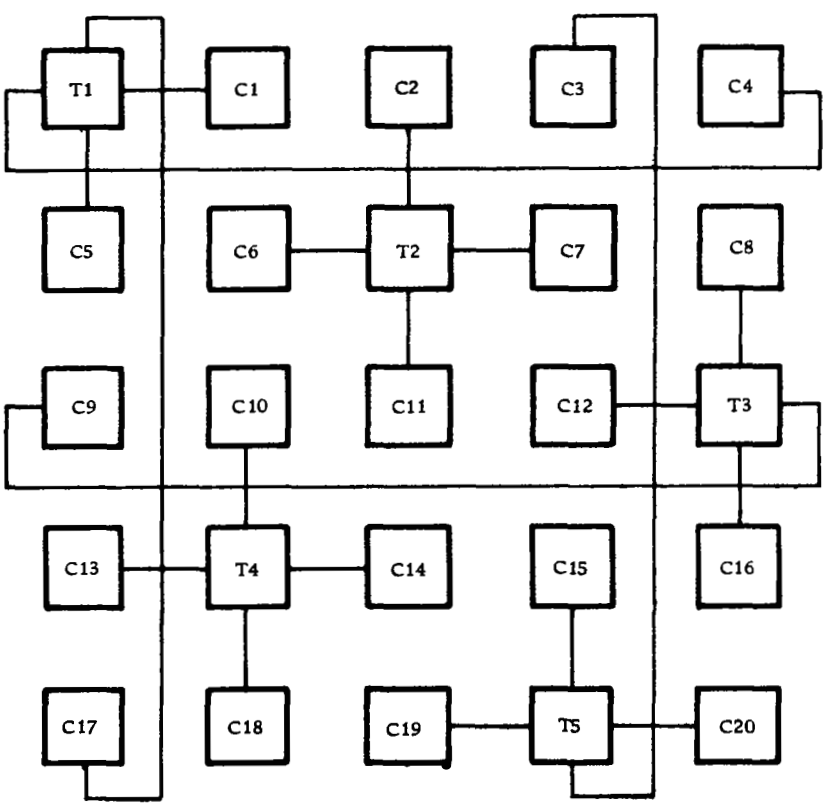


Figure 4-27. Active Redundant Test Cells Within a Cell Group

Figure 4-28 depicts the distributed processor configuration for group testing. There are an estimated 22 groups required for the maximum operational computations. These are G1, G2, ..., G20, Exec, and I/O. In addition there are two groups redundantly added for testing purposes; the Test Store and Temp Store groups. The Test Store holds the self-test program that is used to check each of the other groups and itself. The Temp Store group provides a temporary storage for the contents of a group when the group is being tested and performs the function of the group under test. As such, it is redundantly connected to the inputs and outputs of the system.

Groups are tested sequentially, not simultaneously, under control of the Executive Group. As an example of the operation, consider the testing of G1. First the contents of G1 are transferred to the Temp Store group which is then assigned the function of G1. Next, G1 is loaded with the test program from the Test Store group and G1 tests commence. If no error is detected the contents of the Test Store group are restored into G1 and G1 resumes its operational functions. Testing then proceeds to the next group in a similar manner. If an error is detected during G1 tests, the executive is notified and error isolation procedures are begun. Note that the I/O group is tested by the same procedure without disconnecting system inputs and outputs since the Temp Store group contains redundant I/O connections. Also the Executive group is similarly tested given that it can assign its executive function to the Temp Store group during its self-test.

Conceptually this form of group testing can be done continually during operation provided there is a test time slot available for transfer of data amongst the Temp Store, Test Store, and tested group. During this transfer the groups not being tested are idle. Assuming there are about 800-18 bit words to be transferred from a group, that three such transfers are required, and that the inter-group bus can accommodate whole word parallel data at a 1 mc rate, the 2400 words can be transferred in about 2.40 ms. Extrapolating linearly, about 5 ms are required if transfers are by 9 bit bytes, and 40 ms if transfers are serial. The size of the available test time slot would be determined during a detailed design effort. It would be based on the number of groups present in the most heavily loaded critical phase and the need to test each group at least once each 5 seconds in order to satisfy the critical reconfiguration time requirement.

Next, consider how the cells within a group would be tested. One or several cells of the group would be assigned the local executive function to conduct the test and report test results to the system executive. The remaining cells of the group would execute a test problem in the following manner. Each cell would perform the same macro at the same time, transmit the result of the macro to each of its four neighbors, check the data received from each of its four neighbors against its own computer result, and report to the local executive over the inter-cell bus. Assuming that only one failure will occur at any one time, a cell's failure will generally result in each of its four neighbors identifying it as a failed cell and probably it will identify all of its neighbors as having failed. The local executive will decode the failure reports and format a status message to the system executive over the inter-group communication system. Inherently this procedure provides the ability to isolate the error to the cell level and subsequently bypass the failed cell during operation. Then a group can continue operation even in the presence of one (or possibly more than one) bad cells providing the remaining computing power

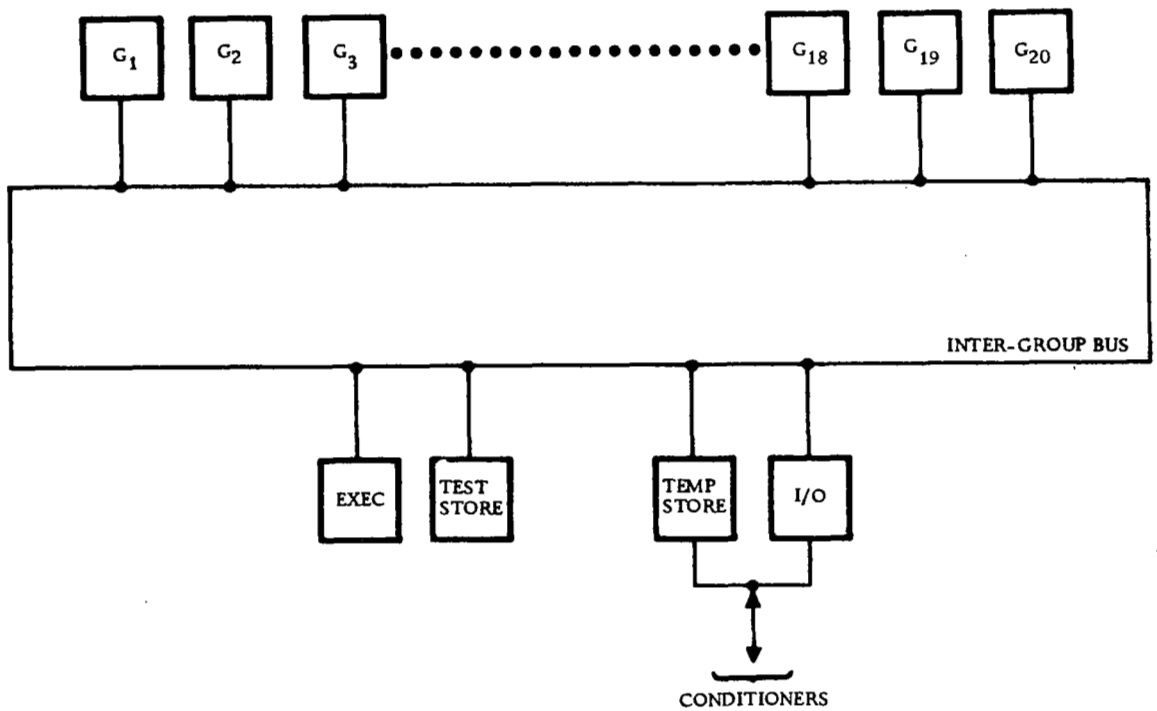


Figure 4-28. Distributed Processor Cell Group Configuration During Group Testing

of the group is sufficient for its assigned task. The extent to which this cell isolation is achievable would be determined during a design effort. A more conservative approach would be to bypass the entire group when any cell within the group failed.

Getting back to the actual test, all operational macros would be executed and in addition probably several macros specifically designed to aid the testing function. Examples of such special macros may be CHECK SUM and DISAGREEMENT DETECT, to quickly sum the contents of the cell's storage registers, and to test results received from a neighboring cell and report to the local executive.

It is likely that the testing will involve a second phase with a new local executive assignment in order to perform a complete check of the cell(s) used for the local executive in the first phase.

In addition to group testing, tests would be performed on input and output signals, in a manner similar to that described for the multiple computer and multiprocessor candidates, to detect and isolate errors in conditioners and input devices.

4.3.3.2 External Status Reports

The executive cell group controls the issuance of external status reports. Two types of reports are issued: one attesting to the ability of the executive to issue a report, and the other to indicate failures elsewhere in the system. The executive processor group is the obvious choice for this function since it receives the results of each of the group's tests and maintains a cell group status board table (described under software considerations in Paragraph 4.3.4).

The ability of the executive to issue a failure report is handled by a pulse stream detection method, similar to that described for the multiple computer and multiprocessor candidates. The BITE is mechanized in the I/O group, with the controlling commands for pulse generation originating at the executive and transmitted to the I/O group periodically. In general, failures in the executive processor or inter-group communication bus will cause an executive processor failure lamp to be lit.

The second status report type is the more normal one wherein the executive group reports the failure of another group or a conditioner or input/output device by means of a numeric readout.

As indicated in the following section on reconfiguration, two of the above sets of indicators are required: one controlled by the executive group of the primary system, and the second controlled by the executive group of the backup or secondary system associated with each of the phases.

4.3.3.3 Reconfiguration

This section discusses the task of reconfiguring the distributed processor after a failure has been detected and reported to the space crew. Inherently this system affords the highest potential probability of mission success and availability of all the candidates because an individual group can continue to operate in the presence of a failed cell(s), and because the system can continue to operate optimally in the presence of a failed group(s), given of course that spare cells are available in the group and spare groups are available in the system. Further, a large number of group failures can be tolerated prior to mission failure because critical computations can be sustained with relatively few operable groups. In essence, there exists a status zone between

full available computing power and mission failure for which additional failures may result only in the elimination of lowest priority computing tasks and not in mission failure. This zone can be termed one of degraded performance.

As in the case of the previous candidates the reconfiguration plan is based on the type of phase in which the failure occurs: either non-critical, critical, or Mars orbital.

Figure 4-32 presents the distributed processor configuration. The solid lines represent the portion of the system required solely to satisfy computational requirements. There are 22 cell groups (Exec, I/O, G1-G20), the primary intergroup bus, and conditioners $C1_1, \ldots, C1_N$. The dashed lines represent redundant hardware required to enable error detection, and rapid reconfiguration during critical phases. Included are the Test Store group, the Temp Store group, spare groups S1, . . . SN (the determination of the number of spare groups is discussed in Section 5), a secondary intergroup bus, and a redundant set of conditioners $C2_1, \ldots, C2_M$. In addition there are redundant connections from G3 and G4 to the conditioners. The functions of the redundant hardware will be described in the succeeding paragraphs, using the nomenclature given in Figure 4-29.

4.3.3.3.1 Non-Critical Phases

During non-critical phases the primary operational system consists of the following cell groups: Exec, Test Store, Temp Store, I/O, and that subset of G5 through G20 required to satisfy the computational requirements of the particular non-critical phase. In addition, the system includes the primary intergroup bus, conditioners $C1_1, \ldots, C1_N$, and associated input/output devices. The remainder of the items depicted in Figure 4-29 are off-line, and selected elements would be brought on-line during the phase only in the event of failure in the primary system, or in preparation for entry into a phase with different computing resources requirements (such as a critical phase or Mars orbital phase), or for backup assurance testing (as described in Paragraph 4.3.3.4).

Only failures in the primary system are considered. Failure considerations for the backup (off-line) system will be discussed in the section on backup equipment assurance.

Failures can be categorized as being either "hard core" or not hard core, and the techniques to isolate the failure source will differ. A hard core failure is one which prevents proper operation of the automatic error detection and isolation procedure.

First, the hard core failure will be discussed and how it can be handled. The hard core includes the Executive cell group, the Temp Store group, the primary intergroup bus, and the portion of the I/O group and a conditioner essential to the control of the executive processor control lamp on the control panel. These elements are somewhat similar to memory or processor failures of the other candidates which caused the computer fail light to come on. In this candidate the I/O and conditioner failures have been tentatively added to the hard core rather than giving the executive group a more direct line to the control panel. The reason is that the system executive may be flexibly assigned; any cell group can be the executive, and all would have to be provided the capability, thereby increasing hardware.

With the available flexibility of communication paths, there are many possible methods to isolate hard core failures. One such method, similar to the multiprocessor method, is to bring a portion of the backup system on-line and have it perform isolation tests on the hard core.

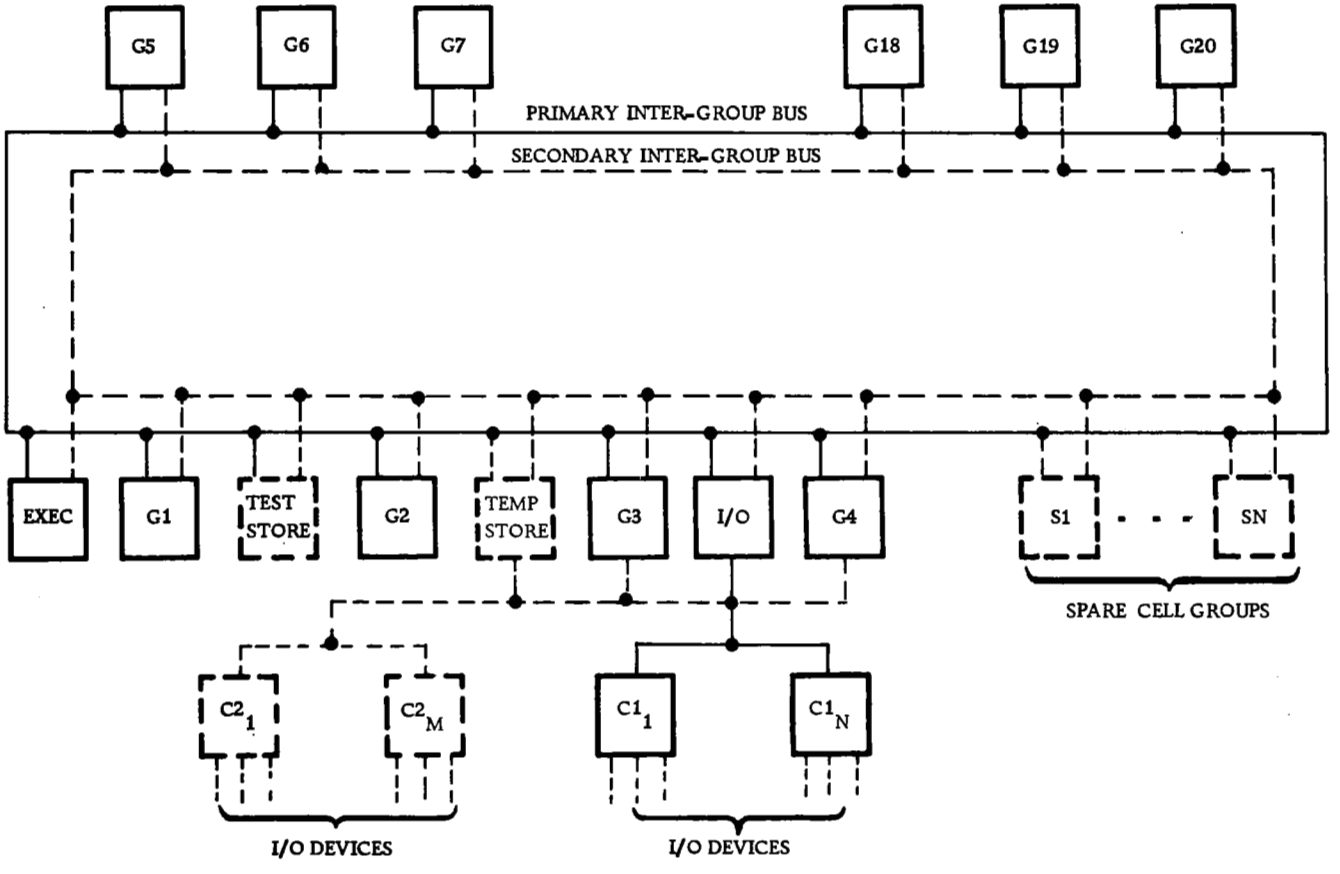


Figure 4-29. General Distributed Processor Configuration

The portion of the backup system conducting the test is G1, G2, G3, G4 and a redundant conditioner communicating with the control panel, say $C2_1$. G1 through G4 perform functions equivalent to the Exec, Test Store, Temp Store, and I/O cell groups respectively of the primary system. The functions of G1 and G2 may be flexibly assigned to idle groups of the backup system. The functions of G3 and G4 are assigned at the time they are connected to the conditioners and the assignment is flexible only in the sense that any two available cell groups of the backup system can be connected to the conditioners.

During testing the secondary intergroup bus is used. First the cell groups of the primary hard core are checked by the group testing method. If necessary, conditioner tests are performed next. If the failure has still not been detected and reported, it is likely to be in the hardware of any of the primary system cell groups connected to the primary intergroup bus. To automatically isolate the cell group causing the failure would involve additional tests wherein the suspected groups are turned on one at a time and the communications checked.

Next assume the failure is not in the hard core. Failures in the cell groups are detected by group testing as described in Paragraph 4.3.3.1. The failed group status is entered by the executive processor in its cell group status table and another available group is assigned the task of the failed group. Note that the backup system is not involved in this procedure (as opposed to primary system processor or memory failures in the multiprocessor candidate). Failures in conditioners or input devices would be handled similarly to the method described for the multiprocessor candidate.

The reconfigured system's structure depends on the failed element. For hard core failures the system consists of a new hard core and set of operating cell groups which can be assigned flexibly. For non-hard core failures of cell groups, the reconfigured system is identical to the original one except for the failed cell group whose function would be reassigned. Similarly, for failures in conditioners or I/O devices.

The present concept calls for spare cell groups to be designed into the system, thereby obviating the need for physical replacement after failure. Any of the available spare groups S1, . . . , SN may be brought on-line to make up for the computing power lost by the failure.

4.3.3.3.2 Critical Phases

The reconfiguration process for the distributed processor during this phase is similar to that described for the other candidates. Inherently, however, this candidate has the greatest potential for sustaining multiple failures during this phase with relatively the least redundant hardware.

During this phase the distributed processor acts functionally like two independent computers as long as no failures occur. Referring to Figure 4-29, the primary system consists of cell groups Exec, Test Store, Temp Store, I/O, a subset of G5 - G20 required for the operational calculations, and conditioners $C1_1$, . . . , $C1_N$ and associated input/output devices. The secondary system consits of the cell groups G1 - G4 which provide the Exec, Test Store, Temp Store and I/O functions for the secondary system, a subset of G5 - G20 (not used by the primary system), and conditioners $C2_1$, . . . , $C2_M$ and associated input/output devices. The primary intergroup bus is reserved for the primary system and the secondary intergroup bus is reserved for secondary system communication.

First, consider reconfiguration for the first failure in the system. If it occurs in the primary system, it is detected by the primary system in either a passive manner by the BITE circuitry associated with hard core failure, by the group testing error detection technique in which case the executive can identify the failed group, or by input-output signal testing as with the previous candidates. In any case, where the failure affects a critical computation, control of the vehicle is automatically passed to the secondary processor. This constitutes reconfiguration for this case and is accomplished in much less than the allowable five seconds.

Non-critical failures in the primary system result in a suspension of the associated computations, not in an automatic switch-over. Failures in the secondary system result only in failure notification.

Next, consider the actions which might be taken after the first critical failure to restore a backup capability for the critical functions and thereby be in a position to withstand a second critical failure. Regardless of whether the primary or secondary system contained the first failure, if it was a cell group not contained in the hard core, the executive can isolate the failed group, place it in a failed status, and bring a spare cell group on-line to assume the function. The system is restored providing parameter values affected by the first failure can be restored. There are several ways this can be done: either by using the last known good set of parameters of the failed system which have been continually stored during operation, or by obtaining the latest values from the system which did not fail. Thus, for a non-hard core cell group failure, any second failure during the critical phase can be tolerated.

If the first failure was hard core, the remaining good system can be requested to perform checks on the hard core of the failed system in a manner similar to that described for the non-critical failures. Where the failure is identified as either Exec or Test Store (or G1 or G2 for the secondary system) the full backup system can be quickly restored by reassigning tasks. Any second failure in the phase can then be tolerated. If the failure is identified as Temp Store or I/O (or G3 or G4 for the secondary system) a full backup cannot be quickly restored because of the need for physically changing connections to the conditioners. A compromise is achievable by assigning an unused cell group to the Temp Store function and then, for the remainder of the critical phase, checking all elements of the reassigned system except the I/O group. Alternately, no compromise need exist if additional groups were provided backup communication paths to conditioners. If neither of these two alternatives were acceptable, the full backup is not restorable, but all the remaining groups of the failed system would be available in a more restricted role for future failures in the good system. Finally, if the hard core failure were identified as on an intergroup bus, it is questionable if the full backup could be restored during the phase because of the actions required to identify the failed group. Again though, all groups of the failed system are available to the good system, on the operating intergroup bus. Similarly, where conditioners or I/O devices fail, a full backup is not restorable, but the remainder of the system is available for use in the event of subsequent failures.

As was the case in the multiprocessor candidate, single point failures that bring the entire system down are of concern. To guard against this an intergroup bus lockout feature, under executive control, has been incorporated. In effect the primary system is locked out from the secondary intergroup bus by the secondary system executive, and vice versa. This feature was described in detail in Paragraph 4.3.2.

4.3.3.3.3 Mars Orbital Phase

Referring to Figure 4-29, all elements depicted play an active role in the system during this phase with the exception of the spare cell groups and possibly the conditioners $C2_1, \ldots, C2_M$, and the secondary intergroup bus.

As opposed to the previous candidates, a minimal backup navigation and guidance function is not performed and hence for certain failures reconfiguration time would be greater than otherwise attainable. Since critical functions are not performed in this phase, there is no 5-second reconfiguration time constraint and the probability of mission success is not affected. The potential increase in reconfiguration time may decrease system availability, but this effect is reduced since only failures in selected portions of the system can cause it. The availability effect is described in Section 5.

The main reason for not implementing the backup navigation and guidance function is the additional cost in hardware, over and above that depicted in Figure 4-29. About six extra cell groups would be required. The rationale behind this will now be explained.

To implement a backup it is necessary to minimize circuitry whose failure would cause both the primary and backup functions to fail. This results in a configuration similar to that in critical phases. Thus a second Executive, Test Store, Temp Store, and I/O cell groups are required. Whereas for critical phases G1 - G4 were assigned to these functions, during the Mars orbital phase they are not available since the system has been sized to reduce hardware by using G1 - G4 for primary computations. Thus four extra groups would be required to perform these functions. In addition, it would be necessary to perform the minimum navigation and guidance calculations in cell groups not performing the primary function, and to communicate results from the prime to the backup for updating purposes. This would probably result in the addition of two more cell groups.

Failures during this phase can be categorized as either hard core, not hard core but affecting the navigation and guidance function, or not hard core and not affecting the navigation and guidance function. In the latter two cases, failures are self-isolatable by the system and where the navigation and guidance function is not affected, reconfiguration is fast, simply requiring the assignment of a spare group to the failed group's function. Where the navigation and guidance function is affected, the spare group is again assigned the failed group's function, but in addition an initialization routine is required since previous values have been lost. It is estimated that it may take approximately 1/2 hour to reimplement the N and G function. Finally, for hard core failures a procedure similar to that described for non-critical phases can be used with one exception. The exception is that the isolation tests would probably be conducted by a hard core made up by reassigning elements of the primary system not in the original hard core.

4.3.3.4 Backup Equipment Assurance

As in the case of the other candidates the backup or off-line equipment is periodically tested in order to insure its ability to take over an on-line role as required.

Referring to Figure 4-29, during non-critical phases the off-line equipment consists of cell groups G1 through G4, the subset of G5 through G20 not required for the computational requirements of the phase, and spare cell groups. In addition, it contains conditioners $C2_1$, . . . , $C2_M$ and associated I/O devices. Normally, this equipment is required either to furnish spares for the primary system, to be operably configured as an active standby redundant system during critical phases, or to provide the additional computing power required during the Mars orbital phase. Tests on the backup would be performed on a request basis. Testing may be conducted and controlled by the primary system, interleaved into its computational cycle in available dead time, or may be largely divorced from the primary system by assigning cell groups G1 through G4 the test controlling role. The latter case will be assumed since reconfiguration for hard core failures in non-critical phases involves the existence of a secondary executive system, as does the preparation for entry into, and the action within, critical phases. Once G1 through G4 are assigned, the test program is loaded from bulk storage. Data is entered to denote which cell groups, conditioners, and input devices are to be tested. Testing then starts, with all communications initially proceeding over the secondary intergroup bus.

The bus lockout feature is used to isolate the primary system from the backup system and thereby reduce the possibility of errors in the backup system affecting primary system operation. Group testing, as described in Paragraph 4.3.3.1, would be used to check all cell groups. Conditioners and input devices can be checked with operational type problems if the input devices are available. Where input devices are not available a combination of built-in test stimuli and the routing of conditioner outputs to inputs would be used. Test results are reported by means of the secondary control panel readout. Where failures exist, the failure can be classified as hard-core in which case further isolation testing is required, or as not hard core in which case the failed group is identified and removed (electrically) from the system. If no failure is detected, the second phase of testing is entered. This involves interface tests, i.e., the ability of the primary system to communicate with elements of the backup system and vice versa. Test results are reported by means of the respective executive readouts. If no failure is detected the backup system has been completely verified and is returned to the off-line status. If a failure exists further testing may be required for isolation, possibly with new executive assignments.

During critical phases a portion of the backup system is on-line in active standby redundancy and is tested operationally similarly to the primary system. No testing of the off-line portion is expected to be done during these relatively short periods.

During the Mars orbital phase the maximum computing power is on-line. The several spare elements that comprise the backup would be checked periodically by the primary system.

4.3.4 Software Considerations

4.3.4.1 Reconfiguration Flexibility

The basic difficulty in programming a distributed-logic machine lies in trying to achieve optimal code (i.e., maximum machine utilization and no usage conflicts) while retaining reconfiguration flexibility. With enough work, a single program can be mechanized in an appropriate array of processor cells, Figure 4-30(a), so that in all possible parallel execution sequences no path-building conflicts arise, and with minimal delays, unused cells, and overhead. If one cell should fail, however, the optimum solution for the new array, Figure 4-30(b), might be unrelated to the original in any direct or predictable manner. At least $(k - 1)!$ solutions would be required to anticipate handling k failures with optimality. For the mission under study, reconfiguration is also necessary to meet several mission phase requirements. Each phase is sufficiently unique (with the possible exception of some coast phases) that a separate solution for each is required. Add to this the fact that unanticipated computations must be processed and therefore the need for reconfiguration flexibility is overwhelming.

The main software advantage of the candidate distributed-logic machine, Figure 4-31, is the fact that it is modularized into approximately 24 mutually exclusive standardized groups of processor cells. This approach permits programming of the mission function into locally optimum, group-size task-modules. It must be noted that overall optimality, for any one configuration, is not as good as it could be for a non-modularized machine, but the optimality level will be consistent after multiple reconfigurations due to mission phasing, unanticipated requirements, or failures. Better overall optimization might also be achieved if the groups were of various sizes and tailored to the specific task needs. Once again, however, the necessary reconfiguration flexibility would be missing.

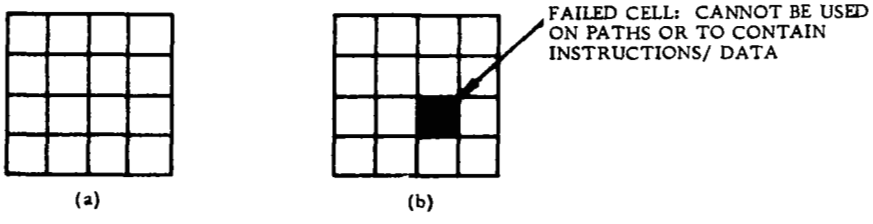


Figure 4-30. Logic Array

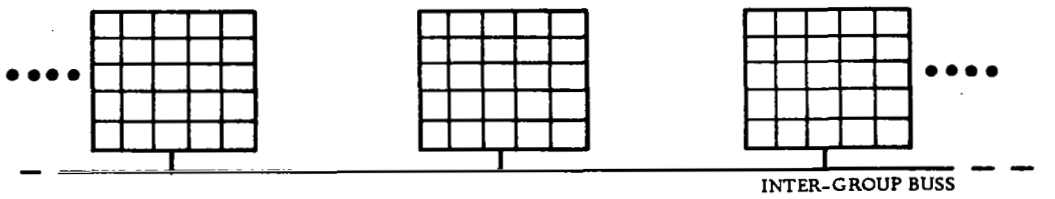


Figure 4-31. The Cell-Group Machine

This flexibility of this organization provides several side-effect benefits including the following:

1. Reduced Executive - The monitoring procedure need be concerned only with groups' status and assignments; the reduction in table size alone is significant. Of course, the reconfiguration process is itself greatly simplified.

2. Graceful Degradation - So long as spare groups are available, no loss in computational capacity will occur. Otherwise, low priority tasks can be deleted or backup task modules, which require fewer groups, can be loaded.

3. Time-Sharing - Those task-modules performing only non-continuous functions during a phase can time-share groups.

4. Easier Programming - It is much simpler to optimize programs for a 25 cell array than a 300 cell array and modifications to one task-module will generally necessitate redesign of just one cell-group's coding, not the entire program.

5. Standard Load Format - The task-modules can be organized easier on the mass storage since they are standard-sized. This also means that a single algorithm will handle all task-module loading into the groups.

4.3.4.2 Support Software Design

The Executive processors, including the I/O Supervisor, will occupy two cell-groups. The most striking difference in the software design for this candidate and the two previous ones (described in 4.2.2.3 and 4.2.3.3) is that program, or task, sequencing is not required whereas a new function, inter-group communication, is necessary. Another significant change is that some executive functions are localized within the task-modules.

4.3.4.2.1 Inter-Group Communication System (ICS)

Each task-module is constructed to be essentially independent; in fact, the size of the cell-groups is partly determined by the capacity requirements of the various independent mission functions. However, some inter-group data transfers will occur on the Inter-Group Bus due to the following:

1. Global data, i.e., those parameters of interest to more than one independent function, must be passed from the task-module which computes it to those that use it. This is not done directly, but through the Global Data Area described below.

2. Some mission functions are too large to be programmed in only one task-module. When multiple task-modules are used, the interface data necessary to connect the parts of the function must be passed.

3. The I/O System will use this bus to pass some data to and from task-modules.

4. Executive macro instructions are issued on the Inter-Group Bus.

5. Messages to various Executive processors must be passed from the task-modules.

In order to avoid conflicts, the ICS monitor controls the usage of the Inter-Group Bus. Each task-module in the computer is allotted weighted usage on a time-shared basis. When a task-module is activated, the number of accesses available per cycle required is allocated; the total number of accesses available must be large enough to handle the highest possible number of cumulative requirements. At a fixed rate, the ICS monitor selects, in turn, particular task-modules for use on the line as follows:

1. Task-module output - A macro is issued to enable the task-module to write a data item (a parameter value or a message) on the line. The data item, which could be null, is then read by the ICS monitor and passed to the ICS decoder, which determines its destination.

2. Task-module input - A data item or macro is written with a key identifying the destination task-module. Only the designated task-module has the proper mask to permit reading of the input.

3. NO-FAIL indication - The task-module must write a special data item which signifies a NO-FAIL condition in that cell-group.

A certain portion of this time-share cycle is reserved for exclusive use by the Executive processors; this will vary depending on how many task-module accesses are currently required. This "executive-time" is used to issue macros intended for all or some of the task-modules.

Each task-module is continuously attempting to use the ICS Bus. A key mask, which is unique for each task-module and is not hardware oriented, is used to control actual access.

In order to avoid extensive bookkeeping in trying to distrubte global data to the proper task-modules a Global Data Area (GDA) is maintained in the ICS. Every global parameter that will be used anywhere in the entire program is allocated fixed location register storage. When a global data item is output from a task-module it will contain a header identifying it as such and its fixed location in the GDA. This header is interpreted by the ICS decoder and that data value is stored in the GDA. When a task-module requires global parameter a message to the GDA monitor is output giving the location of the item and the header to be placed on it before it is input to the task-module.

Figures 4-32 and 4-33 contain examples of possible word formats. These formats are only included to facilitate understanding of the software communication routines and will not be explicitly specified at this time. Further study on an organization like this should include investigation of possible formats. The fixed codes used in Figure 4-32(a) are:

1. C - Control bit = $\begin{cases} 01 \\ 00 \end{cases}$ | $\begin{matrix} \text{data item} \\ \text{message} \end{matrix}$
2. KEY - Key specifying which task-module or Executive processor the data value or message is to be transmitted.
3. ID - The fixed location of data values in the receiver or a message number.
4. TEXT - A data value or a message.

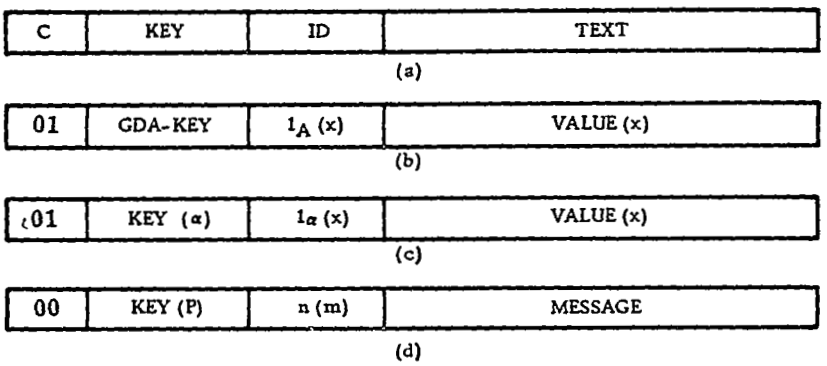


Figure 4-32. Output Data Item/Message Formats

Figure 4-32(b) is an example of a data item containing a value for a global parameter X; a key to the Global Data Area the parameter's location and the value of X are present. Figure 4-32(c) represents a parameter X that is to be input to a task-module. Figure 4-32(d) shows a message being sent to an Executive processor (P); when a processor may receive multiple messages a message number, n(m), must be included.

The data item format is shown in Figure 4-33(a), (b), and the macro format in 4-33 (c), (d); field codes and an example is given for each. The field codes are the same as for Figure 4-32 except as follows:

1. C = $\begin{cases} 11 \mid \text{data item} \\ 10 \mid \text{macro} \end{cases}$

2. S = Selective switch - $\begin{cases} 1 \mid \text{Select task-module via key to execute} \\ 0 \mid \text{All task-modules execute} \end{cases}$

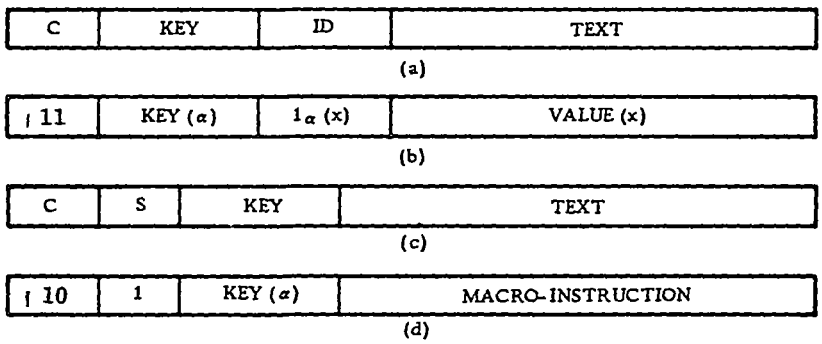
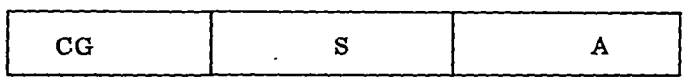


Figure 4-33. Input Data Item/Macro Formats

The ICS monitor does not request or schedule data transmission except as directed by other Executive processors. However, since more than one input to a task-module may be specified, during a time-share cycle, each task-module will have a first-in-first-out data item queue maintained in the ICS.

4.3.4.2.2 Reconfiguration Program

Two main tables are used to monitor and control the configuration of the computer; these are cell-group Status Board and the Task Status Table. Formats of these tables are illustrated in Figures 4-34 and 4-35, respectively.



where:

CG: Cell-group internal name
S: Status Failed, spare, dormant-task, active task
A: Assignment (Task Status Table entry)

Figure 4-34. Cell-Group Status Board Entry

TK	S	CG	LP	Q	PR	ICSA

where:

TK: Task key
S: Status (Active, dormant, requested, delete-flag unloaded)
CG: Cell-group assignment
LP: Mass-storage location
Q: (Tasks in cell-groups) Query mask for NO-FAIL indication verify.
(Tasks not in cell-groups) Queue pointer for wait string.
PR: Priority (non-interruptable, immediate, ASAP, O)
ICSA: Number of ICS accesses required per time-share cycle

Figure 4-35. Task Status Table Entry

As with other candidates, there are three conditions that can necessitate reconfiguration: failure, phasing, and unanticipated requests. There are two primary means of performing reconfiguration: dead restart and transition.

A dead restart must be performed whenever a power failure or ICS failure occurs since, in both cases, the computer is down and the volatile registers are wiped out. When the failure has been corrected, a special load program can be keyed in automatically or manually via the console. This program performs a computer verification test and loads in the Executive task-module which then controls the loading of the other task-modules. A logical diagram of this load program is shown in Figure 4-36.

All other conditions cause transition mode reconfiguration, which is performed in the Reconfiguration Program. Figure 4-37 shows the logic of a phasing reconfiguration. Figure 4-38 (a) and (b) shows the additional logic for failure and unanticipated request reconfiguration. The basic process consists of identifying task-modules to be deleted, scanning a Load Profile, making task-module assignments, and initiating loading. When a task-module terminates execution, it must send an "end-of-task" message to the Reconfiguration Program. This will enable processing of the wait queue of request programs.

If multiple failures reduce the number of cell-groups such that "non-interruptable" task-modules cannot be loaded, a backup load must be initiated. This would consist of critical task-modules and a priority ordered list of backup-mode, non-critical task-modules.

The Load Profiles mentioned above are located on the mass-storage and consist of pointers to initial loads for task-modules. There are Load Profiles for each phase (primary and backup) and for each individual task module.

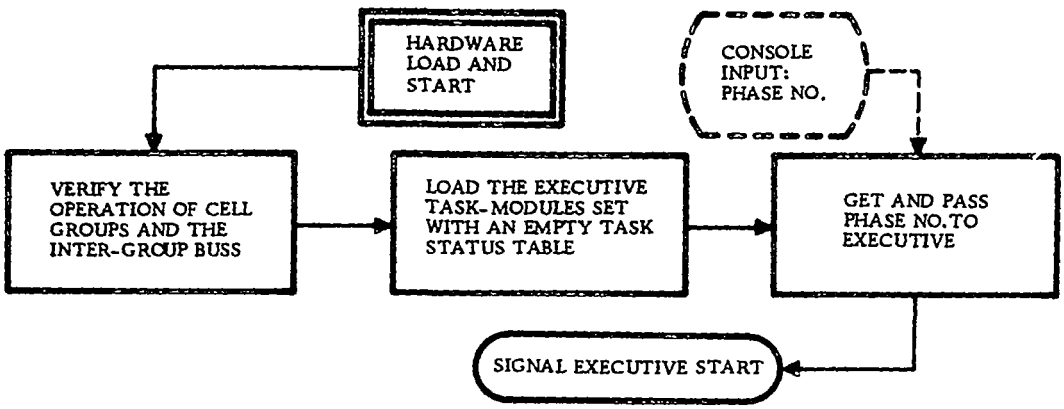


Figure 4-36. Dead Restart Program

4.3.4.2.3 Request Processor

A task-module may upon testing a condition or receiving an input from the console, or completing an assignment, request that another task-module be executed. This request is issued in the form of a message to the Request Processor which contains the key of the requested task-module.

When a request is received, an entry for the task-module is made in the Task Status Table. If an entry already exists, it is checked to see if it is already loaded; if so, an initiate execution command is issued. In all other cases the unanticipated request entry of the Reconfiguration Program is executed.

The request message can also contain a priority to be assigned to the task-module.

4.3.4.2.4 I/O Supervisor

The cell-group which contains the I/O Supervisor is connected to the conditioners.

When a task-module wants to input a parameter, it sends a message via ICS to the I/O Supervisor. This message indicates which sensor is to be sampled and a header which is to be added so that the value may be sent as a data item, via ICS again. Reasonableness tests are performed in the task-module, not in the I/O Supervisor.

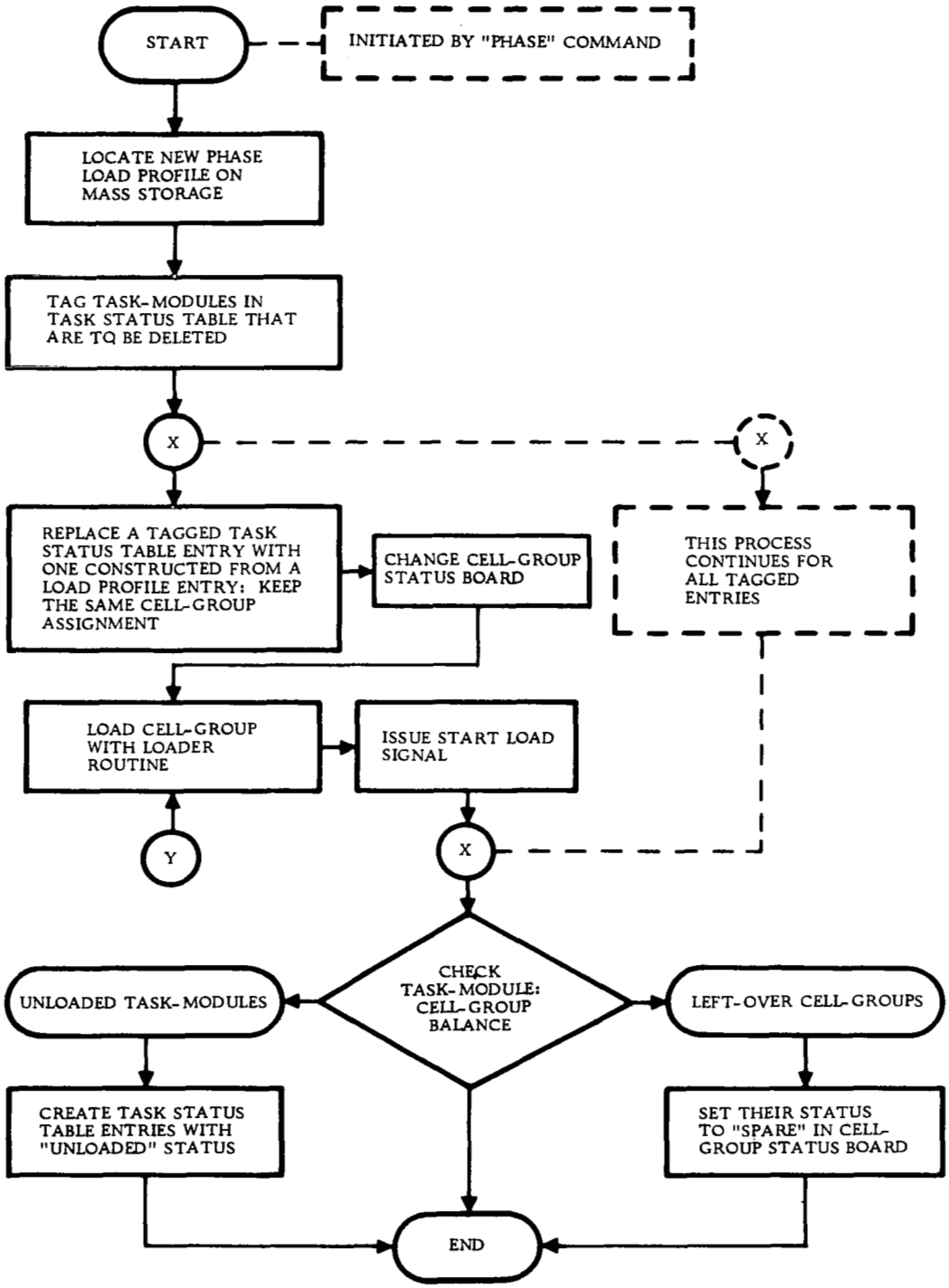


Figure 4-37. Transition Reconfiguration (Phase Start)

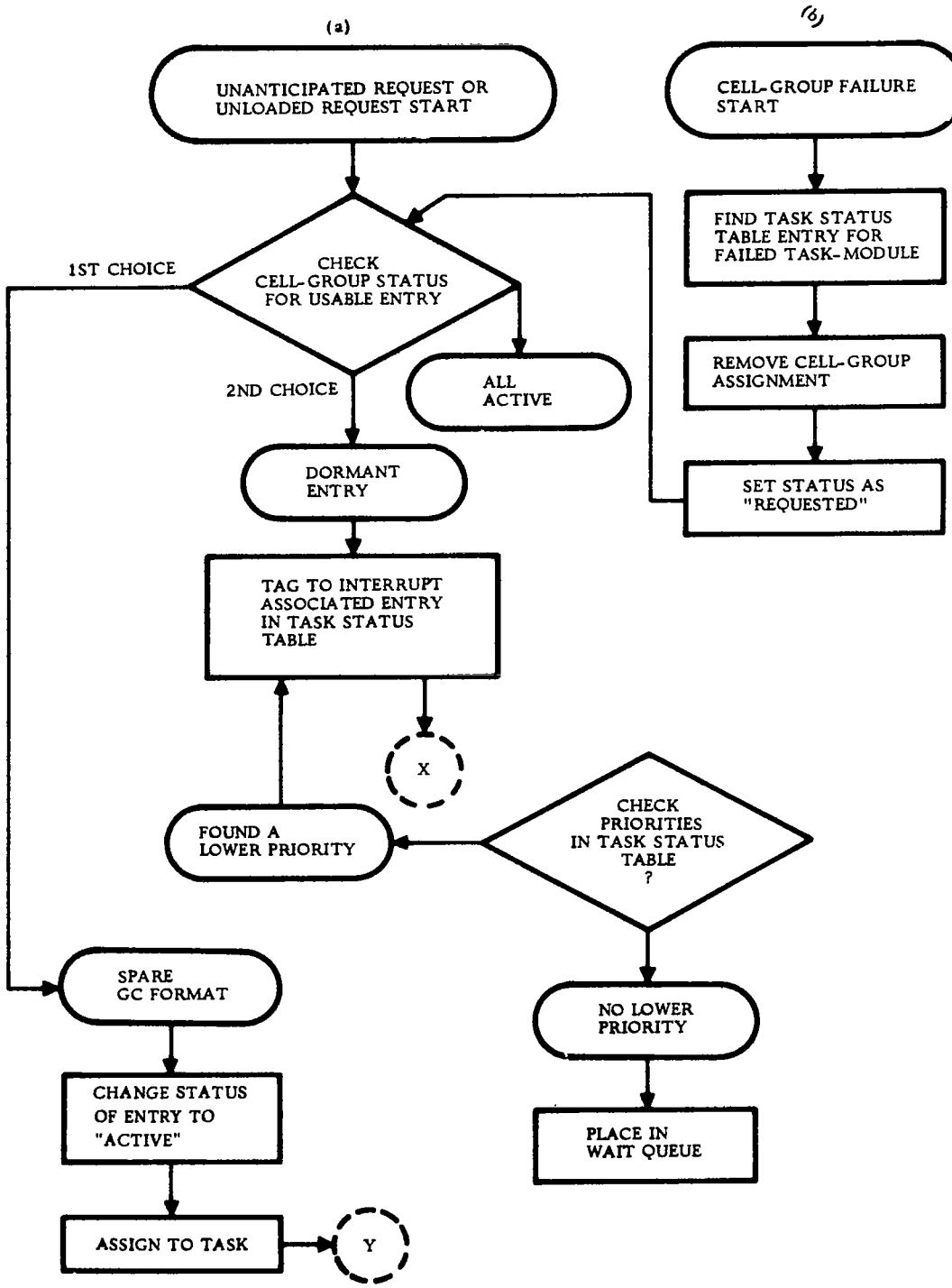


Figure 4-38. Transition Reconfiguration (Unanticipated)

Outputs are handled similarly, except that two transmissions, a select message and then the data item, must be sent by the task-module on the ICS. Verification of output feedbacks is performed in the I/O Supervisor.

4.3.4.2.5 Self-Test Program

(See Paragraph 4.3.3.)

4.3.4.3 Task-Module Software Design

Each task-module is required to perform certain "local-executive" functions. In general, the actual construction of these routines will vary within each in order to achieve local optimality.

The controlling cell of the cell-group will, of course, contain all the scheduling logic for the programs in the task-module.

The logic to transmit the NO-FAIL indication and the end-of-task message must be contained in each task module.

4.3.4.4. Estimate of Software Overhead

The overhead costs are considerably higher for this candidate and are expected to be on the order of 20 percent. This is based on the fact that two cell-groups are used for Executive and I/O operation and at least one cell in each cell-group assigned to task-modules is required for local executive control. In addition, two cell groups will be required for the self test operations.

V. SIMULATION AND EVALUATION OF CANDIDATE ORGANIZATIONS

5.1 SIMULATION AND RELIABILITY ANALYSIS

5.1.1 Monte Carlo Method

5.1.1.1 Introduction

The reliability and availability of the various candidate configurations were investigated by means of a Monte Carlo reliability analysis program. This is a computer program which generates simulated statistics for each configuration. Some consideration was given to closed form analytical expressions for the reliability analysis. However, the mission complexity proved this to be too difficult to derive in the time available.

Monte Carlo techniques of analysis refer to the simulation of random variables in a process by the generation of random numbers or sequences. For reliability analysis, the random event which is simulated is component or subsystem failure. It has been found that electronic equipment exhibit random failure rates which have an exponential distribution. That is, the probability of failure as a function of time, $P_f(t)$, is exponentially distributed.

$$P_f(t) = 1-e^{-\lambda t} \text{ where } \lambda \text{ is the expected failure rate.}$$

This equation is interpreted as meaning: Given that the equipment is currently failure free, the probability that a failure will have occurred by some later time t is given by $P_f(t) = 1-e^{-\lambda t}$.

It can be shown that the expected time to failure is λ^{-1}. The probability density function for $P_f(t)$ is $\lambda e^{-\lambda t}$. The expected value of time, E(t), is then found by

$$E(t) = \int_0^\infty \lambda t\, e^{-\lambda t}\, dt = \frac{1}{\lambda}$$

E(t) is then defined as the mean time to failure, MTTF.

5.1.1.2 Operation of Monte Carlo Program

The Monte Carlo program solves the probability of failure equation in reverse. It generates a random number which is evenly distributed between zero and one. It sets this equal to the probability of failure and solves for time to failure.

$$P_f(t) = 1-e^{-\lambda t}$$

$$t = -\frac{\ln\ [1-P_f(t)]}{\lambda}$$

or $$t = -\ln\ 1-P_f(t) \cdot \text{MTTF}$$

This equation is solved for each piece of equipment in the system. The λ or MTTF which is used in solving for t is a function of whether or not the equipment is turned on or off. It has been found that electronic equipment is susceptable to failure even while it is sitting idle. This idle failure rate is distributed exponentially also and the failure rate is approximately proportional to the active failure rate. For this analysis it is assumed that the idle failure rate is directly proportional to the active failure rate; so that P_f for an idle piece of equipment is:

$$P_f(t) = 1-e^{\frac{-\lambda t}{\alpha}}$$

where α is the constant of proportionality.

A block diagram of the Monte Carlo program is given in Figure 5-1, the Monte Carlo program first generates the time to failure for each piece of equipment based on its original status i.e., active or idle. The program then checks if any of the times to failure of active equipment are less than or equal to the length of the first phase of the mission. If there is a failure, another random number is generated and compared with a probability of detection of the failure. If the random number is lower than the probability of detection the failure is recorded as a detected failure. That equipment is then replaced (in the simulation) and the program continues. If the random number is higher than the probability of detection the failure is recorded as an undetected failure. Idle equipment failures are recorded when the piece of equipment is turned on either when it has replaced an active failed piece of equipment or when it is turned on at the beginning of a new phase. Once the failure is recorded the equipment is treated the same as active equipment which has failed, i.e., it is replaced and downtime is accumulated. Downtime is accumulated in two ways: by the replacement time for failed equipment and by not having any spares in a multi-equipment mode (such as Mars Orbital phase where 2 computers are required in the Multi-Computer Approach). If all the equipment has failed the mission is terminated and recorded as a failure. Downtime was also recorded for undetected failures and noted separately from the downtime identified above. An undetected failure results in a mission failure when a critical phase is entered.

If none of the times to failure are within the first phase, the program generates new times to failure for the equipment which change status (on/off) for the next phase and the program continues as before.

If when all the phases are completed and not all the equipment has failed (computer available for critical phases) the mission is recorded as a success. This process is repeated a large number of times for each configuration and statistics are accumulated which indicate probability of success and availability.

5.1.1.3 Accuracy, Confidence, and number of runs

The number of runs, N, necessary to achieve a given accuracy and confidence in the results can be found from the following equation:

$$N \geq \frac{p\,(1-p)\,K^2\,(\alpha)}{\epsilon^2}$$

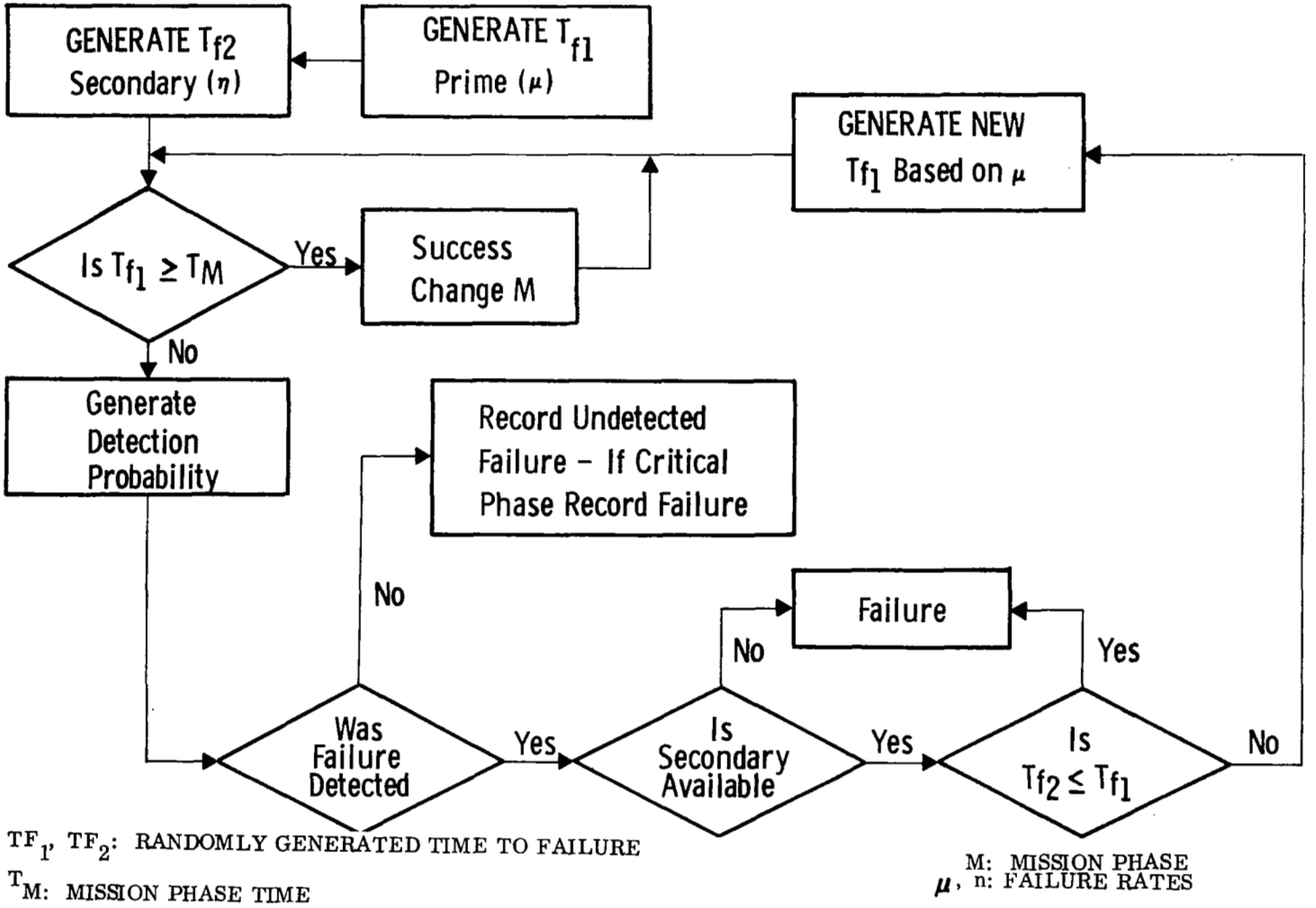


Figure 5-1. Block Diagram of Monte Carlo Simulation

where p is the probability to be determined, $K^2(\alpha)$ is a confidence function which will be discussed below and, ϵ is the allowable error.

This equation is derived below.

The runs generated by the Monte Carlo program are essentially independent Bernoulli trials. Since Bernoulli trials obey the binomial probability law, it is desired to find the statistics which describe binomially distributed probabilities (see Reference 19). In order to simplify this task without sacrificing accuracy the normal approximation to the binomial distribution is applied. This states that:

$$p\left[\frac{\left|N\left(f_n - p\right)\right|}{\sqrt{Np(1-p)}} \leq h\right] = 2\,\Phi(h) - 1$$

Where N is the number of trials

f_n is the relative frequency of success of an event with probability p of success on each trial.

h is the allowable error of $\dfrac{\left|N\left(f_n - p\right)\right|}{\sqrt{Np\,(1-p)}}$

$$\Phi(h) = \int_0^h \frac{e^{-1/2\,y^2}}{\sqrt{2\pi}}\,dy$$

This can be rewritten as

$$p\left[\frac{\left|f_n - p\right|}{\gamma} \leq h\right] = 2\,\Phi(h) - 1$$

where γ is the standard deviation of the results of Bernoulli trials

$$\gamma = \sqrt{\frac{p\,(1-p)}{N}}$$

This equation states that: the probability that the ratio of the difference between the simulated probability and the actual probability, $(f_n - P)$, and the standard deviation, $\left(\sqrt{\frac{p(1-p)}{N}}\right)$, of the Bernoulli trials is less than or equal to some constant h, is equal to twice the positive normal distribution of h minus one. If $h = \epsilon/\gamma$, where ϵ is the allowable error between the simulated and actual probabilities, the previous equation becomes:

$$p\left[\left|f_n - p\right| \leq \epsilon\right] \approx 2\,\Phi\left(\epsilon\sqrt{\frac{N}{p(1-p)}}\right) - 1$$

if K (α) is set as the solution to

$$2\,\Phi\left[K(\alpha)\right] - 1 = \int_{-K(\alpha)}^{K(\alpha)} \frac{e^{-1/2\,y^2}}{\sqrt{2\pi}}\,dy = \alpha$$

where α is the confidence level which is desired then

$$p\left[\left|f_n - p\right| \leq \epsilon\right] \geq \alpha \quad \text{if} \quad \epsilon \sqrt{\frac{N}{p(1-p)}} \geq K(\alpha)$$

which will be satisfied if

$$N \geq \frac{K^2(\alpha)}{\epsilon^2}\,p(1-p)$$

K (α) can be found from a table of the normal curve of error. If the value of 1/2 α is found in the area column the corresponding value in the t column is K (α). Note that for the worse case

$$N \geq \frac{K^2(\alpha)}{4\epsilon^2} \quad \text{when } p = 1/2$$

for p greater or less than 1/2 the number of trials (runs) decreases. This means that if nothing is known of the probability that is being simulated the worse case must be used but if the probability can first be estimated the number of runs can be reduced.

This equation can also be used inversely, that is if a number of runs have been made, an error limit can be established for a given confidence level.

$$\epsilon \leq K(\alpha)\,\gamma \quad \text{i.e.,} \quad \epsilon \leq K(\alpha)\sqrt{\frac{p(1-p)}{N}}$$

10,000 runs were used in the Monte Carlo Simulation and the error was calculated for different values of p (from 0.8 to 1.0) with different confidence values (from 0.75 to 0.95); these calculations are shown in Figure 5-2.

5.1.2 Simulation Results

5.1.2.1 Introduction

This section presents the results from the Monte Carlo reliability simulation of each of the candidate organizations. Two items are to be determined from the simulation: Mission Probability of Success and Computer System Availability. The required goal for both of these items was set at 0.997; this was the value generally used in the references cited in Paragraph 2.1. It should be noted that probability of success refers only to the computer system and not to the combined vehicle systems

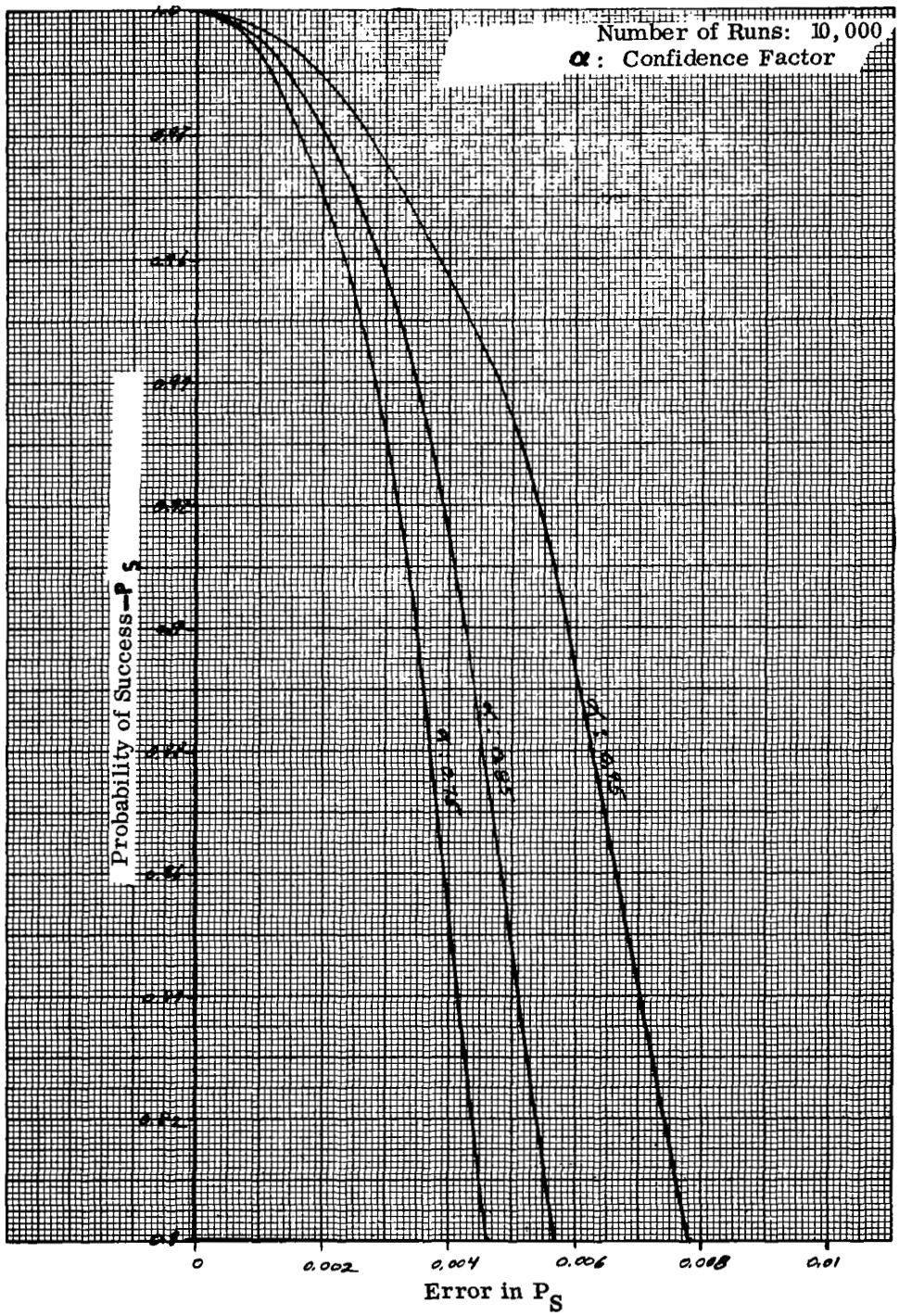


Figure 5-2. Error In Monte Carlo Simulations

probability of success. Probability of success was identified as being able to perform the computations during critical phases (phase 10 Mars Aerobraking and phase 20 Earth ReEntry). This is the appropriate parameter to consider during these phases and not availability. Time to replace a failed module was assumed to be fixed at 1/2 hour for the purpose of this simulation (in some cases a spare module is switched in electronically and time to repair is 0 for this case).

The twenty mission phases were grouped into 15 phases for the simulation, the tabulation below shows this grouping:

Mission Phases	Monte Carlo Phase
1, 2 Atmospheric Ascent Earth Orbit Injection	1
3 Trans Mars Injection	2
4 Trans Mars Coast	3
5 Trajectory Correction	4
6, 7, 8 Spin Up Spin Cruise De Spin	5
9 Mars Approach Correction	6
10 Mars Aerobraking	7
11 Mars Orbit Injection	8
12 Mars Orbital Coast	9
13 Trans Earth Injection	10
14 Trans Earth Coast	11
15 Trajectory Correction	12
16, 17, 18 Spin Up Spin Cruise De Spin	13
19 Earth Approach Correction	14
20 Earth Re-Entry	15

The module structure will be repeated here for each candidate organization, a detailed description of each was given in Section IV, Multi-Computer: one entire module, Multiprocessor: three types of modules, Input/Output, Processor, Memory, Distributed Processor: an array of identical modules.

The computer module useage as a function of mission phase is given below:

Multi-Computer

Phases 1, 2	One Computer Module (one memory section on)
Phases 3, 5, 6, 8, 9, 10, 11, 13, 15, 16, 18, 19, 20	Two Computer Modules (one memory section on in each) (one computer in active redundancy)
Phases 4, 7, 14, 17	One Computer Module (one memory section on, 2nd section on intermittently)
Phase 12	Two Computer Modules (one with one memory section on, 2nd with two memory sections on)

Multiprocessor

Phases 1, 2	One I/O Module, 1 Processor module, 1 memory module on
Phases 3, 5, 6, 8, 9, 10, 11, 13, 15, 16 18, 19, 20	Two I/O Modules, 2 Processor modules, 2 memory modules on (1 each in active redundancy)
Phases 4, 7, 14, 17	One I/O Module, 1 Processor module, 1 memory module on; 1 memory module on intermittently
Phase 12	Two I/O Modules, 2 Processor modules, 3 memory modules on

Distributed Processor

Phases 1, 2	16 groups on
Phases 3, 5, 6, 8, 9, 10, 11, 13, 15, 16, 18, 19, 20	12 groups on (6 in active redundancy)
Phases 4, 7, 14, 17	16 groups on, 3 groups on intermittently
Phase 12	24 groups on

5.1.2.2 Tabulation of Results

Below is a tabulation of the cases that were simulated and a summary of the results

Case No.	MTBF on (hrs)	MTBF off (hrs)	P_{Det}	Spares	P_S
Multicomputer					
(Spares = number of computer modules exceeding 2)					
1	8000	80000	0.99	0	0.6026
2	8000	80000	0.99	1	0.8140
3	8000	80000	0.99	2	0.9220
4	16000	160000	0.99	0	0.8366
5	16000	160000	0.99	1	0.9544
6	16000	160000	0.99	2	0.9874
7	25000	250000	0.99	0	0.9216
8	25000	250000	0.99	1	0.9822
9	25000	250000	0.99	2	0.9953
(Cases 1 through 9 assumed no on/off capability in memory section)					
11	16000	160000	0.99	0	0.8906
12	16000	160000	0.99	1	0.9749
13	16000	160000	0.99	2	0.9923
14	8000	80000	0.998	1	0.8810
15	8000	80000	1.0	1	0.8811
16	25000	250000	0.998	1	0.9921
17	25000	250000	1.0	1	0.9920
21	8000	80000	0.99	0	0.7096
22	8000	80000	0.99	1	0.8721
23	8000	80000	0.99	2	0.9524
24	25000	250000	0.99	0	0.9482

Case No.	MTBF on (hrs)	MTBF off (hrs)	P_{Det}	Spares	P_S
25	25000	250000	0.99	1	0.9900
27	8000	80000	0.75	1	0.7520
29	25000	250000	0.99	2	0.9953
30	25000	250000	0.90	1	0.9646
31	25000	250000	0.75	1	0.9278
32	8000	40000	0.99	2	0.9345
33	16000	80000	0.99	2	0.9895
34	8000	400000	0.99	2	0.9657
35	16000	800000	0.99	2	0.9926
36	25000	250000	1.0	2	0.9995
37	25000	1,000,000	0.99	2	0.9970
38	25000	25000	0.99	2	0.9845
39	16000	16000	0.99	2	0.9456
40	8000	80000	1.0	0	0.7119
41	8000	80000	1.0	2	0.9619
42	25000	250000	1.0	0	0.9529

Multiprocessor

(Spares = Number of the following sets of modules: 1 I/O Module, 1 Processor Module, and 2 memory modules exceeding the following baseline configuration 2 I/O modules, 2 Processor Modules and 4 memory modules.)

43	*	*	0.99	0	0.8227
44	*	*	0.99	1	0.9593
45	*	*	0.99	2	0.9851

*$MTBF's_{on}$: I/O 66,700 hrs

Processor 28,600 hrs

Memory 20,000 hrs

*$MTBF_{off} = 10 \times MTBF_{on}$

Case No.	MTBF on (hrs)	MTBF off (hrs)	P_{Det}	Spares	P_S
46	**	**	0.99	0	0.9717
47	**	**	0.99	1	0.9941
48	**	**	0.99	2	0.9971
49	**	**	0.75	2	0.9233
50	**	**	1.0	0	0.9746
51	**	**	1.0	1	0.9987
52	*	*	1.0	0	0.8219
53	*	*	1.0	1	0.9676
54	*	*	1.0	2	0.9949

**MTBF's $_{on}$: I/O 208,300

Processor 89,500

Memory 62,500

**MTBF $_{off}$ = 10 x MTBF $_{on}$

Distributed Processor

55	*200,000	2,000,000	0.99	1	0.9998
56	*200,000	2,000,000	0.85	3	0.9978

*GROUP MTBF

Spares = Number of groups exceeding 24

5.1.2.3 Discussion of Results

5.1.2.3.1 Multi-Computer

The results from the simulation are given in Figures 5-3 through 5-7 for the Multi-Computer candidate. Figure 5-3 shows the mission probability of success (P_S) as a function of the number of computers used, three MTBF's are shown 8,000, 16,000 and 25,000 hours. In addition, the dashed curve below each solid curve shows the effect of not having the capability of turning off part of the memory. The solid line for each MTBF assumed that the memory was divided into two sections with the capability of turning one section on and off independently of the rest of the computer; it is seen that this capability had a significant effect on the P_S. The conditions on these curves were a Probability of Detection of Failures (P_d) of 0.99 and an on/off ratio of failure rates ($\lambda_{on}/\lambda_{off}$) of 10.

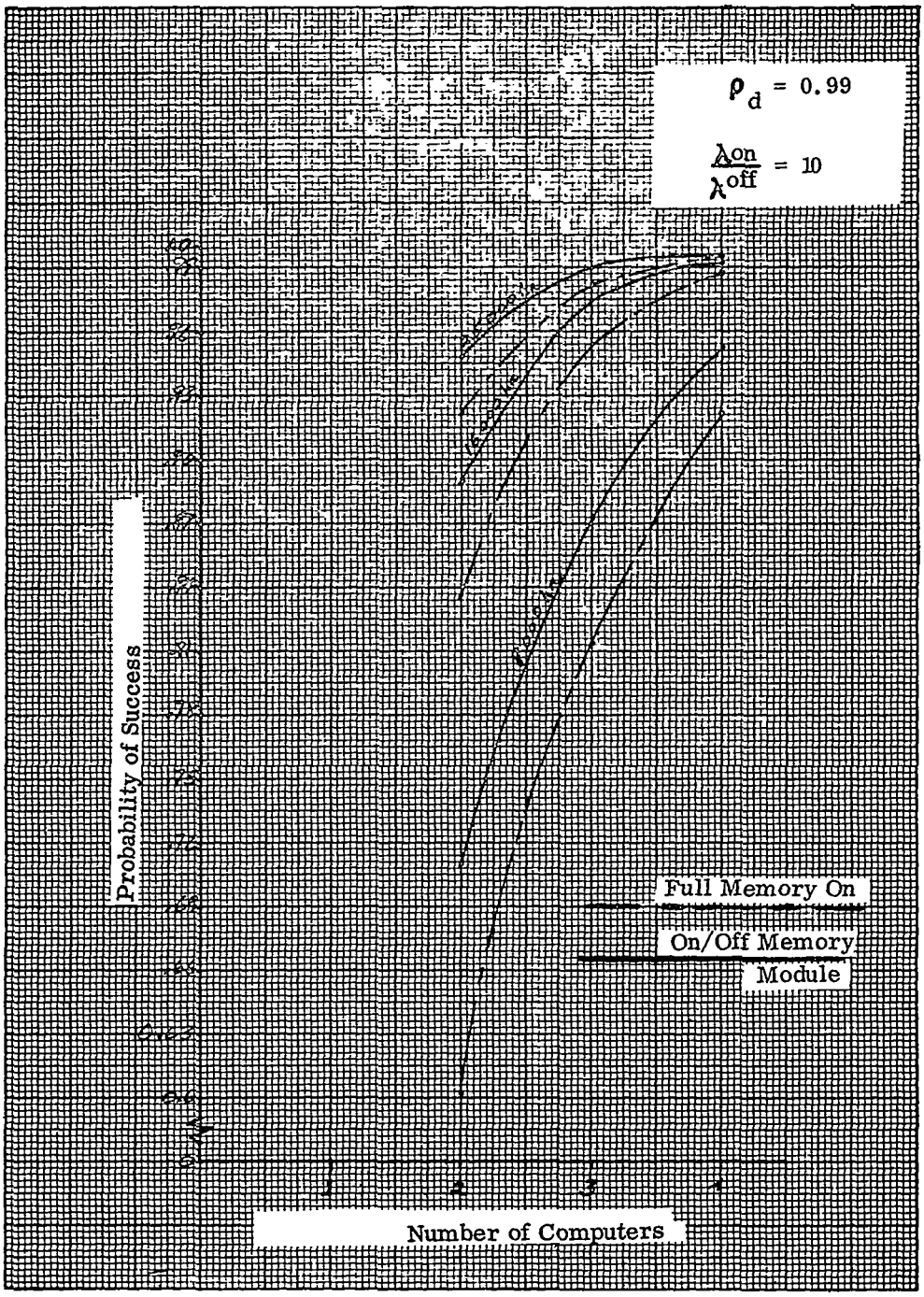


Figure 5-3. Multicomputer Probability of Success

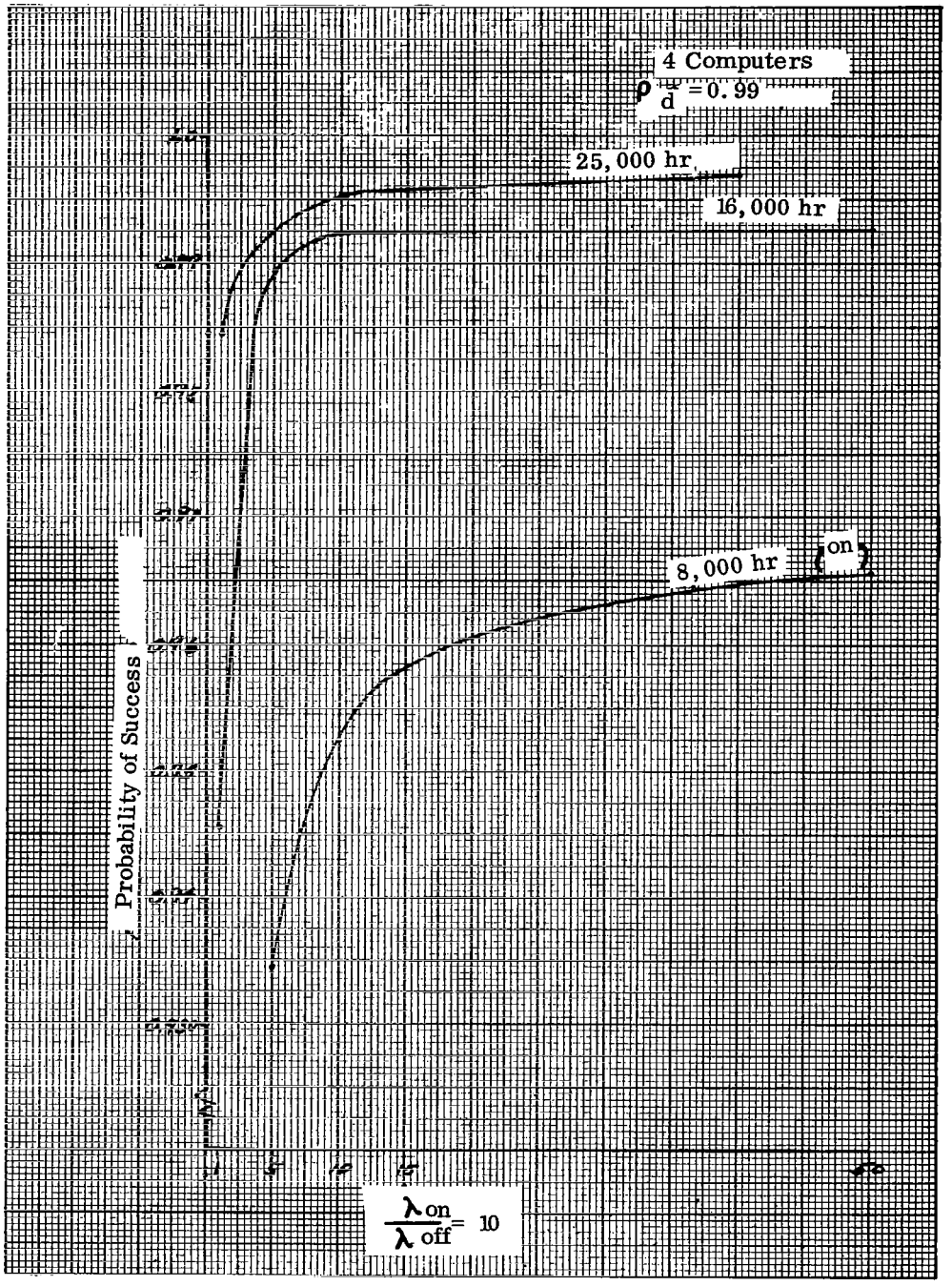


Figure 5-4. Multicomputer On-Off Failure Rate Effects on P_S

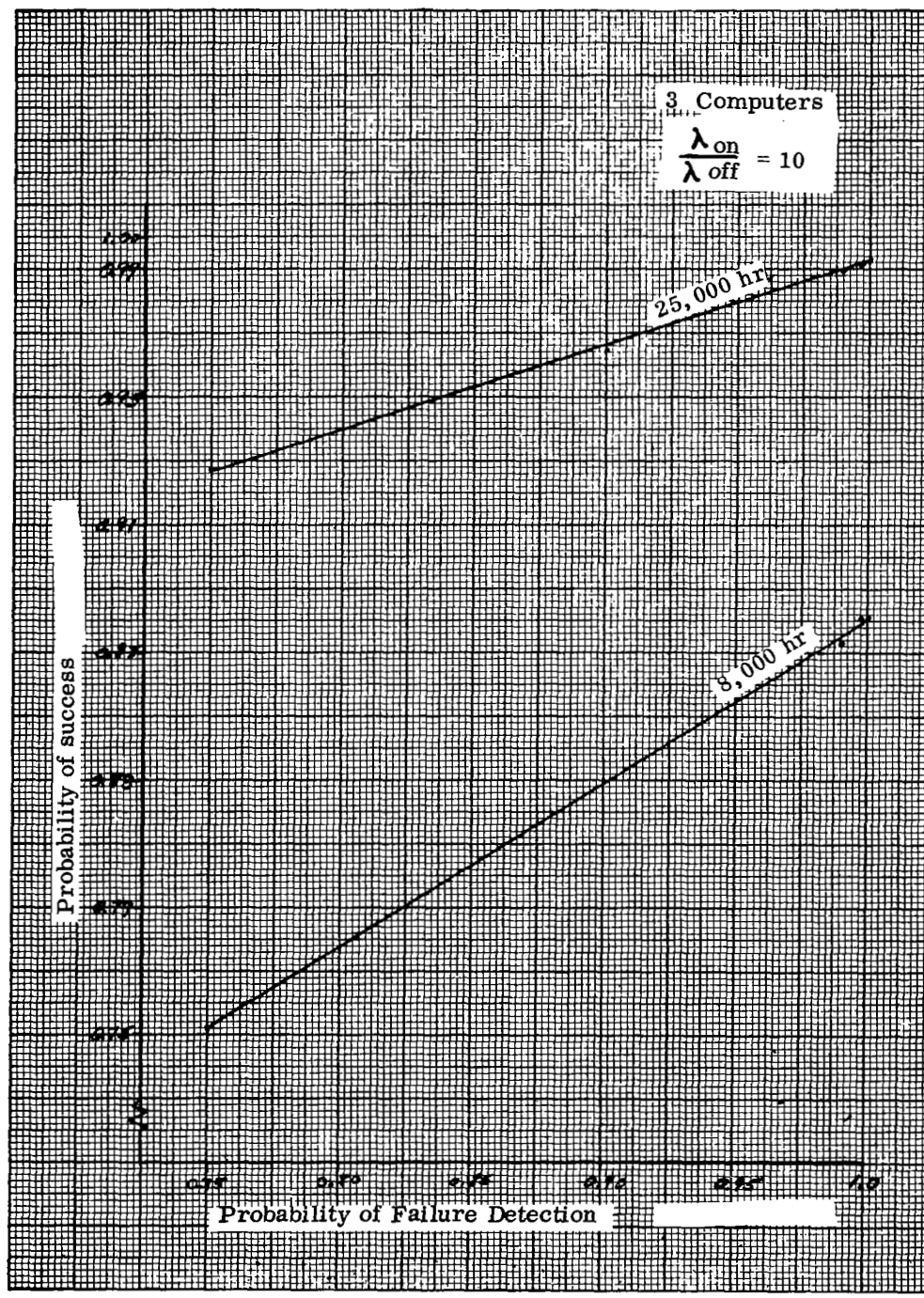


Figure 5-5. Multicomputer Failure Detection Probability Effects on P_S

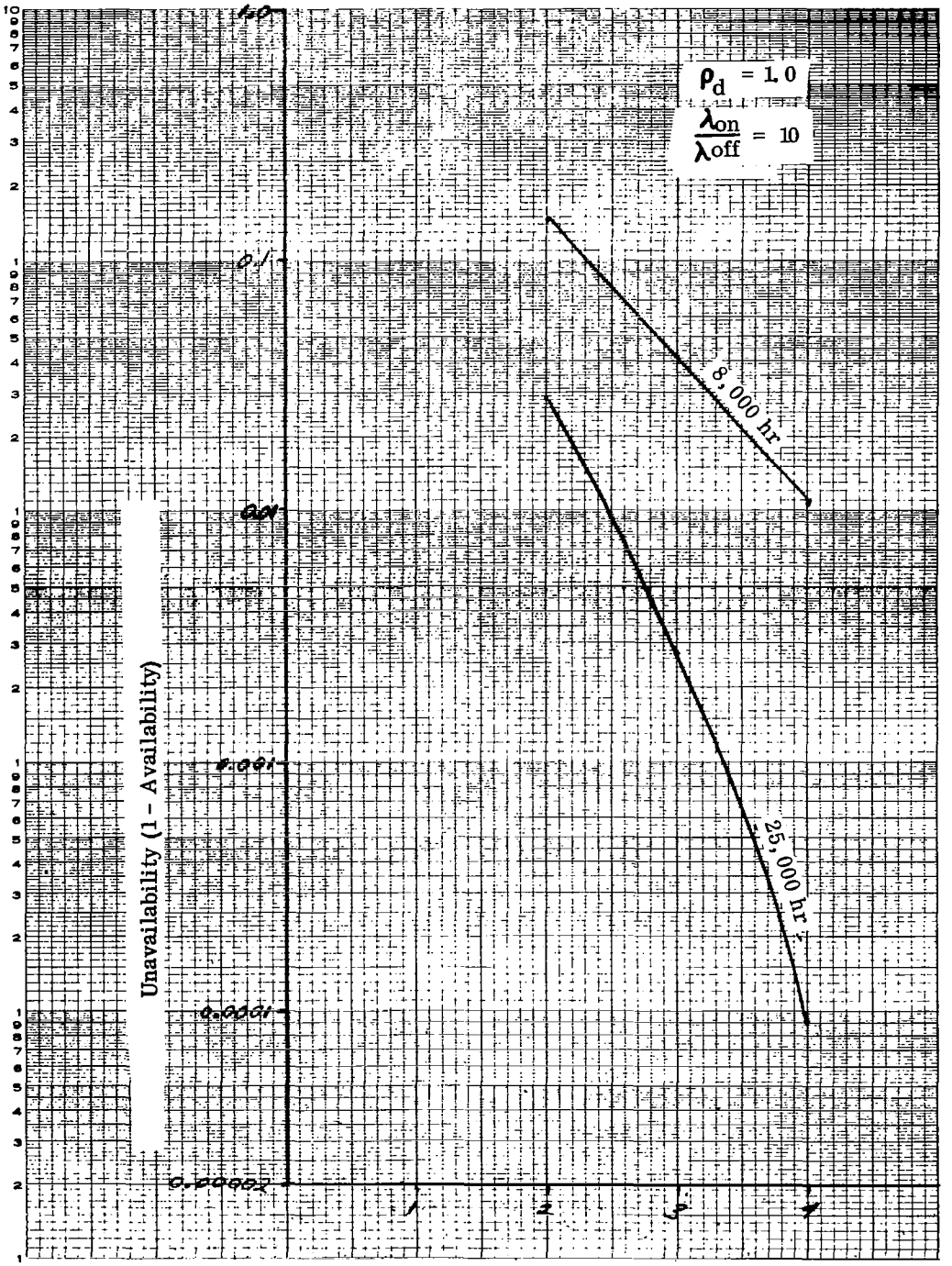


Figure 5-6. Multicomputer Unavailability

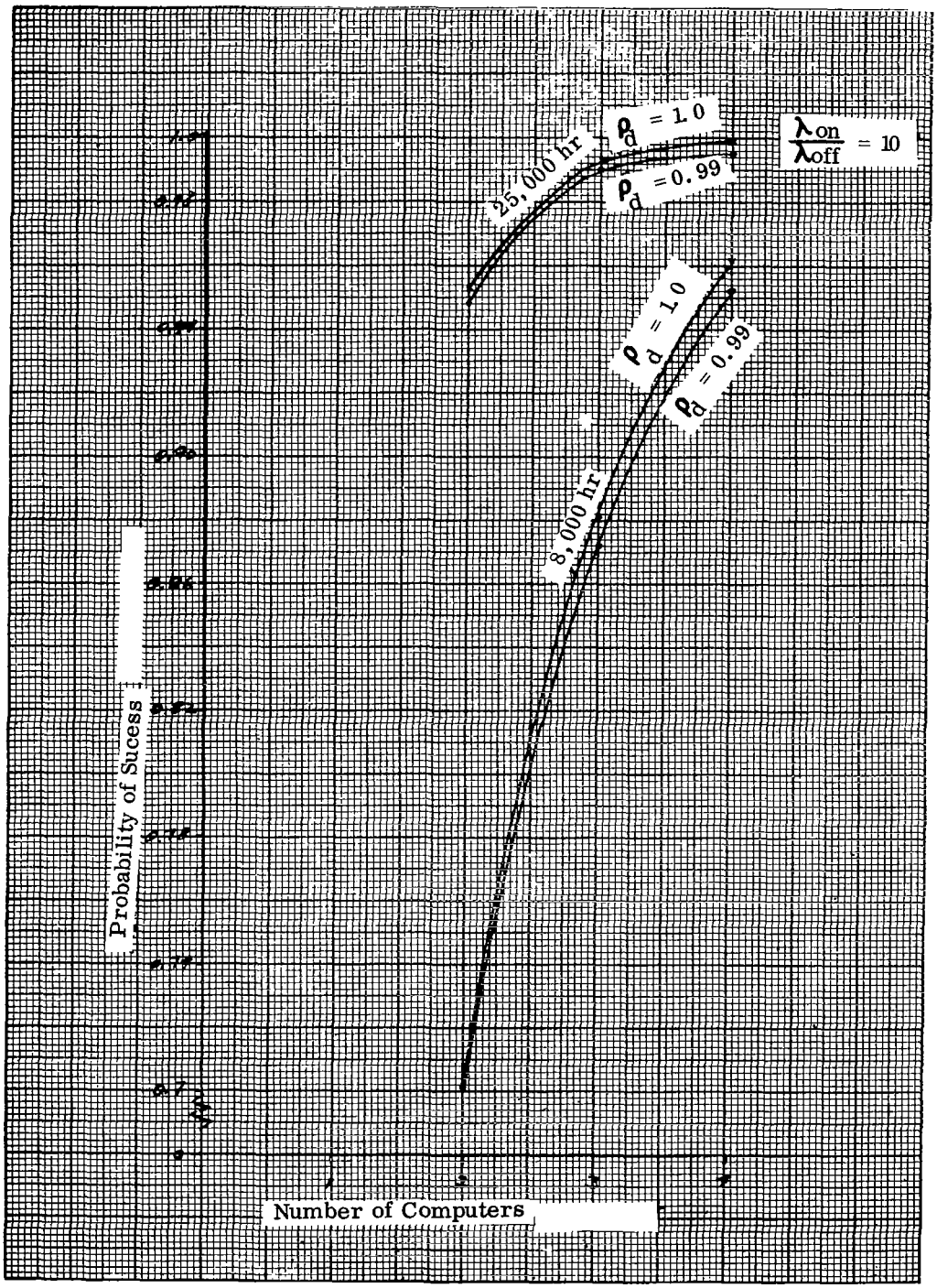


Figure 5-7. Multicomputer P_S vs Number of Computers with $P_d = 0.99$ and 1.0

The effects of on to off ratio in failure rates are shown in Figure 5-4. This curve shows that the improvements in P_s with an increasing $\lambda_{on}/\lambda_{off}$ ratio are quite significant, particularly with lower on time MTBF's. It is seen that most improvement generally has been realized in P_s by the time a ratio of 10 has been reached.

Probability of failure detection has a limiting effect on P_s as more and more spares are added. Figure 5-5 shows the effects of P_d on a system of 3 computers with a $\lambda_{on}/\lambda_{off}$ ratio of 10. It is seen that the linear region of this curve has been traversed for the P_d's considered (0.75 - 1.0). Two MTBF's were plotted 25,000 hrs and 8,000 hrs. This curve points out a significant fact: that gains in P_s as function of P_d are linear, that is, an increase in P_d from 0.95 to say 1.0 produces a linear gain in P_s, which is the same gain in increasing P_d from 0.80 to 0.85.

It was mentioned previously that Availability was the other parameter of consideration. Downtime was accumulated due to two factors: time to replace with a spare and the time that the full computing capability was not available (for example 1 computer only working during phase 12, Mars Orbital, when 2 are required). Availability is defined by: (Mission Time - Down Time) / Mission Time. Down time was very low and to visualize the large numbers for availability, unavailability was computed in its place so that a semi log plot might be constructed. Unavailability is simply Down Time/Mission Time, the results are shown in Figure 5-6 for a P_d of 1.0 and $\lambda_{on}/\lambda_{off} = 10$.

Figure 5-7 contains a comparison of P_s vs the number of computers for a $P_d = 0.99$ and $P_d = 1.0$, the limiting effect of P_d is seen as more spares are added.

5.1.2.3.2 Multi-Processor

The results of the Multi-Processor Simulations are shown in Figures 5-8, 5-9, and 5-10. It should be noted that considerably less points were obtained for this candidate, this is due to two factors: 1) the amount of computer time to simulate this system increases considerably due to its added complexity in the smaller module breakout and 2) Many of the results obtained for the Multi-Computer follow through, such as the $\lambda_{on}/\lambda_{off}$ ratio effects, and it was thought not worth while repeating them.

Figure 5-8 shows the P_s vs number of computer systems curve, a computer system is defined here as an I/O module, a Processor module and 2 Memory modules, this basically has the capability of the Multi-Computer plus some additional features as explained in Section IV. Two MTBF values were considered and the MTBF (on) for each of the modules is indicated on the curve. The group $MTBF_A$ corresponds fairly closely to the 8,000 hr Multi-Computer while the group $MTBF_B$ corresponds to the 25,000 hr Multi-Computer (it should be remembered that the Multiprocessor system exceeds the Multi-Computer capability and this is not a true comparison between the two). Again a P_d of 0.99 was used in this curve; to determine what P_s would be achieved with a P_d of 1.0 (for $MTBF_B$) a point was obtained at $P_d = 0.75$ and a straight line projected (Figure 5-9). P_s then turned out to be 1.0 which is exactly what was expected after studying the results of case 45 ($P_d = 0.99$) on the computer print out. This case showed that there were no failures during the critical phases and P_s was 1.0 prior to entering the first critical phase and did not change until the next critical phase was entered. What this meant is simply that the only failures occurring having an effect on P_s were undetected failures (an undetected failure did not amount to a mission failure until a critical phase was entered with that undetected failure).

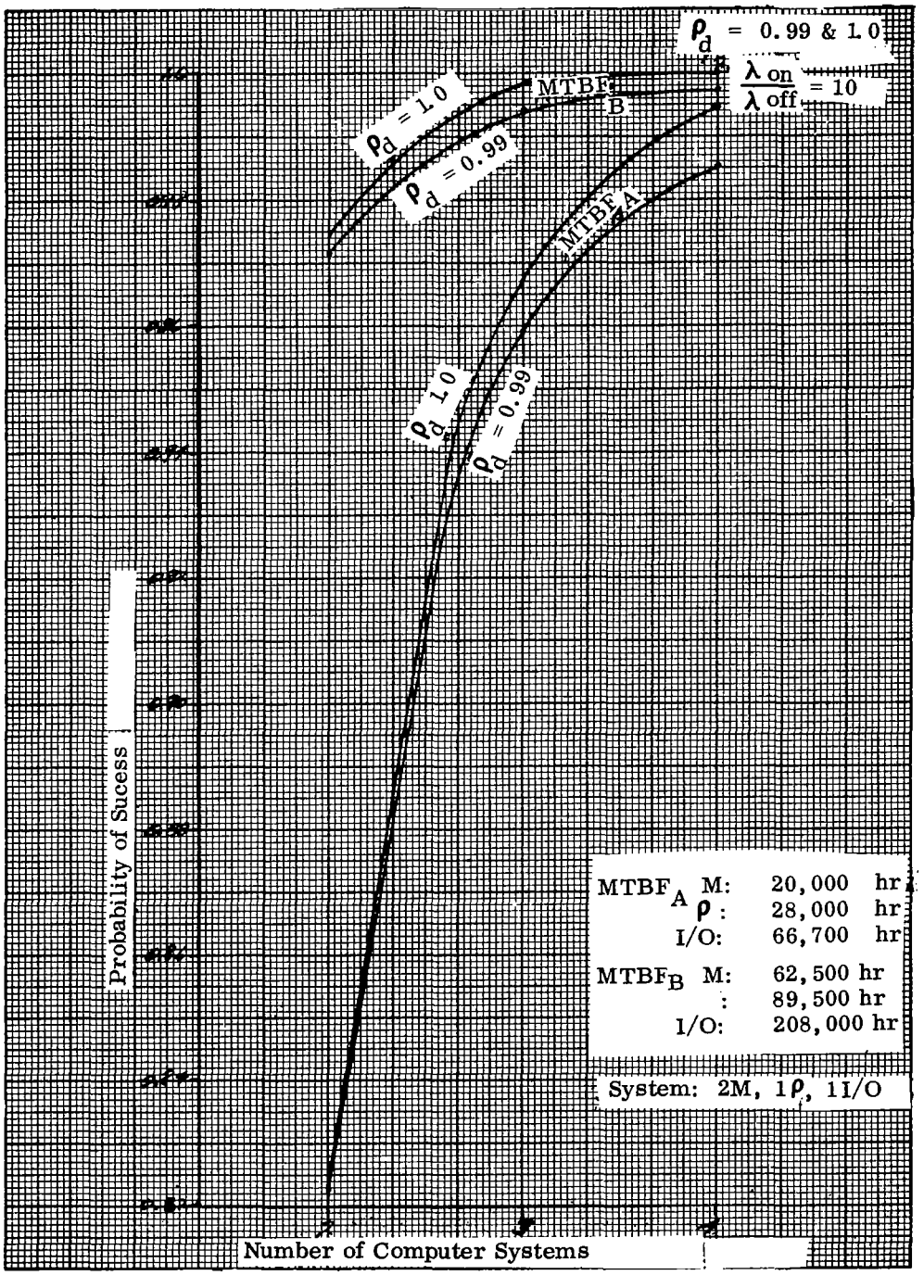


Figure 5-8. Multiprocessor Probability of Success

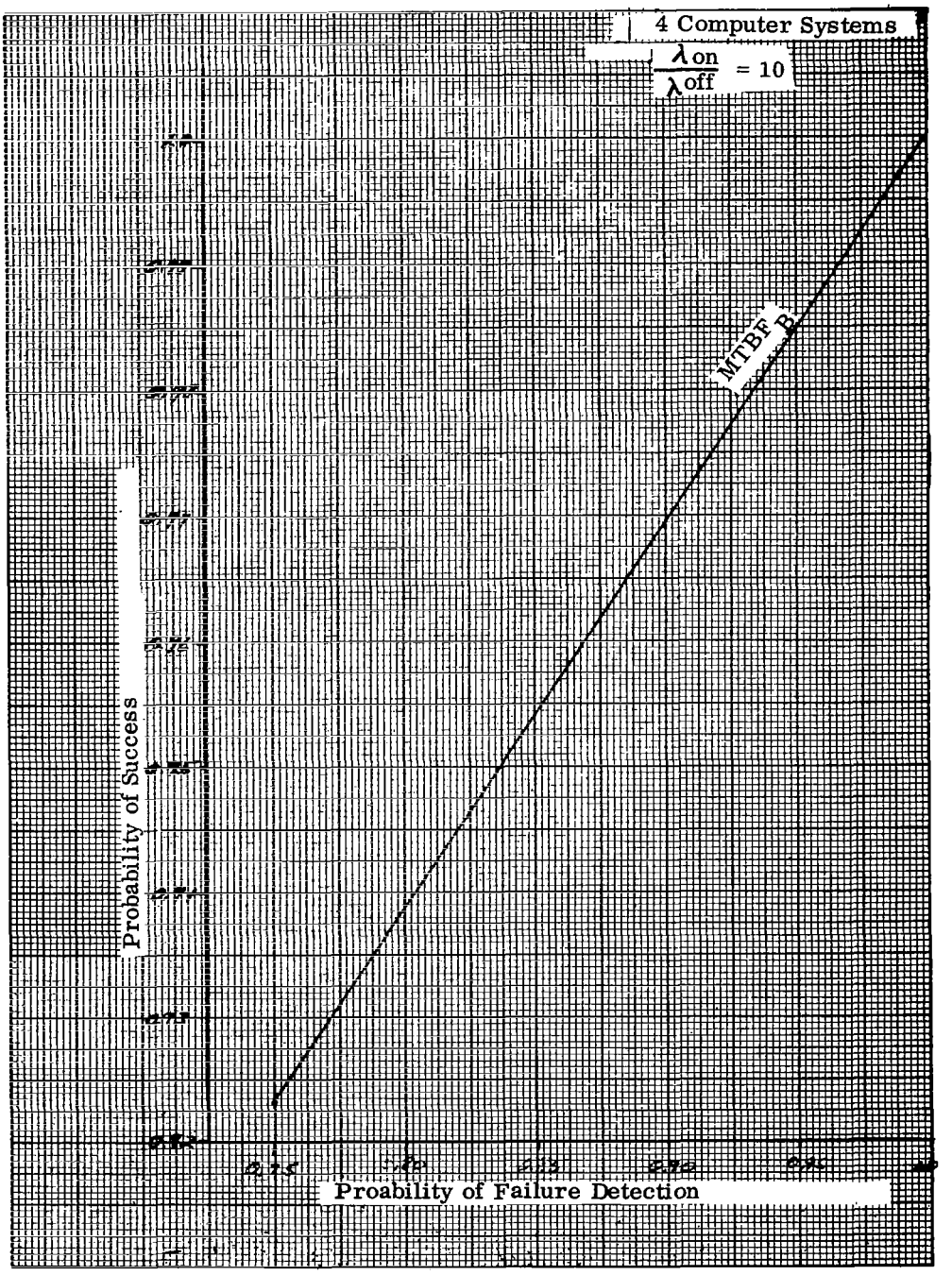


Figure 5-9. Multiprocessor Probability of Failure Detection Effects on P_s

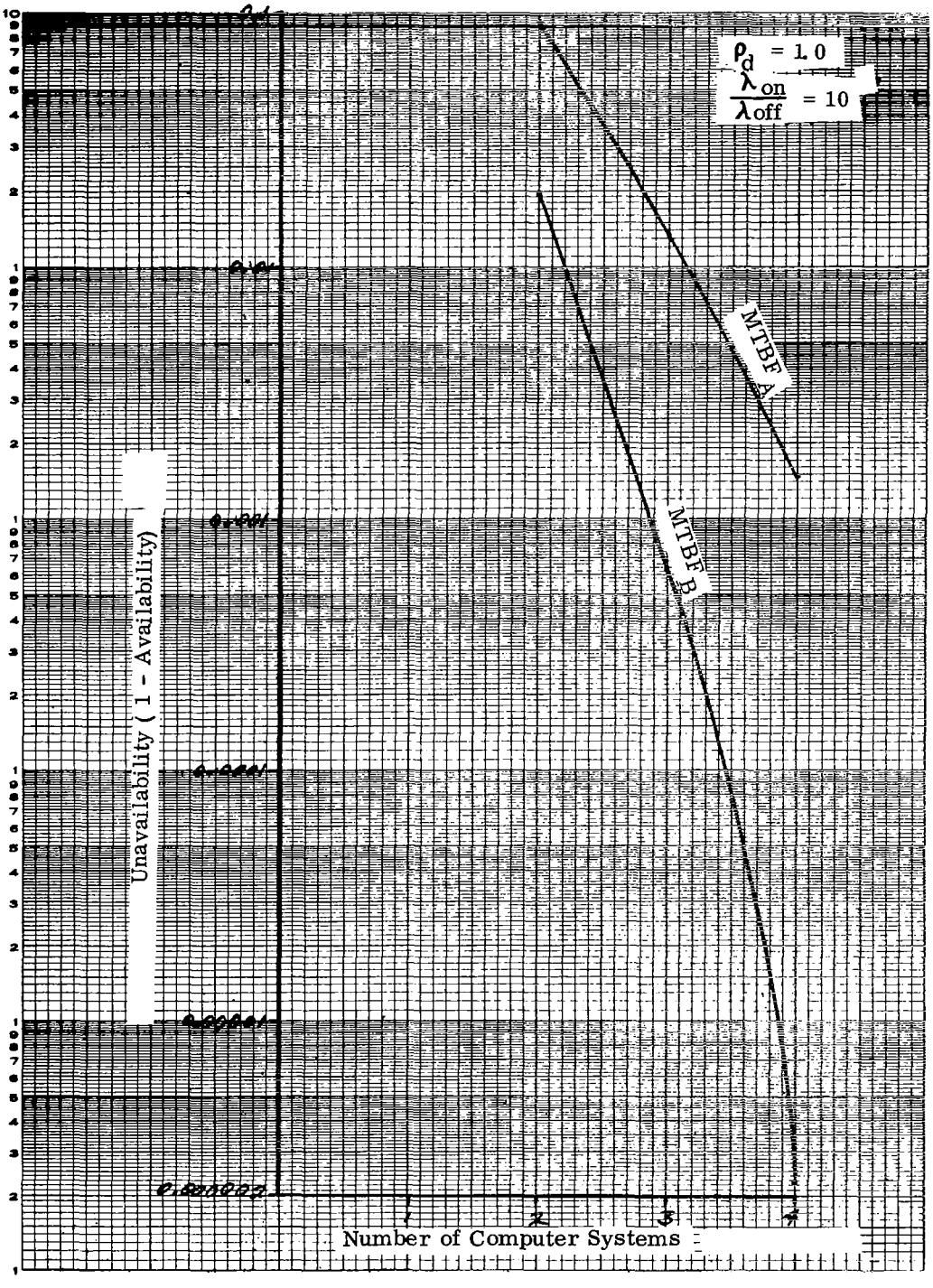


Figure 5-10. Multiprocessor Unavailability

In addition, points were later generated with a $P_d = 1.0$ for both MTBF's and also plotted on Figure 5-8. It can be seen that P_d is quite a limiting factor as a higher P_s is approached.

Unavailability is shown in Figure 5-10 for the two MTBF's, the curves are similar to the Multi-Computer except that they are considerably better in terms of a much lower Unavailability.

5.1.2.3.3 Distributed Processor

No curces are plotted for the Distributed processor organization for two reasons 1) as above the runs take considerably longer due to the Modularity involved, 2) the P_s was expected to be very high due to the sparing capability of modules left over from non-critical phases requiring large amounts of modules.

Two runs were made, each with a 200,000 hr MTBF (on) for each group, a P_d of 0.99, and $\lambda_{on}/\lambda_{off}$ of 10. The first run assumed only 1 extra group was provided as a spare, P_s was 0.99980 and deviated from 1.0 only due to undetected failures, downtime was 261.03 hours. It should be noted that the only MTBF considered was 200,000 hrs, this is considered a lower bound on the group MTBF and since P_s was met easily with this value, higher MTBF's were not simulated.

Another run was made with a P_d of 0.85 and three spare groups. This run gives two effects, the effect of P_d on P_s and the effect of spares on downtime. P_s was 0.9978 and downtime was 219 hours. Two important facts should be pointed out here: 1) the limiting factor of P_d in an organization like this and 2) a new, more refined, definition for downtime is necessary since downtime was accumulated when all 24 groups were not operating during the Mars Orbital Phase. Certainly the entire computer system is not down and a more reasonable answer is 23/24 of the system is available or some lower fraction depending on the functions being performed by the groups (since the loss of one group may preclude the use of another group, etc.).

As examples of the computer print outs associated with each case, three print outs are shown in Figures 5-11 through 5-13. It should be noted that downtime is given in two columns, the first "Replacement" is due to replacing a failed module with a spare or not having the full computing capability available as explained previously, the second, "Undetected Failure" is due to having an undetected failure and still using the computer system as though it were functioning correctly (once a critical phase is reached with an undetected failure, a mission failure is scored). Replacement downtime is the value used in computing availability. The column "Total Equipment Failures" indicates this function for the total 10,000 runs for each case. The numbers in the downtime columns are the average for each run.

5.2 CRITICAL EVALUATION AND RECOMMENDED APPROACH

5.2.1 Evaluation of Organizational Features

A summary of some of the organizational features is given below.

5.2.1.1 Multi-Computer

The principal advantages of this organization are the minimal number of components and communication lines necessary per computer, a good match of hardware

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING STUDY - MULTICOMPUTER ORGANIZATION

CASE 42 -- SYSTEM STATISTICS

Phase	NUMBER OF HRS. IN CONTINUOUS OPERATION	PROBABILITY OF SUCCESS	AVERAGE DOWNTIME PER MISSION		TOTAL EQUIPMENT FAILURES
			REPLACEMENT	UNDET. FAILURE	
1	80.2	1.00000	0.00150	0.00000	35
2	80.4	1.00000	0.00000	0.00000	0
3	200.2	1.00000	0.00185	0.00000	41
4	200.4	1.00000	0.00000	0.00000	1
5	2962.6	0.99510	4.85783	0.00000	984
6	2962.8	0.99510	0.00098	0.00000	0
7	2963.2	0.99510	0.00196	0.00000	0
8	2963.8	0.99510	0.00294	0.00000	0
9	3938.1	0.99150	129.06033	0.00000	648
10	3938.3	0.99150	0.00170	0.00000	0
11	4058.1	0.99090	1.05699	0.00000	37
12	4058.3	0.99090	0.00182	0.00000	0
13	10180.5	0.95290	158.90771	0.00000	2005
14	10180.7	0.95290	0.00942	0.00000	0
15	10180.9	0.95290	0.00942	0.00000	0

$MTBF_{on}$ = 25000 Hrs. P_d = 1.0

$MTBF_{off}$ = 250,000 hrs. 2 Computers

Figure 5-11. Monte Carlo Simulation of Spaceborne Multiprocessing Study - Multicomputer Organization

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING STUDY - MULTICOMPUTER ORGANIZATION

CASE 17 -- SYSTEM STATISTICS

NUMBER OF HRS. IN CONTINUOUS OPERATION	PROBABILITY OF SUCCESS	AVERAGE DOWNTIME PER MISSION		TOTAL EQUIPMENT FAILURES
		REPLACEMENT	UNDET.FAILURE	
80.2	1.00000	0.00130	0.00000	32
80.4	1.00000	0.00000	0.00000	0
200.2	1.00000	0.00170	0.00000	44
200.4	1.00000	0.00000	0.00000	0
2962.6	0.99980	0.08696	0.00000	1079
2962.8	0.99980	0.00004	0.00000	0
2963.2	0.99980	0.00008	0.00000	0
2963.8	0.99980	0.00012	0.00000	1
3938.1	0.99960	8.97428	0.00000	687
3938.3	0.99960	0.00008	0.00000	0
4058.1	0.99960	0.04912	0.00000	34
4058.3	0.99960	0.00008	0.00000	0
10180.5	0.99200	18.41161	0.00000	2292
10180.7	0.99200	0.00160	0.00000	0
10180.9	0.99200	0.00160	0.00000	0

$MTBF_{on}$ = 25000 hrs
$MTBF_{off}$ = 250,000 hrs

P_d = 1.0
3 Computers

Figure 5-12. Monte Carlo Simulation of Spaceborne Multiprocessing Study - Multicomputer Organization

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING STUDY - MULTICOMPUTER ORGANIZATION

CASE 36 -- SYSTEM STATISTICS

NUMBER OF HRS. IN CONTINUOUS OPERATION	PROBABILITY OF SUCCESS	AVERAGE DOWNTIME PER MISSION		TOTAL EQUIPMENT FAILURES
		REPLACEMENT	UNDET.FAILURE	
80.2	1.00000	0.00130	0.00000	34
80.4	1.00000	0.00000	0.00000	0
200.2	1.00000	0.00205	0.00000	60
200.4	1.00000	0.00000	0.00000	1
2962.6	1.00000	0.04280	0.00000	1161
2962.8	1.00000	0.00000	0.00000	0
2963.2	1.00000	0.00000	0.00000	1
2963.8	1.00000	0.00000	0.00000	3
3938.1	1.00000	0.02040	0.00000	795
3938.3	1.00000	0.00000	0.00000	0
4056.1	1.00000	0.00205	0.00000	57
4058.3	1.00000	0.00000	0.00000	0
10180.5	0.99950	0.57407	0.00000	2587
10180.7	0.99950	0.00010	0.00000	0
10180.9	0.99950	0.00010	0.00000	1

$MTBF_{on}$ = 25000 hrs
$MTBF_{off}$ = 250,000 hrs

P_d = 1.0
4 Computers

Figure 5-13. Monte Carlo Simulation of Spaceborne Multiprocessing Study - Multicomputer Organization

to requirements, and relatively simple failure detection to a module. The good match of hardware to requirements results in an efficient use of the hardware, and simple failure detection to a replaceable module which in this organization is an entire computer. The executive program for this approach is relatively straight forwarded since each computer operates essentially as a separate unit on all programming tasks.

Some disadvantages with this organization are adaptability to a change in requirements, the relatively large module size in meeting reliability requirements, and difficulty in attempting to detect failures to a lower level than a computer. The problems in meeting additional computer requirements are a severe disadvantage with this organization. If there is a need for increased computational capability, another complete computer must be added, as a result, a relatively small change in the requirements can cause a large change in the hardware in the computer system. Another problem area apparent in the Manned Mars Mission due to this limited flexibility is in the computer system for the Lander Vehicle. The computer requirements are expected to differ substantially between the Lander and the Orbiting vehciles and as a result considerable inefficiency is expected with an approach such as this.

The module is an entire computer in this approach and to meet reliability requirements by adding spares means that relatively large spare modules must be added. To detect failures to a lower level is difficult with this organization if it is attempted to make the modules smaller.

5.2.1.2 Multiprocessor

The main advantages with this organizational approach are flexibility in terms of expansion to a change in requirements, possibility of withstanding multiple failures, localization of failures to modules due to full intercommunication providing the ability to make good use of spare equipment, less down time due to replacement, and a good match to the requirements.

One of the most important advantages of the Multi Processor is its ability to expand (or contract) in relatively small increments to meet changes in computational requirements. For example in the Manned Mars Mission considered, if in the Mars orbital phase 350,000 operations per second were required in each of the two processors, this organization would be able to meet this requirement by adding an extra Processor module to the system. A Multiple Computer organization would have to add a complete computer to meet these requirements. This change means more power and less reliability for the Multiple Computer system; whereas the power and reliability would only change slightly for the Multi Processor system. Changes of this type may be required between missions as well as between different classes of missions such as Mars Landing vs Mars Fly by. In addition, the requirements for the Lander vehicle of the manned Mars mission may impose these types of changes in the requirements. It is obvious from the above that the Multi Processor also has the capability of providing a relatively close match to the requirements. The relatively small module size also offers the possibility of turning modules on and off between mission phases as computational requirements change with the resultant reliability gains over a multi-computer or single conventional computers as shown in the simulations.

Another important feature of the Multiprocessor is that due to the relatively smaller size of the modules and increased intercommunication capability it is considerably simpler to isolate failures to the module level. This also provides the

ability to withstand certain multiple failures. For example in a system containing three memories, two processors, and two I/O modules, it is possible to have any two memories fail, any one processor and any one I/O unit fail and still construct a working system. This working system could carry out the critical computations of a critical mission phase. It can also be seen that after a module fails it may be replaced while the other module takes over a part of its task. This offers the potential of having less system downtime and hence increasing computer system availability.

In terms of disadvantages the Multiprocessor has some problems none of which are severe. The most significant item is that the number of lines for communication between the modules increases considerably as the number of modules in the system increases. This presents some packaging problems and may reduce reliability due to the extra connections. Another problem is that expansions in terms of modules are limited and must be anticipated in the design phase. Finally the software is more complex than that for the Multi-Computer, however, this increase in complexity is quite small.

5.2.1.3 Distributed Processor

The Distributed Processor organization offers the following advantages: a wide adaptability to changes in computational requirements due to the capability for expansion in terms of quite small hardware increments, the possibility of "graceful degradation," a good use of the MOS/SOS technology, and the possibility for high reliability and low power due to no main memory in the organization.

A very important feature of this organization is its adaptability to changes in requirements. Additional groups may be added to the organization as required without any redesign of the system. Likewise groups may be deleted from the system as they fail, of course all of the computational requirements may not be met as these failed groups are deleted. However, the reduction in capability may only result in the elimination of a small portion of the computational task thereby resulting in "graceful degradation" due to failures. This organization also makes good use of the technology in that an entire cell is mechanized on one MOS/SOS chip and it is this use of the technology which eliminates the need for a main memory thereby offering significant gains in power and reliability. It should also be noted that reliability requirements can be met relatively easily since the module increments are quite small. In addition, these spare modules will generally be located in the same physical package as the primary modules and therefore offer the possibility of a completely sealed package in which failed modules are replaced by electronic switching. This will also have a significant improvement for system availability since repair time is eliminated. Finally the capability of turning modules on and off has a significant gain in reliability particularly in light of the very close match to the varying requirements that can be realized.

These advantages are not realized without some problems, some of which are: relatively more complex software and expansion capability must be accounted for in the design. The software is considerably more complex for this organization as explained in Section IV. Some of the factors contributing to this complexity are the increased executive functions due to group interactions and control, and the selection of optimum MACRO's and MICRO's.

5.2.2 Critical Evaluation

The computer configuration required for each organization was chosen from the results of Paragraph 5.1.3 to achieve a 0.997 probability of mission success and a 0.997 availability. It should be noted that P_S was weighted as 100 in terms of relative importance as compared to the other factors weighted below, this resulted in using P_S as a design criteria due to its heavy weighting. This resulted in the following configurations:

Multi-Computer

4 Computers (25,000 hr MTBF)	(2 spare computers over those actually required)

Multiprocessor

3 I/O Modules (208,000 hr MTBF) 3 Processor Modules (89,500 hr MTBF) 6 Memory Modules (62,500 hr MTBF)	(1 spare I/O and processor and 2 spare memory modules over those actually required)

Distributed Processor

27 Groups	(3 spare groups over those actually required)

Some of the organizations actually have a probability of success greater than that required and this could also be added in to the evaluation; however, all that was considered here was that the reliability requirements were satisfied.

The actual criteria of importance for the Manned Mars mission which were used to measure the utility or effectiveness of the various computer organizations are given below along with their relative importance or ranking.

1.	Power	10
2.	Volume	1
3.	Weight	1
4.	Cost	1
5.	Development Risk	1
6.	Growth Potential	4

The first four objectives of the system are self explanatory, in general, it is desired to minimize them. Development Risk is defined as the probability of meeting the development schedule with a fixed design and within a stated budget. This is a factor to consider when choosing amoung subsystems with advanced state-of-the-art concepts or hardware, or with complex hardware.

Growth potential is also an important objective, since it is not always possible in advance to predict new or improved sensors. In addition, this potential tends to offset development risk, since it allows the possibility of alternate development schemes. This was defined as the expected modification costs for adding additional hardware to the system.

Using mathematical evaluation techniques, the three candidates were evaluated. The techniques and the actual mathematical evaluation are contained in detail in Reference 17, only a summary will be given here.

The characteristics of the candidates used in the evaluation included: Power, Volume, Weight, Cost, Development Risk, Growth Potential, and Flexibility. These characteristics were related through a matrix to the system objectives. When numerical values are inserted in the matrix, an expression for the incremental value or worth of a candidate is obtained. The results were:

Distributed Processor:	$\Delta V = +0.0366$
Multi-Processor:	$\Delta V = -0.0027$
Multi-Computer:	$\Delta V = -0.0326$

ΔV is the incremental increase in value of the actual candidate compared to the average candidate.

It is thus seen that the distributed processor candidate has the greatest positive ΔV. The actual numerical values used in the matrix were in anticipation of technology for 1980 time period missions. For current technology the multi-processor would result in the greatest ΔV, since the risk and cost associated with the distributed processor would be very high. After reviewing the values used in the evaluation, it was decided to be more conservative and question the distributed processor technology availability for these missions. Therefore, the multi-processor candidate was selected for further study. It was also felt that this technology would undoubtedly be available for these missions, in addition to being potentially available for earlier applications.

VI. DETAILED DESIGN OF THE MODULAR MULTIPROCESSOR ORGANIZATION

The multiprocessor organization was presented and functionally described in Section IV. A block diagram of the organization is shown in Figure 6-1. The multiprocessor consists of two processor modules, three memory modules, and three input/output modules. These modules satisfy the mission computational requirements. Expandibility of one more of each type of module is provided as indicated by the dotted lines in the figure. The organization features full intercommunication between modules as described in Section IV (any I/O module may communicate with any memory module and any processor module may communicate with any memory module). This section of the report will provide a more explicit specification of the contents and operation of each moudle in the system along with a presentation of the system software and fault and error control methods.

6.1 MODULES

6.1.1 Processor

There are two processors in the system for the Mars Lander Mission although the capability to expand to three processors is included. The processors operate on two's complement fixed or floating point operands. They use a 500 nanosecond clock with four clock times (bit times) per memory cycle which gives a capability of 250,000 short operations per second. A condensed block diagram of the processor module is given in Figure 6-2.

The processor features, such as instruction format, index/banking schemes, etc., were introduced in Section IV and selected on the basis of programming evaluations discussed in that section. A summary of these features will be given here prior to introducing the processor details.

The processor instruction word format is shown in Figure 6-3. The first 6 bits of the instruction are used for the operation code. Operation code extension for instructions that do not require full addresses give the facility for many more than 64 instructions. The instruction uses a banking scheme so that it is only necessary to have an address decrement in the instruction word. The banking scheme will use full length registers thereby reducing banking problems.

One bit, I, is used for indirect addressing. The format used for the indirectly addressed word provides the facility for multiple level indirect addressing and indexing.

Index/banking is accomplished by the B bit and the T bits (3) of the instruction word. The B bit is used to specify one of two full length registers and the T bits none or one of seven other full length registers to be used for index/blanking. It is important to note that there is no real distinction between bank registers and index registers since they are both full length (18 bits). Any of the registers can be added to the address decrement to generate a full length address or certain combinations of two registers can be added together and added to the address decrement to generate a full length address. A number of advantages with this index/banking scheme were given in Section IV.

Two upper accumulators are used. Full arithmetic capability is provided between the accumulators and between the accumulators and the index/bank registers.

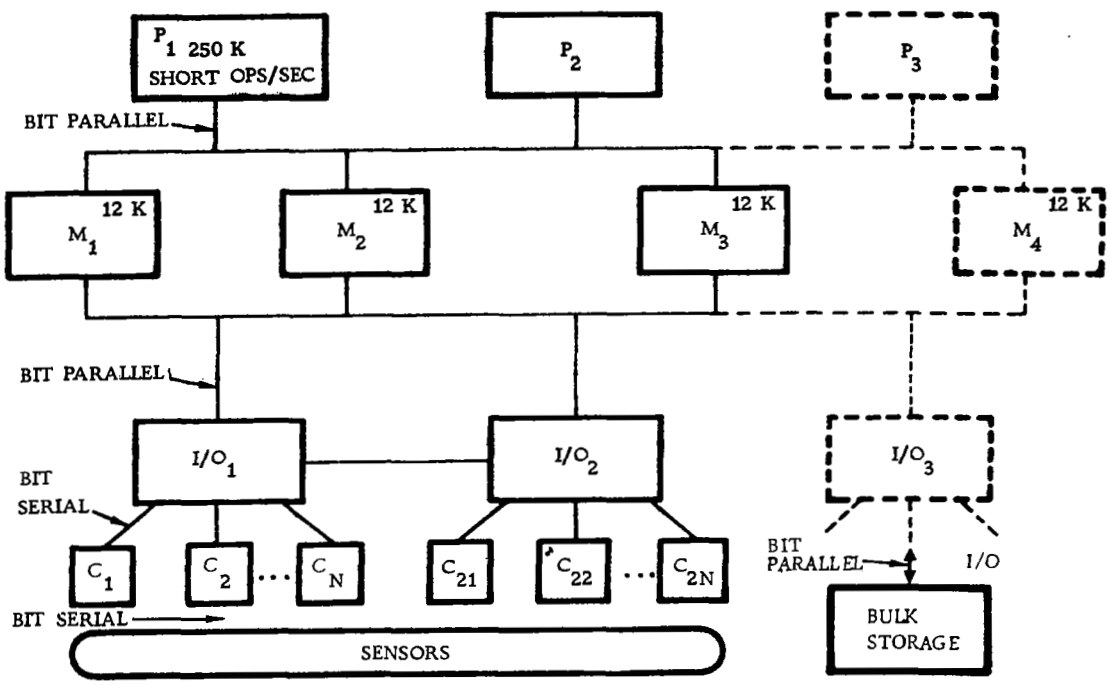


Figure 6-1. Multiprocessor Organization

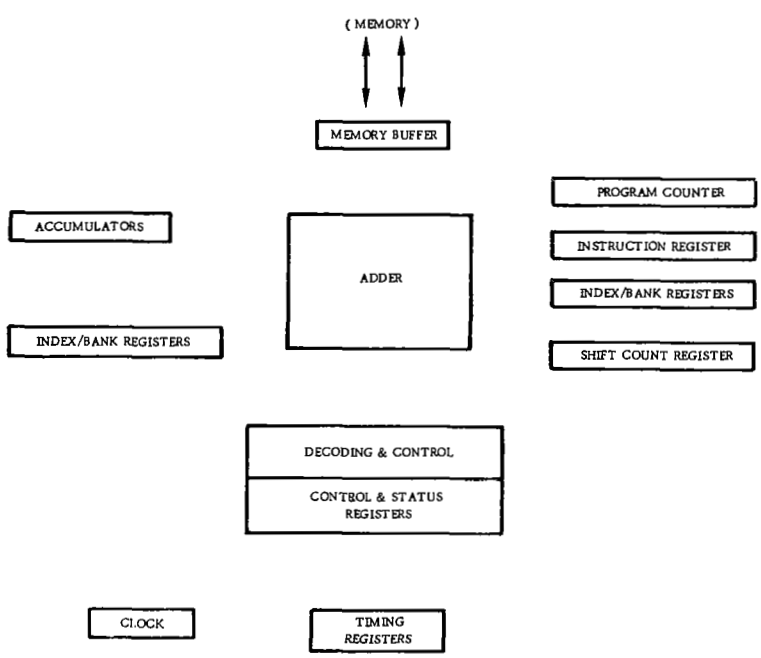


Figure 6-2. Processor Block Diagram

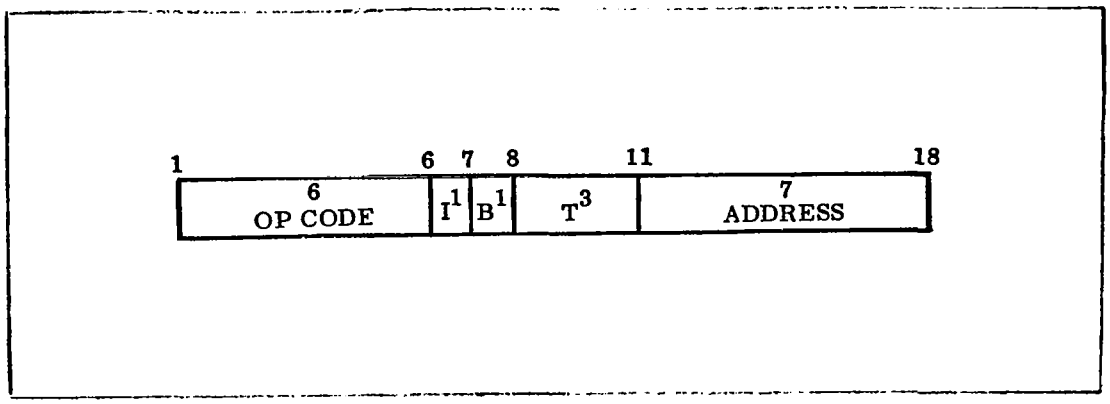


Figure 6-3. Instruction Word Format

6.1.1.1 Real Time Clock and Interrupt Features

6.1.1.1.1 Clock

A real time clock as shown in Figure 6-4 will be used in each processor. The lower 25 bits of this clock are hardware registers, while the upper 18 bits, or more if desired, are in the memory. The hardware portion is divided into two sections, an 18 bit clock register (RTC) and a 7 bit clock extension register (EXT). The RTC register can be set and read by processor instructions. Once it is set it will count down to zero and send out an interrupt. The Ext register can not be read or set, but it can be initialized to zero in order to setup precise timing at the beginning of a computation phase. Unlike the RTC register this clock and the memory clock count up.

It should be remembered from discussions in 4.2 of the executive program scheduler that the real time clock is used to interrupt a processor whenever it is time to initiate the highest rate periodic program; therefore the chosen approach for the clock is to set the RTC for the time closest below the highest rate program's period. When the clock counts down to zero it will interrupt the executive in order to notify the scheduler that the specified period is up. The executive will then carry out its tasks and then waste any remaining time until it is precisely time to start the program (within 2 μs). Just before giving control to the program it will again set the RTC to the proper period. Note that this method of operation gives the ability to time a period down to 2 μs (one instruction) even though the least significant bit of the RTC register has a value greater than 2 μs. This fine of precision on the periodic program rates is not necessary if it is possible to specify that the chosen rate for a sensor should be a multiple of 64 μs, for example; however it does give the increased flexibility of operating with any sensor that may have been setup to operate at almost any specific rate (a multiple of 2 μs). Therefore, a non-setable clock, although requiring no executive action for resetting, would not have the full flexibility of the above clock. The only penalty actually payed for this clock scheme is that one must

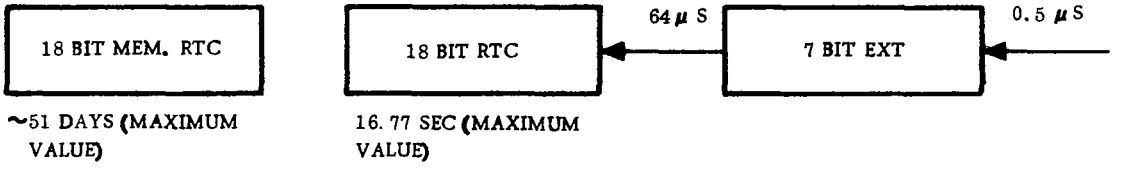


Figure 6-4. Real Time Clock

in each phase of computation keep track of the time base of the memory real time clock. This is because it is incremented each period by the executive just prior to giving control to the highest rate periodic program. This is a small penalty.

The time weighting of the RTC register depends on the variation from phase to phase in the rates of the highest rate programs. This variation has not been accurately established at this time; however a reasonable estimate for the RTC register is that shown in Figure 6-4. This weighting was chosen to allow for fairly low frequency periodic programs, up to 16.77 seconds, while not being forced to waste too much time in order to handle higher frequency programs. A maximum of 64 μs may have to be wasted, but the majority of this time is actually well spent in storing the registers of the interrupted program.

It should also be noted that the RTC interrupt should have a high priority so that accurate length periods can be maintained. If a long wait occurs after the zero interrupt, greater than 64 μs, a pulse or two could be missed causing the period of a program to drift. This is not tolerable if it continues for any length of time. Therefore making the RTC of high enough priority that it will be honored within 64 μs eliminates any problems.

A list of instructions for the RTC register is given below:

1. SRC

(set real time clock)	(M) ⟶ RTC 4 μs	*The executive can of course set or reset the memory real time clock by normal instructions accessing the memory since it keeps track of the memory clock location. (M) is the contents of memory location M.

2. RRC

(read real time clock)	RTC ⟶ U 2 μs	*This enables reading of the RTC whenever desired. The memory RTC can be read by the executive with normal instructions; however the RTC Ext is not readable. U is the upper or lower accumulator.

The programmer will also have the ability through the LPR (load processor registers) instruction to reset the RTC and the RTC extension registers (0 ⟶ RTC, Ext). This will be done at the start of a mission phase when it is desired to synchronize the real time clock with the I/O and with mission time.

The clock will run continuously while the processor is operating; therefore starting the clock is not necessary though initializing it (resetting) is necessary. The interrupt mask register will be able to inhibit the RTC from receiving pulses from the RTC Ext so that the clock can be ignored or halted whenever desired.

A fill clock will be used with the present executive scheduler since this scheduler will operate in a more predictable manner if each periodic program is of a fixed length. Branches within a program may leave some free time at the end of some executions of the program. Rather than wasting this time by waiting until a fixed time is over the executive can check to see how much time is left. If it is over some Δt, a fill clock can be set with the time left (t_1), and background programs can be executed for this period. The specification of the Δt depends on the amount of time (overhead) necessary to get into and out of a background program. 128 μs would seem to be a reasonable Δt, but this number can be varied. The maximum t_1 necessary depends on the amounts of time left over by various program branches. It is clearly very difficult to explicitly specify, but a few milliseconds seems reasonable. If more time is necessary it will simply be necessary to reset the fill clock for another period. Therefore with a 128 μs Δt, 32 ms should easily provide a sufficiently long t_1. This specifies an 8 bit fill clock that gets its least significant pulse from the RTC extension in the same manner as the RTC. Note also that 8 bits means that it can be simply loaded from the decrement of an instruction. The fill clock is set and then counts down to zero and interrupts the processor. It must also be preserved if an interrupt occurs. There seems to be no reason to read or halt the fill clock with the present executive structure.

3. SFC, B

(set fill clock)	m ⟶ FC if B = 0 U_{10-17} ⟶ FC if B = 1 2 μs	*This provides for a fixed load of the FC from the decrement or a load from one of the accumulators following a subtraction and comparison. m is the address decrement in the instruction word.

6.1.1.1.2 Interrupts

There are only three actual processor interrupts presently planned for the system, a memory interrupt, a real time clock interrupt, and a fill clock interrupt. The rest of the situations that might generally generate interrupts are handled by a request processor program that periodically scans an I/O status word in the I/O units. This approach was chosen since the I/O rates are not high enough to warrant

a more hardware oriented system. The system also includes a two-bit interrupt mask register so that the real time clock or fill clock interrupts can be masked by the executive. The I/O unit can also be interrupted, but this situation will be discussed in 6.1.3 with the I/O unit presentation.

The highest priority interrupt is the RTC zero transition interrupt. On the occurence of this signal a flip flop is set, and the executive will be entered after the present instruction is completed. This signal notifies the executive that it is time to setup execution of the highest rate periodic program and to update the memory real time clock.

The second highest priority interrupt is the "no response" interrupt. This interrupt sets a flip flop in the processor and notifies the executive that the processor has not received a response from the addressed memory. This will be caused by a one shot noting a failure of the memory to respond to a request within 14 μs or by a lockout being on to this processor. Fourteen μs was chosen for the no response failure time to allow for two other processors to receive two memory cycles appiece (IO instruction) and three I/O units to receive memory cycles. This is a worst-case situation with the maximum number of modules in the system. The executive will take over after the interrupt and check to see what caused the no response condition. If a failure has occured the failure status word will be updated if necessary and the appropriate reconfiguration operations initiated. If a processor is simply correctly locked out of a memory, a new program will be scheduled.

The third interrupt, of lowest priority, is the fill clock zero transition interrupt. It notifies the executive that a time gap due to a branch in a program has been filled with background. The executive will then either call the next periodic program or will reload the fill clock if the initial time gap was greater than the 32 ms capacity of the clock. A flip-flop is also set on the occurence of this interrupt and the executive will again be entered after the present instruction is completed.

When any of the above interrupts occur, a status word for the interrupted program must be stored in the memory. The status word will be automatically stored in four sequential memory locations, regardless of the type of interrupt, specified by a hard-wired address in each processor and the processor's primary memory register (a two bit register in each processor specifying which memory module serves as the primary module for this processor). The contents of the five location status word is shown below. The use of the specified registers and flip flops will become clear after reading Paragraph 6.1.1.3.

Location	Contents
1)	P (Program counter)
2)	B_1 (Bank register one)
3)	L (Lower accumulator)
4)	T_1 (Tagged register one)

5)

1 - 7 Fill clock	8 - 9 Interrupt code	10 - 13 Arithmetic code	14 - 16 Processor mode

(control word)

Explanation of bits in 5)

8-9	01	Fill clock interrupt
	10	No response interrupt
		(Storing the real time clock interrupt is not necessary since it has the highest priority)
10-13	0001	G (Greater than)
	0010	L (less than)
	0100	E (equal)
	1000	O (overflow)
14-16	001	FM (Floating Point Mode)
	010	RM (Repeat Mode)
	100	LS (Load Status)

The status words shown here must be picked up by the hardware so that an executive program will be able to gain control of the processor without losing the status of the interrupted program. The explicit sequence of operations that occur following an interrupt are given below. It should be noted that an interrupt may occur at anytime; however, no interrupt action occurs until the present instruction is complete (for repeat instructions only the present instruction cycle will be completed no repeat cycles will be executed).

1. After the present instruction is complete the above status word is stored in five consecutive memory locations by the hardware. The initial storage location for the status word is specified by the fixed wired address and primary memory register. This address is loaded into the program counter after it has been transferred to the memory buffer register. The program counter is then used to address the memory for five sequential write cycles. The five cycles load the above status words.

2. The contents of the sixth location following the wired address is picked up and placed in the program counter. This location is used as a jump into the appropriate executive routine. Note that a different executive routine can be entered for each type interrupt even though all interrupts use the same storage locations for the status words. The program counter is simply used directly to jump for the real time clock interrupt, it is incremented once for the no response interrupt, and it is incremented twice for the fill clock interrupt.

3. Each executive interrupt routine that is now entered must first move the status words to an appropriate storage area. This is done so that future interrupts will not destroy the stored information before the interrupted program can be restarted. The executive will then pickup the rest of the processors registers and place them in the same storage area. These registers, U_1, U_2, B_2, and T_2-T_7 are all directly addressable. When all the processors registers have been stored the appropriate interrupt flip flop is reset and the executive will begin the tasks associated with the interrupt. It should also be noted here that in order to store the complete

set of registers the executive does not have to alter any of the processor registers not automatically loaded into the status word. T_1 and B_1 are simply loaded with bank addresses so that all the other processor registers including those in the memory (status word) can be moved directly to any area of memory desired.

The sequence of actions that take place if a higher priority interrupt occurs while a lower priority interrupt is being processed must now be specified. In this situation the higher priority interrupt takes control of the processor after completion of the present instruction. If the initial five status words have not all been stored, the new interrupt completes this process and then jumps to its executive routine. If these words have been stored, the hardware will use the hardwired address plus five to pick up the jump for the priority interrupt. The executive then takes over and begins storing the status word in the normal manner. Note that in the latter case the existing status word (setup by the earlier interrupt) and processor registers can be stored since each interrupt executive routine does not change any of these locations until after it has turned off its interrupt flip-flop. (While the flip-flop is "on" the executive is only involved in storing the complete status word in an appropriate storage area.) The above sequence of events means that when the originally interrupted program is returned to the processor, it will be immediately interrupted by the lower priority interrupt. This is of course simply the original interrupt trying again to get processed.

The interrupt process discussed above may take greater than 90 μs to store all fourteen processor status words in a specified storage area. This would be a worst case time if both processors were using the same memory (each processor would get one half the memory cycles) and if each I/O unit was to request a cycle. This situation is unlikely if the programs are carefully allocated; however it could occur especially if a third processor were added to the system. This situation would cause no problems unless a periodic program was waiting for execution. The only time that this situation exists is when the real time clock interrupt occurs. A little careful thought as follows will show this to be the case. If a no response interrupt occurs during a periodic program execution, the system has definitely failed since a periodic program cannot be locked out of its own primary memory. Therefore the processor-memory combination will be turned off and the astronauts notified. If a fill clock interrupt occurs and a periodic program is to be executed next, the other processors are locked out of this processors' memory since periodic programs must already be processed. (Remember that the lockout is on any time periodic programs are being executed or a critical computation phase is taking place.) As a result the interrupt and executive routines will be handled directly by the memory in less than 50 μs. In order to limit the storage time when a real time clock interrupt occurs, the memory lockout is simply set at the time this interrupt occurs. The storage sequence will then take a fixed length of time to be executed (approximately 46 μs).

The only part of the interrupt process not yet specified is the method of loading the processor to restart an interrupted program. The addressable registers U_1, U_2, B_2, and T_2-T_7 are loaded by instruction. The same basic hardware that was used to store the five interrupt status words is then used to read B_1, L, T_1, the control word, and P from five sequential memory locations. A load status command (LDS) that loads the program counter with the initial address of the five status words is used to initiate the above sequence of operations.

Program completion interrupts have not been included in the above since jumps to certain executive routines will be placed at the end of the programs. Internal failure interrupts from self-check routines or hardware have also not been included (except for the no response interrupt) since the failure detection studies of Section IV, 4.2 specified that the processor and a memory or I/O unit be turned off after the detection of any internal error. These failures will also set failure status flip-flops in the I/O units.

An arithmetic interrupt may be necessary for ground checkout, but it does not seem to be useful for the operational phases of the mission. Therefore its inclusion should be investigated along with any ground checkout studies.

A need for external interrupts (this includes any external sources, e.g. console, vehicle subsystems, etc.) is also not foreseen at this time. The requests and signals that could be considered interrupts will instead set flip-flops in the I/O units. This status word in the I/O unit can be set by requests from scientific experiments or from the astronauts, and by external failure signals from the various system modules. The status word will be periodically monitored by the executive, and the requests passed to the request processor program and executed as necessary. As an example of the above procedure consider what happens when the failure bit of a status word is set indicating a failure of processor No. 1 and/or memory No. 3. These modules will first be turned off. The correctly operating processor will detect this failure when its present round of periodic programs have been completed. It will than make the highest priority background program the diagnostic program to localize a failure to a processor or a memory.

6.1.1.2 Instruction Set

This section gives the instruction set selected for the multiprocessor. The set was chosen from a larger list initially generated and given in Reference 18, the third quarterly report.

The operation times listed for each instruction assume a 2 μs memory cycle. This cycle time and its relation to the processor are discussed in Paragraph 6.1.2. on the memory. This discussion points out that the execution times for a number of instructions could actually be decreased since the desired memory (1 μs access time and 2 μs cycle time) will probably actually have a 1.5 μs cycle time; however, since the explicit control unit and control sequences have not been designed yet, only approximate instruction execution times using a 2 μs memory cycle are given in the following list.

The following abbreviations are used:

Repeat Mode: RM

Floating Point Mode: FM

Address Decrement: m

Address after banking and/or indexing: M

Contents of addressed memory position: (M)

Replaces: $\longrightarrow$

Upper Accumulators, Lower Accumulator: U_1, U_2, L

Accumulators: A

Index/Bank registers, accumulators: R

Memory Buffer Register: MB

Index/Bank Registers specified by T bits: T_n

Index/Bank Registers specified by B bit: B_0, B_1

Program Counter: P

n position left shift of A: LA^n

n position right shift of A: RA^n

n position left cycle of A: LA_c^n

n position right cycle ofA: RA_c^n

In order to show the primary tasks carried out in each bit time of an instruction, the instruction and operand cycle of the "add" instruction are given below.

Add (4 μs)		Instruction Cycle		
Bit times	1	2	3	4
	P to memory (set up inst. address)	memory to MB (processor receives instruction)	m + B $\longrightarrow$ MB (Set up operand address)	MB + $T_n \rightarrow$ MB P + 1 $\rightarrow$ P
		Operation Cycle		
	1	2	3	4
	MB to memory (Set up operand address)	memory to MB (processor receives operand)	U + MB $\longrightarrow$ U (addition performed)	

It should also be noted that two columns, RM and FM, are included in the instruction list in order to show which instructions can be executed in the repeat mode or floating point mode. The instructions not checked in the columns can be used in the special modes, but they will be executed as shown for the normal mode of operation. The floating point mode of course uses all double length words.

Operation times for the repeat mode depend on the number of operands to be processed. For example, for a list of n operands, a repeat mode instruction would take 2 μs plus n times the execution time for the instructions operand cycle. The operation times for floating point instructions are very dependent on the amount of

hardware added to speed the instructions up. Since the requirements for floating point operations are not explicitly specified, no operation times will be given for the floating point operations; however, as an example an implementation of a simple floating point add and multiply operation using the existing multiple registers was investigated. This add operation takes 25 μs and the multiply 40 μs. These operation times could easily be decreased by adding additional hardware if future requirements studies dictate this. The functional operation of both the floating point and repeat modes are discussed in greater depth in Paragraph 6.1.1.3.

6.1.1.2.1 Arithmetic and Logical Instructions

1. ADD

			RM	FM
ADU[1]	(M) + U ⟶ U 4 μs (2 memory cycles)	*Since the memory contents are loaded into the memory buffer register the actual add is between the MB and the accumulator. The add operation itself takes 500 ns.	X	X

2. ADD - register to register (Op code extension) [2]

ADR	$R_1 + R_2 \longrightarrow R_1$ 2 μs	*This instruction and all other register instructions use the register format shown below.

Register Instruction Format:

1 : 6	7 : 14	15 : 18
Op Code	Registers	Op code extensions

1 : 6 - These bits are always the same. They use one of the available 64 op codes to specify the register class of instructions. The explicit instruction is specified by 15:18.

7 : 14 - Bits 7:10 specify R_1 and bits 11:14 specify R_2. In instructions using only one register, only bits 7:10 are of interest. As mentioned earlier R could be any of the Index-bank registers or accumulators.

15 : 18 - These bits specify the register instruction to be executed.

The capability of using the index-bank registers and accumulators in register operations costs very little in terms of hardware; as a result this full capability has been included. Note that full register to register operations have been

(1) Recall that the accumulator bit can make ADU either ADU_1 or ADU_2.

(2) All instructions labeled "register to register" or "register" will be implemented by op code extension.

included even though present evaluations show little need for more flexibility than register to accumulator operations. However only a minimum of usage is necessary to warrant the small amount of extra hardware.

			RM	FM
3. and 4.	Subtract - same as 1 and 2.			
5.	Complement - register			
	COR — R_1 ⟶ R_1	*Floating Pt. would complement R_1 and R_2 (if this mode is desired here)		
	2 μs			
6.	Multiply			
	MPU (M) x U ⟶ U, L	*A two bit at a time multiply will be used.	X	X
	~ 10 μs			
7.	Sum of Products Multiply			
	SPM (M) x U_2 + U_1, L ⟶ U_1, L		X	X
	~ 10 μs	*Note that this is a full length sum of products multiply. This is not possible with only one accumulator.		
8.	Multiply - register			
	MPR R_1 x R_2 ⟶ R_1, L	*The only restriction is that R_1 or R_2 do not include L. Placing the additional restriction that R_1 is U_1 or U_2 could save some control hardware complexity.		
	~ 8 μs			
9.	Square - register			
	SQR U x U ⟶ U			
	~ 8 μs			
10.	Divide			
	DIU U, L ÷ (M) ⟶ U quotient L remainder		X	X
	~ 14 μs	*This time is for a straight one bit at a time divide.		

11. Divide - register — RM FM

DIR $R_1, R_2 \div (MB) \longrightarrow R_1$ quotient
L remainder

~ 12 μs

*Adding the restriction that R_1 and R_2 are only U_1, U_2, and L would simplify the control hardware.

12. And

ANU $(M) \cdot U \longrightarrow U$ — X X

4 μs

13. And - register to register

ANR $R_1 \cdot R_2 \longrightarrow R_1$

2 μs

14. and 15. Or - Same forms as And

16. and 17. Exclusive Or - Same forms as And

18. Absolute value - register

ABR $|R_1| \longrightarrow R_1$

2 μs

Register Transfers

19. Exchange - register to register

EXR $R_1 \longleftrightarrow R_2$

2 μs

20. Transfer - register to register

TRR $R_1 \longrightarrow R_2$

2 μs

Shifts

The shift instructions use a special format as shown below. It should also be noted that on all non-cyclic right shifts the sign is spread; whereas non-cyclic left shifts insert zeros.

1 : 6	7 : 8	9 : 11	12 : 16	17 : 18
Op Code	Register Specification	Index Tag - T_n	Shift Count	Op Code Extension

1 : 6 - These bits are always the same. They use one of the available 64 op codes to specify the shift class of instruction. The explicit shift instruction is then specified by 17:18.

7 : 8 - These bits specify the register to be shifted. Note that the index registers could also be shifted, but this feature is felt to be of little value.

01 U_1

10 U_2

11 L

9 : 11 - This gives the index register to be used to index the shift count. The index for the shift count is in bits 14:18 of the specified index register.

12 : 16 - These bits specify the amount to shift the indicated register up to a 32 position shift.

17 : 18 - These bits specify the shift instruction to be executed.

21. Short Right Shift — RM: — FM: X

SRS $RA^n \longrightarrow A$

$(2\mu s + \frac{n-1}{2}\mu s^{(1)})$

(same time for all shifts)

*Note that floating point mode can be used to convert the short shifts to long shifts with L as the right hand register.

22. Short Right Cycle

SRC $RA^n_c \longrightarrow A$ — FM: X

23. Short Left Shift

SLS $LA^n \longrightarrow A$ — FM: X

(1)This number and all shift execution times may be shortened considerably by putting in hardware for group shifting. The need for this feature should be investigated in the future.

			RM	FM
24.	Short Left Cycle			
	SLC	$LA_c^n \longrightarrow A$		X

Load and Store

25.	Load U				
	LDU	(M) ⟶ U		X	X
		4 μs			
26.	Load Address - register				
	LDA	M ⟶ U	*This instruction is of value when used for generating indirect addresses.		
27.	Load B Registers				
	LDB, B, T_n	(M) ⟶ B M = T_n + m	*B is either register B_1 or B_2		
		4 μs			
28.	Load Tagged Registers				
	LDT, B, T_n	(M) ⟶ T_n M = B + m	*T_n is any one of the tagged index-bank registers.		
		4 μs			
29.	Load Immediate				
	LDI, B, T_n	m ⟶ B if T_n = 000 m ⟶ T_n if $T_n \neq$ 000	*This instruction is useful for loading the index-bank registers when they are being used for indexing only.		
		2 μs			
30.	Store U				
	STU	U ⟶ (M)		X	X
		4 μs			
31.	Store Zero				
	SOM	0 ⟶ (M)	*This instruction is useful in the executive scheduler routine for setting up the chaining between programs.	X	
		4μs			

			RM	FM
32.	Store Decrement			
	STD $U_{11-18} \rightarrow (M)_{11-18}$	*This instruction is useful for changing decrements in instruction words.	X	
	4 μs			
33.	Store B Register			
	STB, B, T_n $B \longrightarrow (M)$			
	$M = T_n + m$			
	4 μs			
34.	Store T Register			
	STT, B, T_n $T_n \longrightarrow (M)$			
	$M = B + m$			
	4 μs			
35.	Increment and Replace			
	IAR $(M) + U \longrightarrow (M)$	*This instruction could be made as fast as 4 μs if it is considered useful enough to add another memory processor interface line; otherwise it will take 6 μs. The 4 μs execution time takes advantage of the 1 μs access of the NDRO memory. In any case if used frequently this instruction will save time and storage.	X	X
	6 μs			
36.	Transfer Memory to Memory			
	TMM, B, T_n $(M_1) \longrightarrow (M_2)$	*B + m gives the address of M_1 and T_n gives the address of M_2. The instruction saves a memory cycle and some storage; however its real value will come with use in the repeat mode to transfer blocks of storage.	X	
	6 μs			

Control Operations

37. Unconditional Jump — RM FM

JMP M ⟶ P

2 μs

38. Jump on Minus or Zero U

JMMU U ≤ 0 M ⟶ P

2 μs

*This instruction could be included in the JMC instruction 48; however, it is used frequently enough to be included directly. This of course decreases execution time for this operation.

39. Jump and Set Index

JSX, T_n P ⟶ T_n

M + B ⟶ P

2 μs

*This instruction is for subroutine linkage. It must be preceded by a load bank, if not indirected, in order to setup B with the proper subroutine address. If indirected relative to B, the proper location of the subroutine can be kept in the working storage region for this program. This latter approach is the most convenient. Note that the command also stores the return address in an index-bank register. A JMP instruction indexed by the same index-bank register will return back to the original program exit plus the displacement (m). This instruction is convenient if it is desired to have the calling sequence (subroutine parameter values) in the program bank after the JSX. (For example the periodic programs may obtain the I/O variables and place them here.) In this

case the loaded index register can also be used by the subroutine to get at the calling sequence.

40. Jump and Store Return

JAS P ⟶ (M)

(M+1) ⟶ P

6 μs

*With this instruction the working storage has a place for the return address above the location that contains the subroutine address. While in the subroutine, the same bank-index register that is used for working storage reference can be used to get at the calling sequence in the working storage. The setting up of the subroutine address in P does not have to be indirected but it is shown this way since it is easiest. Another approach would use a bank register set up so that T_n would address the subroutine. The jump would then load this index in the program counter and use B + m to store the program counter. This method would take only 4 μs, but it would require setting up T_n first.

41. Decrement and Skip on T_n comparison

DSTC, B, T_n T_n-1 ⟶ T_n

if $T_n \leq$ (M)

then P + 2 ⟶ P

M = B+m

4 μs

*The register to be compared is first decremented by one. This instruction is good for counting down a register that is being used as an index and bank register.

42. Decrement and Skip on B comparison — RM FM

DSBC, B, T_n B-1 ⟶ B

if B ≤ (M)

then P + 2 ⟶ P

M = T_n + m

4 μs

43. Decrement and Skip on Immediate Index Comparison

DSIXC, B, T_n T_n-1 ⟶ T_n

if T_n ≤ m } if $T_n \neq 000$

then P + 2 ⟶ P

B-1 ⟶ B

if B ≤ m

P + 2 ⟶ P } if $T_n = 000$

then

2 μs

44. Compare — X

CMP U > (M); 1 ⟶ G; 0 ⟶ E, L
U = (M); 1 ⟶ E; 0 ⟶ G, L
U < (M); 1 ⟶ L; 0 ⟶ G, E

*This instruction requires the inclusion of G, E, and L flip flops in the processor.

4 μs

45. Compare Immediate

CMPI R > m, 1 ⟶ G, 0 ⟶ E, L
R = m, 1 ⟶ E, 0 ⟶ G, L
R < m, 1 ⟶ L, 0 ⟶ G, E

*The B and T_n bits are decoded so that they will indicate one of U_1, U_2, L, B, or T_n.

2 μs

46. Compare B Registers — RM FM

CMB

B > (M); 1 → G, 0 → E, L

B = (M); 1 → E, 0 → G, L

B < (M); 1 → L, 0 → G, E

M = T_n + m

4 μs

47. Compare T_n Registers

CMT

T_n > (M), 1 → G, 0 → E, L

T_n = (M), 1 → E, 0 → G, L

T_n < (M), 1 → L, 0 → G, E

M = B + m

4 μs

48. Jump on Conditions

JMC, B, T_n

If any condition true then

B + m → P

2 μs

*T_n holds a masking bit for each condition. The conditions can be L, G, E, O (over flow), or I/BL (I/O busy or locked out). These conditions are held in flip flops in the Processor. The masking T_n bits are "anded" with the respective conditions in order to determine a jump.

49. Compare and Skip

CAS

U > (M); P + 1 → P

U = (M); P + 2 → P

U < (M); P + 3 → P

4 μs

*This instruction could have been made non-indexable with T_n, and then used the T_n registers for a branch on the conditions. — X

50. Compare Absolute Greater and Skip — RM FM

CAGS $|(M)| > |(U)|$ then $P + 2 \longrightarrow P$

*This is a useful instruction for scientific experiment data compression.

51. Compare Tolerance and Skip

CTS If $U + L \geq (M) \geq U - L$ then $P + 2 \longrightarrow P$

4 μs

*This instruction is useful in status monitoring and data compression.

52. Compare-Register

CMR

$R_1 > R_2$ $1 \longrightarrow G, 0 \longrightarrow E, L$

$R_1 = R_2$ $1 \longrightarrow E, 0 \longrightarrow G, L$

$R_1 < R_2$ $1 \longrightarrow L, 0 \longrightarrow G, E$

2μs

53. Increment T_n Registers

INT, B, T_n $T_n + (M) \longrightarrow T_n$

$M = B + m$

4 μs

*This instruction is useful for moving up and down lists, and for handling the addresses in matrix manipulations.

54. Decrement Index-Bank Registers and Skip

DXS, B, T_n

$T_n - m \longrightarrow T_n$

$T_n \leq 0$ $P + 2 \longrightarrow P$ } if $T_n \neq 000$

$B - m \longrightarrow B$

$B \leq 0$ $P + 2 \longrightarrow P$ } if $T_n = 000$

2 μs

55. Decrement and Jump on T_n Greater Than Zero — RM FM

DJTZ, B, T_n $T_n - 1 \longrightarrow T_n$

$T_n > 0, M \longrightarrow P$

$M = B + m$

2 μs

56. Decrement and Jump on B Greater Than Zero

DJBZ, B, T_n $B - 1 \longrightarrow B$

$B > 0, M \longrightarrow P$

$M = T_n + m$

2 μs

57. Execute*

EXC $(M) \longrightarrow MB$

2 μs

*This instruction sets up the address of the next instruction in the memory buffer register. The program counter is not incremented for this instruction. It will then be incremented and used to continue the normal instruction flow after the instruction addressed by the execute (unless this instruction is a jump). The execute is then just a one instruction jump. It should be noted that this is the only instruction that does not leave the memory buffer empty after its completion; as a result it is non-interruptable.

58. Repeat

REP $(M) \longrightarrow T_7$

$M = B + m$

4 μs

*This instruction initiates the repeat mode for the instruction following it. A repeat flip flop is set and an index-bank register, T_7, is loaded

RM FM

with the number of operands to be processed. This register is counted down to zero and the repeat mode terminated. T_6 is also used to hold the program counter during the execution of the following command in the repeat mode. A functional description of the repeat mode is given in the next section.

59. Repeat Immediate

REPI m $\longrightarrow T_7$

2 μs

*Same as 58 except for immediate loading of T_7.

60. Set Real Time Clock

SRC (M) $\longrightarrow$ RTC (1)

4 μs

61. Read Real Time Clock

RRC (register) RTC $\longrightarrow$ U (1)

2 μs

62. Set Fill Clock

SFC, B m $\longrightarrow$ FC B=0 (1)

$U_{10-17} \longrightarrow$ FC B=1

63. Load Status Address

LDSA (M) $\longrightarrow$ P

1 $\longrightarrow$ LS

*This instruction loads the program counter with the initial address of five status words that will then be automatically loaded into the processor. LS is the load status flip flop in the processor that

---

(1) See section 6.1.1.1(a) for comments on these instructions.

is used to start the loading operation. Since the status word addresses will be picked up from an executive table, there is no need to indirect this instruction. The instruction operation was explained in paragraph 6.1.1.1(b).

64. Call I/O

CIO

(M) ⟶ I/O

(M+1) ⟶ I/O

6 μs

*This instruction is used to send two control words to an I/O unit in order to start an I/O operation. Bit 17 of the address is set to one in order to specify that the memory contents should be sent to the I/O unit. The I/O unit number is specified in the first two bits of the control word. A full explanation of this instruction is given in paragraph 6.1.3 on I/O.

65. Load Processor Registers - register

LPR

MB_7 = 0 then reset all flip flops

MB_7 = 1 then set all flip flops

$MB_{8,9}$ ⟶ PMR

$MB_{10,11}$ ⟶ IMR

MB_{12} ⟶ FM

MB_{13} ⟶ F

MB_{14} = 1 then 0 ⟶ RTC, RTC EXT

2 μs

*IF the corresponding bits in MB_8 to MB_{13} are one the appropriate flip flops are set or reset depending on MB_7. If the bits are 0, nothing occurs. An explanation of the flip flops is given in paragraph 6.1.1.3.

66. Load Lock Out - register RM FM

LLO	MB_7 = 1 then set lockout	*$MB_{8,9}$ give the I/O unit and $MB_{10,11}$ give the memory unit to be locked out or enabled. Upon receipt of the instruction the processor immediately sends a "1" signal back to the memory on its lockout line. The memory then uses bits 8, 9 in its output data register to store the I/O unit from which it will receive requests. It also uses bit 7 to decide on setting or resetting its lockout registers. This operation takes effect before the next memory cycle. The I/O unit is also sent a signal by the memory notifying it that only this memory is to be used.
	MB_7 = 0 then reset lockout	

It should be remembered that some of the instructions presented above are register to register instructions; therefore these are implemented by op code extension. A large number of these instructions have been included to give added flexibility. Some of the instructions given above do not require an accumulator specification; therefore, these can make use of the accumulator bit for further op codes. In addition some instructions do not require both the accumulator specification and the indirect bit for indirect purposes; therefore these bits may be used to increase the op codes. (The immediate instructions are good examples of this.)

A 5 bit op code and an accumulator bit are used for all instructions where it is possible to use the two accumulators; for the other instructions the op code will be considered to be 6 bits. (This distinction is only of real importance to the hardware). The op code is considered to be 7 bits for instructions that do not use multiple accumulators or indirect addressing.

Following is a tabulation of the instructions into five columns: (a) those requiring an accumulator tag, (b) those not requiring an accumulator tag, (c) register to register instructions (op code extension beyond 6 bits), (d) shift instructions (op code extension beyond 6 bits), and (e) instructions without indirecting or multiple accumulators.

Accumulator Tag	No Accumulator Tag	Register	Shift
(5 + U) op code	(6 bit) op code	ext op code	ext op code
ADU	LDB, B, T_n	ADR	SRS
SBU	LDT, B, T_n	SBR	SRC
MPU	SOM	COR	SLS
SPM	STB, B, T_n	MPR	SLC
DIU	STT, B, T_n	SQR	
ANU	TMM, B, T_n	DIR	
ORU	JMP	ANR	
EORU	JSX, T_n	ORR	
LDU	JAS	EORR	
LDA	DSTC, B, T_n	ABR	
STU	DSBC, B, T_n	EXR	
STD	CMB	TRR	
IAR	CMT	CMR	
JMMU	JMC, B, T_n	RRC	
CMP	INT, B, T_n	LPR	
CAS	DJTZ, B, T_n	LLO	
CAGS	DJBZ, B, T_n		
CTS	EXC		
SFC, B	REP		
	SRC		
	CIO		

6.1.1.2.2 No Indirect Bit, No Accumulator Bit

LDI, B, T_n

DSIXC, B, T_n

CMPI

DXS, B, T_n

REPI

LDSA

Six op code bits will be used for the tagged and non-tagged instructions above. This means that when counting the number of instructions used, the accumulator tagged instructions should be multiplied by 2. One instruction should be added to designate register op code extension instructions and one to designate shift instructions. One half of the no indirect bit instructions should be added since they can use the indirect bit to distinguish between two instructions. The above is represented by the following calculations:

$$\text{Number of Instructions} = 2\text{x (Acc. tag instr.)} + \text{(no acc. tag instr.)} + 2 + 1/2\text{ (no ind. bit instr.)}$$

$$= 2 \times 19 + 21 + 2 + 3$$

$$= 64$$

This is, of course, the number of op codes available from six bits.

6.1.1.3 Functional Description

The past sections have discussed the basic processor features and its operation in the overall multiprocessor system. This section will give an explicit explanation of the processor's internal operation. This will include a presentation of registers, timing, and control.

6.1.1.3.1 Registers

Figure 6-5 shows all the registers in the processor and the majority of the control flip-flops. (A few additional flip flops may be added instead of gating to aid in the explicit implementation of the instructions.) There is also, of course, a large amount of gating that is not shown. However the effects of this gating on the operation of the processor will be explained. Figure 6-6 gives a better understanding of the processor operation by presenting some of the connections in the processor. The control flip-flops in these figures will be discussed in 6.1.1.3.4.

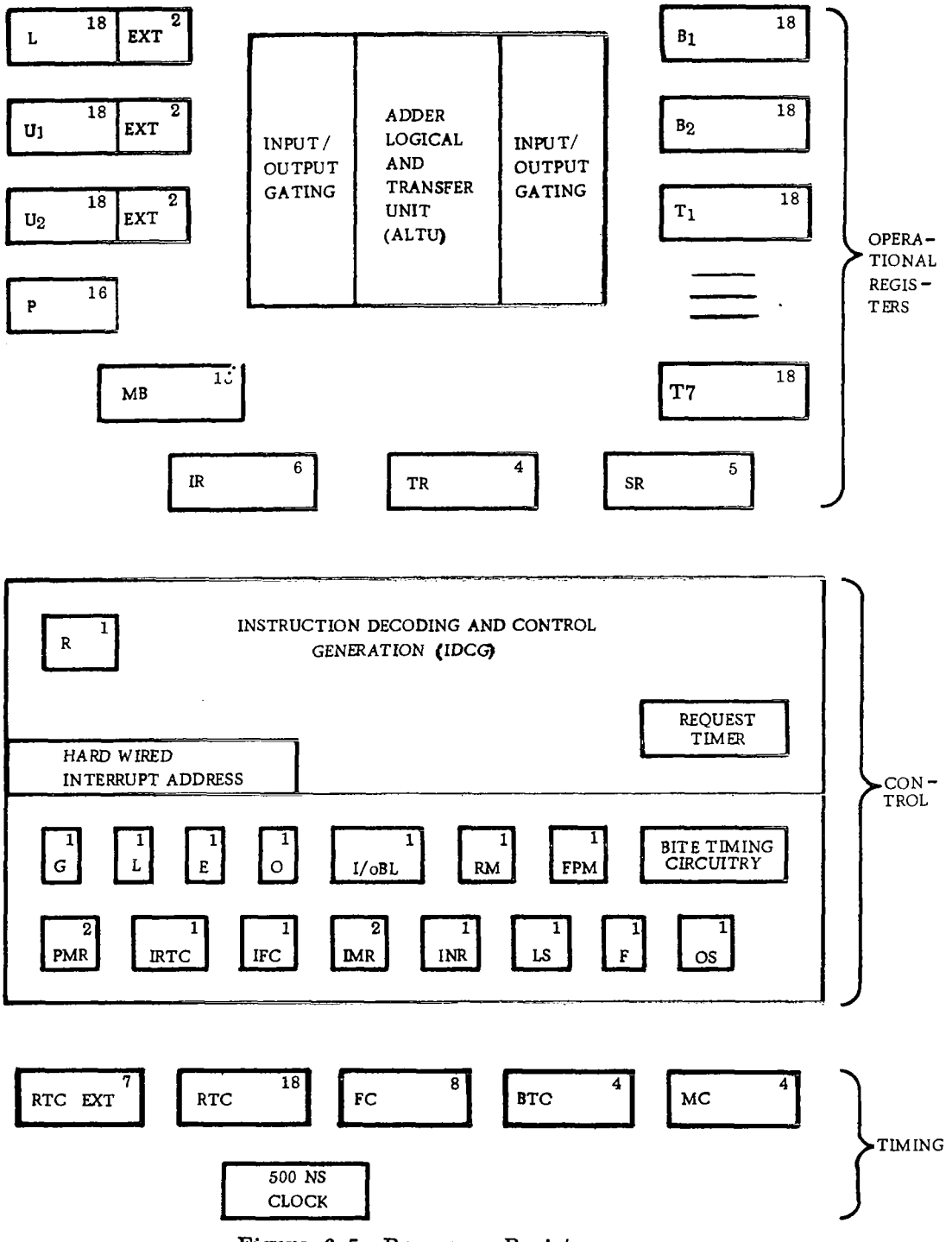


Figure 6-5. Processor Registers

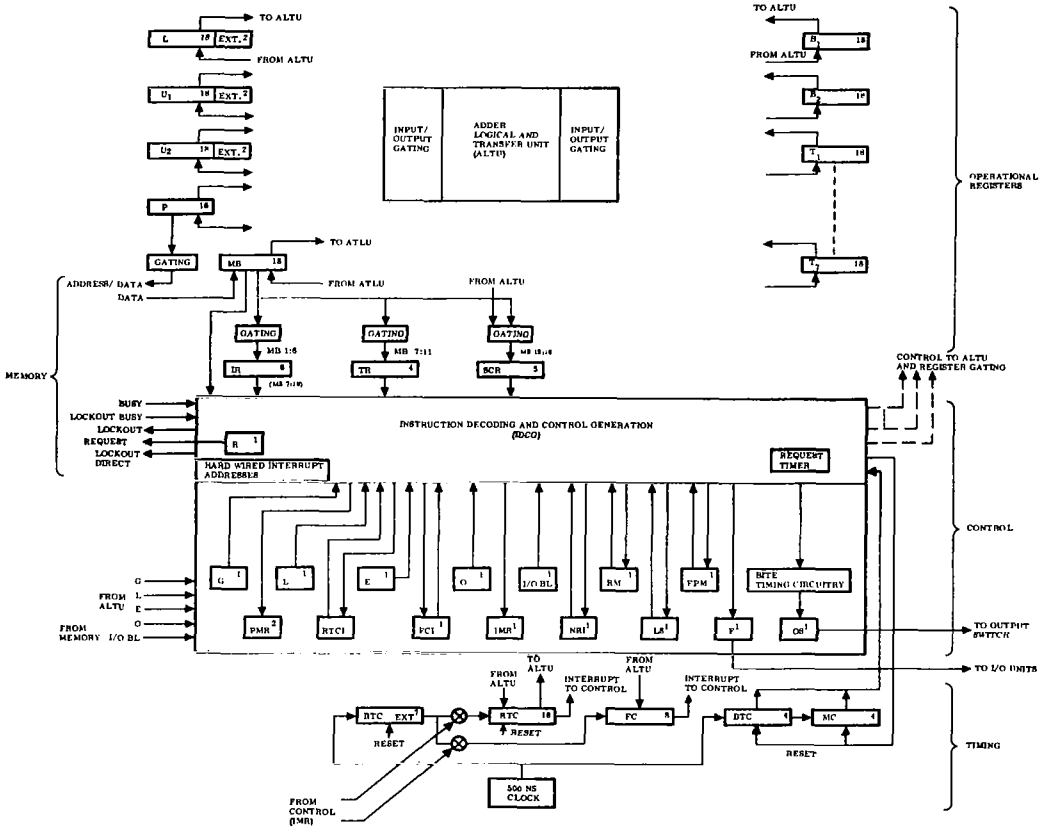


Figure 6-6. Processor Registers and Connections

L - Lower Accumulator: The lower accumulator is used primarily in multiply, divide, and floating point operations to hold the lower half of a data word; however it can also be used for hot storage and data manipulation in shift and register operations. This accumulator, U_1, and U_2 have a two bit extension onto their eighteen bits in order to hold the overflow carries which may be generated in the two bit at a time multiply operation. This extension is not shown on the bank-index registers; but either the accumulator extension will be time shared for register multiplies or the extra bits will be added.

U_1, U_2 - Upper Accumulators: The upper accumulators are the primary arithmetic and logical registers in single precision operations. They are also used to hold the upper half of data in floating point operations and to hold and manipulate data in shift and register operations.

P - Program Counter: This register is used to sequence the flow of control in the processor. It is not only used to access instructions but also to provide memory addresses for interrupt status word storage and for the operand cycles of instructions operating in the repeat mode. It must therefore be connected both to the ALTU and to the memory interface lines.

MB - Memory Buffer: The memory buffer receives data and instructions from the memory, sends data and operand addresses to the memory, and holds the divisor in divide operations and the multiplicand in multiply operations. It also holds one of the operands in all other arithmetic and logical operations with the memory, and holds the next instruction address for an "execute" command. In addition to the above tasks since the MB receives all instructions it keeps many of these bits for the instruction decoding and operation. For example, it holds the B bit for address generation, the register to be shifted in a shift operation, the registers to be operated on in register operations, and the op code extension for register and shift operations.

B_1, B_2 - B Index - Bank Registers: These registers hold both index and bank values for address generation and looping control. They also provide hot storage and take part in register operations and comparisons with memory contents. One of these two registers is added to the address decrement for all operand address generation.

T_1 to T_7 - T_n Index - Bank Registers: These registers have the same functions as the B registers. The only difference is that operand addresses can be generated without adding any T_n register to B + m. (Tag 000 specifies no indexing with the T_n registers.)

ALTU - Adder, Logical, and Transfer Unit: This unit contains all the circuitry for carrying out arithmetic and logical operations including comparisons. It also provides for transfers amongst all the registers mentioned above and detection of overflows.

IR - Instruction Register: The instruction register holds the six bit op code throughout the instruction execution.

TR - Tag Register: The tag register holds the indirect, and T_n bits of the instructions. It is necessary so that m + B can be generated, stored in MB, and then added to T_n prior to an operand cycle.

SCR - Shift Count Register: This register holds the shift count for shift commands, for normalizing floating point numbers, and for equalizing operand exponents in floating point addition and subtraction. It is counted down to zero by one count for each shift. The register can be loaded from the ALTU in addition to the MB since shift counts may be indexed prior to being loaded into SCR for execution.

6.1.1.3.2 Timing

The basic functions of the RTC Ext, RTC, and FC have been described earlier, however their operation as depicted in Figure 6-6 will be briefly discussed along with the operation of the bit time counter (BTC) and mode counter (MC). All the counters in the system operate from a 500 ns clock. At this time it is not clear if a good small substitute for a two megacycle crystal will be developed by the 1975 technology time frame; however even with a crystal oscillator only a small portion of a MOS/SOS chip will be necessary for the oscillator circuitry and one shot. The crystal would then either be mounted on the SOS chip or in a separate small pack. The clock will provide the basic time unit pulse to the RTC Ext and BTC. These counters in turn count up and drive the RTC and FC, and the MC respectively. It should be noted from Figure 6-6 that the operation of both the RTC and FC can be inhibited by control signals from the interrupt mask register in the control unit. The bit time counter provides the control unit with four lines, each signifying a separate bit time in the instruction or memory cycle. The mode counter provides timing for the execution of the longer instructions. A three bit mode counter is sufficient for the longest single precision instruction (divide), but as many as five bits may be necessary in order to provide timing for floating point divide and multiply. This will depend on how much hardware is added to speed up the floating point operations. The end of this section discusses floating point in greater depth. A four bit mode counter is shown in Figure 6-5. The bit time and mode counters are reset to zero by the control unit at the start of an instruction or operation cycle. This occurs when the processor is accepted by a memory. After this the counters count up and there values are used by the control unit until the instruction execution is complete.

6.1.1.3.3 Memory Interface

The lines on the memory-processor interface are given below.

Component Processor

Interface Memory

Output (to memory)

request	*One separate line to each memory - It requests memory cycles.
address/data	*18 bit two-way bus common to all memory modules - It sends addresses to the memory and sends and receives data.

read/write	*Bit 18 of the address/data bus - This line is available for read/write designation since the memory address sent over the lines is only 14 bits (each module contains 12K words).
lockout	*One separate line to each memory - This line is sent by a LLO command to notify the memory to look at bits 7 to 9 in its data register for lockout information.
lockout direct	*One separate line to each memory - This line is sent on the occurrence of a real time clock interrupt to notify the memory to lock out all other processors.
power off	*One common line to all memories - This line used to turn off all lockouts by this processor after it has failed or has been turned off by the astronauts.
I/O	*Bit 17 of the address on the address/data bus - This line notifies a requested memory that the data word for the present processor memory cycle should be sent to an I/O unit.

Input (From Memory)

busy	*One separate line to each processor from each memory - It is used to notify a processor of acceptance of a request.
lockout	*One separate line to each processor from each memory - It notifies a processor if this processor is locked out of any memories.
data	*The same common 18 bit bus listed under output.
I/O busy or locked out	*One seperate line to each processor - It notifies the processor that the called I/O unit is busy or locked.

The timing of a memory cycle is given below and shown in Figure 6-7.

1. The processor sends a request to a memory (0 to 1 transition occurs on the request line) and at the same time it places the memory address and read/write request on the address/data lines.

2. The memory module contains a simple round robbon type of scanner for sequentially selecting and granting processor and I/O requests for memory cycles. After the memory scanner picks up the address from the bus and sends the processor a not busy signal (0 to 1 transition occurs on the busy line). The memory uses the next 500 ns to address the specified memory position and to load its data register if a read is required.

3. The processor uses the 0 to 1 transition of the busy signal to start its bit time counter and prepare to read or write. For a read cycle the processor memory buffer is loaded during bit time two by the 1 to 0 transition of the memory busy signal. The only requirement is that this load must be complete 1μs after the memory accepts the processor request

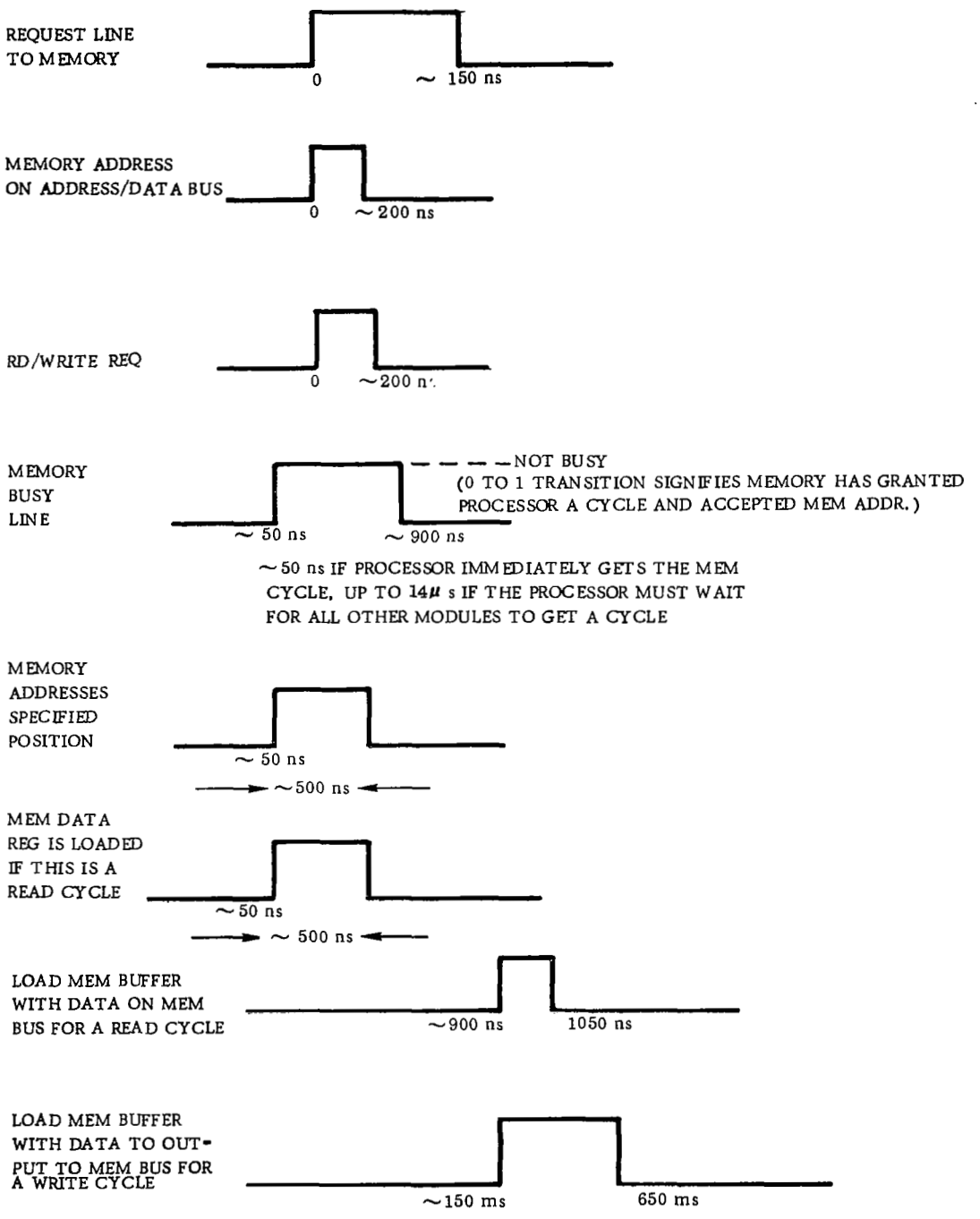


Figure 6-7. Memory Cycle Timing

For a write cycle, the processor loads its memory buffer register and bus to the memory with the data. It then turns off its request signal upon the 0 to 1 transition of the memory busy line (this will occur approximately 100 ns after the memory busy signal. The 1 to 0 transition of the request signal causes the memory to load its data register with the information on the common bus. The write may therefore be accomplished in approximately 650 ns.

6.1.1.3.4 Control

The control section of the processor receives a number of lines from the memory and from various parts of the processor which it then uses to set control flip-flops or to generate sequences of control signals that get sent throughout the processor and back to the memory. This operation is depicted in Figure 6-6 and will be functionally described here.

The flip-flops greater than (G), less than (L), and equal (E) represent conditions generated from a comparison carried out in the ALTU. After the comparison, these flip-flops are set or reset by the ALTU as appropriate. The control unit then uses these flip-flops to control future processor actions during a JMC (jump on conditions) instruction. The overflow (O) flip flop is set or reset by the ALTU after arithmetic overflows. It can then be used to cause a jump during execution of a JMC. It is also used in floating point operations to signify the need for the hardware to normalize. The I/O busy or locked out (I/O BL) flip flop is set or reset by a signal from the memory (the I/O sends the signal to the memory first) during a CIO (Call I/O) instruction. It notifies the processor that the I/O unit requested is busy or locked out from the requesting memory; as a result a CIO command should generally be followed by a JMC instruction to check the I/O BL flip-flop. If an I/O unit is not available, the processor control can then jump to the executive so that a new program can be scheduled.

After an interrupt occurs, the control unit generates a store status sequence to store five words in memory. As soon as the present instruction is completed, the appropriate flip flop, RTCI (real time clock interrupt), FCI (fill clock interrupt), or NRI (no response interrupt) is set. These flip flops are set respectively by the RTC zero interrupt, the FC zero interrupt, or the request timer "one shot" and gating. (This latter hardware checks to see if the processor is requesting a memory cycle from a memory it is locked out of, or if the processor has not been granted a request for greater than 14 μs) The control unit then executes the interrupt sequence given in paragraph 6.1.1.1. As also mentioned in this section the IMR (interrupt mask register) can be used to inhibit the RTC and FC so that these interrupts will not occur. This section also mentions that the load status flip-flop (LS) is set by the LDS command in order to reinitialize an interrupted program.

The Bite timing circuitry and output switch flip-flop (OS) are used to check failures of the processor and to switch control of critical outputs to another processor. These functions were discussed in paragraph 4.2.2.2.

The failure flip flop (F) is set by the checking hardware or by a software self check routine using the LPR command. This flip flop sends the processor status to the I/O units and also causes the processor to turn off. The I/O units have a failure status word for the status of all modules. This is discussed again in paragraph 6.1.3.

The section of the control unit labled instruction decoding and control generation (IDCG) has the task of sending out sequences of control signals. These sequences are generated by decoding and combining all input control flip-flop lines, memory signals,

timing information, and MB, IR, TR, and SCR register contents. The purpose of most of the control lines into and out of the IDCG section can be understood from the earlier register and memory interface explanations. For example, the memory buffer register sends bit positions seven to eighteen to the control section in order to provide the B bit, R_1 and R_2 for register operations, R_1 for shift operations, and op code extension for both register and shift operations. The control section then uses these lines along with others, such as the op code from IR, to generate control sequences to implement a given instruction. Some of the lines providing control signals to the processor registers are shown in Figure 6-6 coming out of the right side of the IDCG. For example, these lines go to the ALTU to initiate transfer or arithmetic operations, etc., to the P, B, and T registers to increment or decrement by one, or to the accumulators to cause shifts.

The repeat mode operation was explained in paragraph 4.2.1.2., however the repeat mode instruction cycle timing will be shown here in order to offer a deeper understanding of the operation of the mode. It is initiated by the REP command setting the RM flip flop and loading T_7 with the number of operands to be processed. The program counter of the instruction to be processed in the repeat mode is saved in T_6, and this counter is then used to address the memory for all repeat mode operand cycles. The cycles of course continue until T_7 has been counted down to zero. The above operation is demonstrated by the following timing diagram of the REP instruction and the "add" instruction in the repeat mode.

		Instruction Cycle		
Bit Times:	1	2	3	4
REP (4 μs)				
	P to Memory (address inst)	Memory to MB (receive inst)	m+B $\longrightarrow$ MB 1 $\longrightarrow$ RM	P + 1 $\longrightarrow$ P
		Operation Cycle		
Bit Times:	1	2	3	4
REP (4 μs)				
	MB to Memory (address operand)	Memory to MB (receive operand)	MB $\longrightarrow$ T_7	
		Instruction Cycle		
Bit Times:	1	2	3	4
ADD (2+2n μs)				
	P to Memory (address inst)	Memory to MB (receive inst)	m+B $\longrightarrow$ MB P $\longrightarrow$ T_6	MB+T_n $\longrightarrow$ MB MB $\longrightarrow$ P

Operation Cycle

Bit Times:	1	2	3	4
ADD (2+2n μs)				
	P to Memory (address operand)	Memory to MB (receive operand)	U+MB $\longrightarrow$ U T_7 - 1 $\longrightarrow$ T_7 P+1 $\longrightarrow$ P	$T_7 \neq 0$ then continue $T_7 = 0$ then 0 $\longrightarrow$ RM $T_6 \longrightarrow$ P

Floating point operations are only briefly discussed in this report since the conclusion to definitely include a floating point mode cannot be made without a much deeper investigation of the applicable requirements; however, they have been briefly investigated in order to offer an understanding of their possible operation and implementation in this multiprocessor system. This mode can be implemented with the same set of processor registers as defined earlier. The only necessary additions are a good amount of control hardware and the ability to mask operations on the mantissa (30 bits) from affecting the exponent, and conversely. This can be fairly simply accomplished by hardware additions in the ATLU and at the registers. For example, during part of an operation only the mantissas could be allowed to go to the ATLU (zero's can be automatically substituted for the exponent bits). At the conclusion of such an operation of course the exponent bits (zeros) would not be loaded back into a register. In fact the addition of a masked mode of operation would make some of the additional hardware useful for both the masked and floating point modes. The sequence of primary hardware operations for a floating point add is given below.

ADD

1. Subtract exponents
2. Store the difference in SCR (shift counter)
3. Normalize the smallest operand with the count.
4. Save the exponent
5. 30 bit precision add of the mantissas
6. Normalize the result if overflow occurs
7. Store answer with exponent

This operation would take approximately 25 μs with the present processor hardware; however, if the requirements show it to be worthwhile, this operation could be substantially speeded up. For example the memory buffer could be made double length in order to hold both parts of the mantissa for the add, the adder could be made double length, and group shifting could be added to speed the normalization. These same innovations would also substantially increase the speed of floating point subtract, and multiply while making the hardware implementation of floating point divide practical. It would probably also prove worthwhile to make a number of single precision operations available in double precision. This could be accomplished by

eliminating in floating point mode some of the single precision operations. These free op codes could then be used while in floating point mode for useful single precision operations, such as single precision load, etc.

In a final design of the processor additional provisions will have to be made for ground check out. This may even require the addition of a debug mode with halt instructions etc. Explicit specification of these features would be tied to development of the ground check out equipment.

6.1.1.4 Rough Chip Distribution

Section III, 3.1 discussed the circuit densities, connections, and yields for MOS/SOS technology in the 1973-1975 time frame. The conclusions of this section were that device densities with reasonable yields including crossovers should be on the order of 5, 500 FET's per roughly 150 mils square. This was felt to be relatively conservative since processing break throughs could easily enable chips of approximately the same densities and four times the area to be produced with good yields. The processor described in this section requires approximately 330 flip-flops for its implementation. An approximation for the gates and the drivers (for interface lines) in the system would give a rough total (including the flip flops) FET or device count of 11, 000. If the processing break throughs develop, this processor could easily be placed on a 250 mil square chip. One feature of the processor that might help to enhance its implementation on a single chip is the similarity amongst the accumulators and the bank-index registers. It may be possible to build one or at most two register types and place spares on the single chip. Discretionary wiring techniques could then be used to connect the correctly operating registers and thus improve the chip yields. If the above break throughs do not materialize the processor could be placed on two smaller chips. The distribution amongst chips would be as follows:

Chip One

L, U_1, U_2, P, MB, B_1, B_2, T_1 –T_7, ALTU, RTC Ext, RTC, FC, BTC, MC, 500 ns clock.

This would amount to approximately one-half of the devices - 5, 500 FET's

Connections to chip - approx. 140

Chip Two

IR, TR, SCR, All control flip-flops, all control gates.

This would also take approximately one-half the devices - 5, 500 FET's

Connections to chip - Approximately 130

The above distribution of hardware requires 150 lead packages for the chips. This is much more sophisticated than todays 40 lead packs; but as pointed out in 3.1, these packages should be available. The above organization offers an additional advantage of being able to use a microprogrammed control unit so that simply changing "chip two" changes the instruction set and operation of the processor. Chip one could then be standardized for a number of diverse missions that may require different instruction sets. Another approach to the distribution of processor hardware would

be to simplify "chip two" by placing some of the control generation hardware in "chip one". This would of course increase the FET density in "chip one", but it would also provide a sizable reduction in inter-chip connections. The final decision on hardware distribution must wait for the final design stages when the technology base is precisely known.

6.1.2 Memory

Both magnetic and semiconductor memories have been studied for the multiprocessor main memory. The magnetic studies first looked at todays DRO core memories and NDRO plated wire memories as examples of the state of the art. A batch fabricated NDRO multiword memory was then chosen as the preferred magnetic memory approach for the 1973-1975 time frame. The semiconductor memory studies investigated an NDRO MOS/SOS coincident select memory organization. Both this system and the magnetic system are shown to meet all the system requirements while offering relatively little risk in being able to meet the reliability goals of the Manned Mars Mission in 1980. The MOS/SOS memory is shown to dissipate less power than the magnetic memory, but it will probably also offer slightly greater development risks. As can be seen from the above, neither system has the decided advantage; as a result a choice between the two cannot now be made. Both systems should be developed and investigated in much greater detail in order to be able to choose the most desirable approach.

6.1.2.1 LSI Semiconductor Memory

6.1.2.1.1 Introduction

This section describes a solid state memory candidate for the main multiprocessor memory. It should be remembered that there are one to four memory modules (3 for the mission considered) with the following characteristics:

No. of words	12, 000	(Variable Storage)
Bits per word	18	
Read/write cycle time	2 μs	
MTBF	62, 500 hours	
Failure rate goal	1.6% per 1000 hrs.	

The above MTBF and failure rate goals where obtained from the Monte Carlo simulations, described in Section V.

The design of the solid state memory is based on projected production technologies in the 1973-1975 time period. As for the processor modules, MOS/SOS technology has been chosen as representative.

When the reliability and performance requirements of the multiprocessor memory are translated to hardware requirements for the semiconductor main memory, three very important features become apparent.

1. Large Scale Integration via Batch Fabrication is essential for attaining reliability and performance objectives.

2. One or more standby power sources are needed for volatility circumvention for a read/write semiconductor memory.

3. The use of memory circuits which have extremely low standby power dissipation is very desirable in order to achieve: (a) LSI with low operating temperatures and, consequently, enhanced reliability; and (b) volatility circumvention by means of one or more small standby power sources.

These three reasons make the use of complementary MOS field effect transistors manditory for achieving a read/write memory cell with extremely low standby power. A comparison of the approximate standby power for a bipolar transistor memory cell and a MOS field effect transistor memory cell illustrates the problem.

Item	Complementary MOS Memory Cell	Complementary Bipolar Memory Cell
Standby current* per cell	2 nA	0.2 mA
Number of cells per array	4, 096	4, 096
Standby current* per array	8 μA	800 mA
Number of Array per module	54	54
Standby current* per module	0.432 mA	43.2A
Standby current* for 3 modules	1.29mA	129A

*nominal value at 25°C

Both of the memory cells in the above example are inherently volatile. However, due to the extremely low standby current of the complementary MOS memory cell, and the availability of high reliability rechargeable secondary batteries, i. e., heremetically sealed Nickel Cadmium batteries designed for space applications, it is practical to use one or more standby power supplies for volatility circumvention. In the case of the complementary bipolar memory cell, the standby power supply current is so large that volatility circumvention is very difficult to achieve. Very large and heavy standby batteries would be required for volatility circumvention for a few hours. This latter problem may be solved by a number of separate well isolated power lines from the redundant primary spacecraft power supplies; however, a relatively large amount of power would be drawn from these batteries by the bipolar cells.

One other principal factor must be considered in choosing between MOS field effect transistors and bipolar transistors as a memory circuit element. This is demonstrated reliability. A choice for 1967 is easy – bipolar transistor. However, a choice for 1973-1975 must be made. This allows 6 to 8 years for the MOS production technology to become mature and for reliability data to become available. The

basic problem with MOS field effect transistors has been silicon surface instabilities. Many improvements were made from 1964 to 1967. Stable P-channel MOS transistors are being made in 1967 by several manufacturers. Obtaining stable N-channel enhancement mode MOS transistors has been more difficult. However, at the 1966 International Electron Device Meeting, Signetics and Westinghouse reported successful fabrication of monolithic complementary MOS transistors. In a paper entitled "Monolithic MOS Complementary Pairs" by K. K. Yagura, G. M. Catlin and J. D. Hutchensen of Signetics Corporation, it was said that "In recent years, stable, discrete N-MOST's and P-MOST's have been produced and marketed, but is has generally been found that the fabrication of the two devices on the same substrate led to incompatible processing steps. The major problems in process compatibility have now been solved and stable complementary MOS pairs are being produced which show excellent potential for use in high speed, low power integrated circuits." Signetics reported a lift test of 800 hours at 125°C resulted in less than 50 mv drift in either P-MOST or N-MOST (threshold voltage).

Westinghouse presented a paper entitled "Integrated Complementary MOS Circuits" by J. C. Tsai, H. W. Van Beek, C. C. Roe and F. Schliesing where they reported "Life test data shows that both type MOS transistors were stable after temperature bias tests at 150°C for a few thousand hours."

Autonetics has successfully fabricated both N-channel and P-channel MOS transistors utilizing Silicon-on-Sapphire. This technology appears to offer the possibility for at least as good if not greater reliability than bulk MOS technology. Various circuits functions fabricated at Autonetics out of bulk MOS technology have logged a substantial number of hours. These hours have been on bulk MOS wafers in demonstration equipment and on life test. Only a very few failures have occured, as a result this limited information has shown a failure rate between 2% and 4% per 1000 hours. These rates should go down substantially as more data is accumulated since the portion of equipment made from newer devices have shown no failures to date. These rates should also be substantially reduced as life test results are fed back to the processing labs in an attempt to improve reliability.

The work done by the above three companies shows the interest in complementary MOS circuits. This interest will provide the incentive for developing a mature production process for LSI complementary MOS circuits in the next few years. The fact that stable P-channel and N-channel MOS transistors are being made in the laboratory today shows that the reliability problems which have beset MOS field effect transistors in the past are being solved. Six to eight years should be more than sufficient for maturing the batch fabrication techniques to make LSI complementary MOS arrays.

6.1.2.1.2 Organizational Considerations

The organization of the memory has been chosen to enchance the reliability of the system. This is done by minimizing the number of external connections in the memory system and the number of leads on each array of cells by using coincident selection rather than a linear selection technique. For an example, the number of leads on an array utilizing coincident selection will be compared to those on an array utilizing linear selection. For convenience, assume the array contains a matrix of 60 by 72 memory cells with the appropriate decoding and output circuits included in

the array[1]. Assume that the array is to be used in a 4, 320 word 18 bit memory. (This example is of course for a memory slightly more than a third of the size needed for the multiprocessor main memory.)

Item	Coincident Select Array	Linear Select Array
Input leads for addressing	13	6
Input lead for control	2	2
Power supply	2	2
Input lead for data	1	18
Output leads for data	1	18
Total per array	19	46

For this example, 18 arrays are required in either case to make a 4, 320 word 18 bit memory. Note that the coincident select array is organized as 4, 320 words of one bit whereas the linear select array is organized as 240 words of 18 bits. This is a decided advantage for the coincident select system because no additional logic gating is needed at the array outputs. There are only 18 output leads from the 18 coincident select arrays and each lead provides one bit of the 4, 320 words. In the case of the linear select system, there are 18 arrays each having 18 output leads, making a total of 324 outputs leads. Since a selected word could come from any one of the 18 arrays, an 18 input "or" logic gate is needed for each of the 18 bits in the word. Because these "or" logic gates have so many leads, it may not be practical to integrate them in one or two packages. Using 9 packages each with two 18 input gates and 40 external leads would be reasonable for a rough comparison here although improved future packaging methods may make many more than 40 leads per wafer practical. The resulting comparison of the two systems in the example is shown in the following table:

	Coincident Select System	Linear Select System
Number of LSI memory arrays	18	18
Number of LSI logic arrays	none	9
Number of leads per memory array	19	46
Number of leads per logic array	--	40
Total number of leads	342	1, 188
Total number of arrays	18	27

(1) 60 by 72 is convenient for a linear select organization since it allows an even number of 18 bit words to be fit into an array.

Many small differences between the two systems have not been included, but they should not change the results shown above very much.

For the reasons outlined above, a coincident selection technique is recommended for the 12K-18 bit multiprocessor semiconductor memory modules.

6.1.2.1.3 Explicit Memory Organization

Starting with the basic memory cell, a description will be given of a cell, an array, a module subassembly, and finally a memory module. Figure 6-8 shows the schematic of a conventional complementary MOS bistable circuit without any provisions for reading or writing. Referring to the schematic of Figure 6-8 note that Q_1 and Q_2 are never both on simultaneously except for a very short transient time during a change in logic states. Since one of these two MOS transistors is always off in a standby mode, the only current drawn from the supply voltage is the leakage current of the "off" transistor. In one logic state, Q_1 and Q_4 are on, Q_2 and Q_3 are off. In the other logic state, Q_1 and Q_4 are off, Q_2 and Q_3 are on. There are two leakage current paths in this basic memory cell, one through Q_1 and Q_4 and the other through Q_3 and Q_4. Addition of read and write circuitry will add at least one more leakage current path, making a minimum of three per memory cell.

The magnitude of the leakage current in each of the three paths depends upon the specific processes and type of isolation being used, i.e., silicon-on sapphire, and the junction temperature. Specifically, the leakage current depends primarily on the area of the p-n junction, the lifetime of the minority carriers in the vicinity of the p-n junctions, and the junction temperature. For MOS transistors fabricated on silicon-on-sapphire with a 1973-1975 mature process, the estimated nominal leakage current per memory cell is 0.1 ηA at 25°C. A value of 2 ηA per cell, at 25°C, is assumed for all calculations. This is 20 times the estimated value and still leaves total memory leakage current very small. Because of the fact that leakage current is approximately doubled for every 10°C rise in junction temperature, it is recommended that the ambient temperature of the multiprocessor be maintained at 35°C or less. This is practical since the power dissipation of a memory module is estimated as only 0.47 watts during continuous operation and 5 milliwatts during standby at 25°C. Keeping the ambient temperature low should also enhance the reliability of the system.

A block diagram of a possible memory cell design is shown in Figure 6-9 along with a truth table.

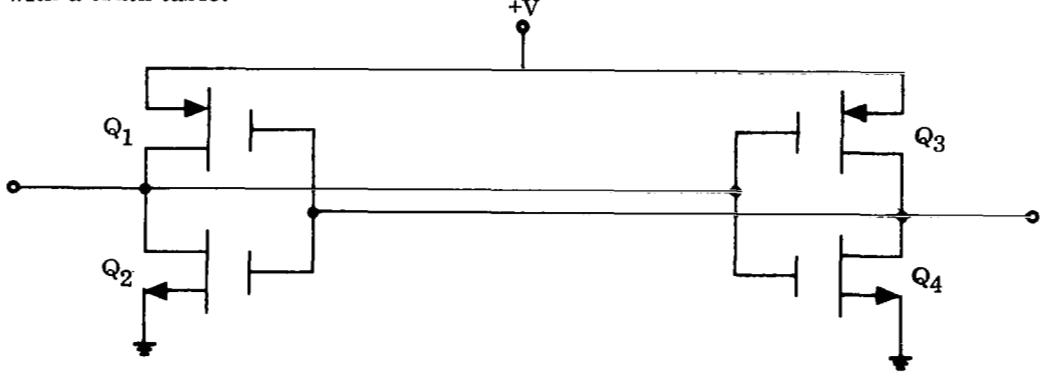
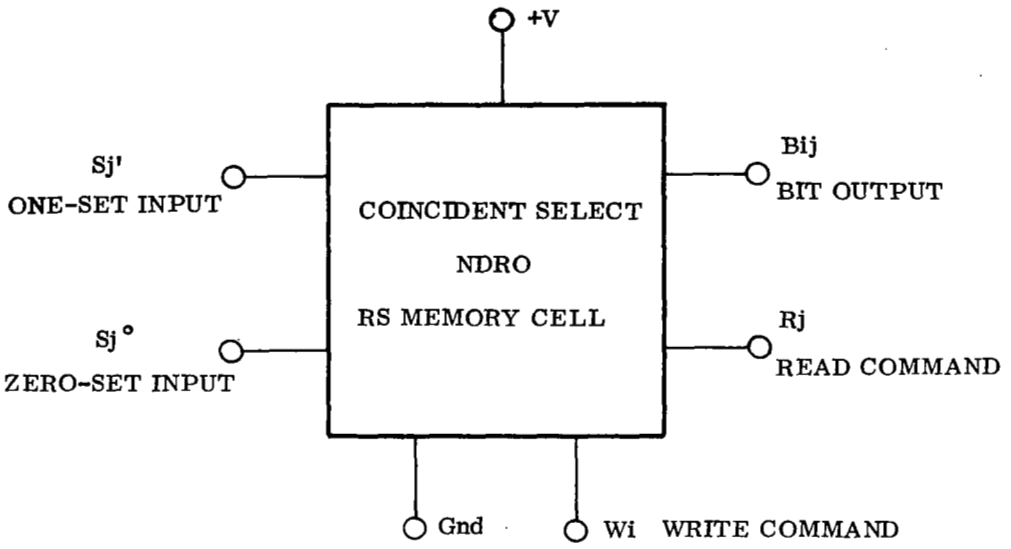


Figure 6-8. Basic Memory Cell Utilizing Complementary MOS Transistors Without Selection or Readout Provisions



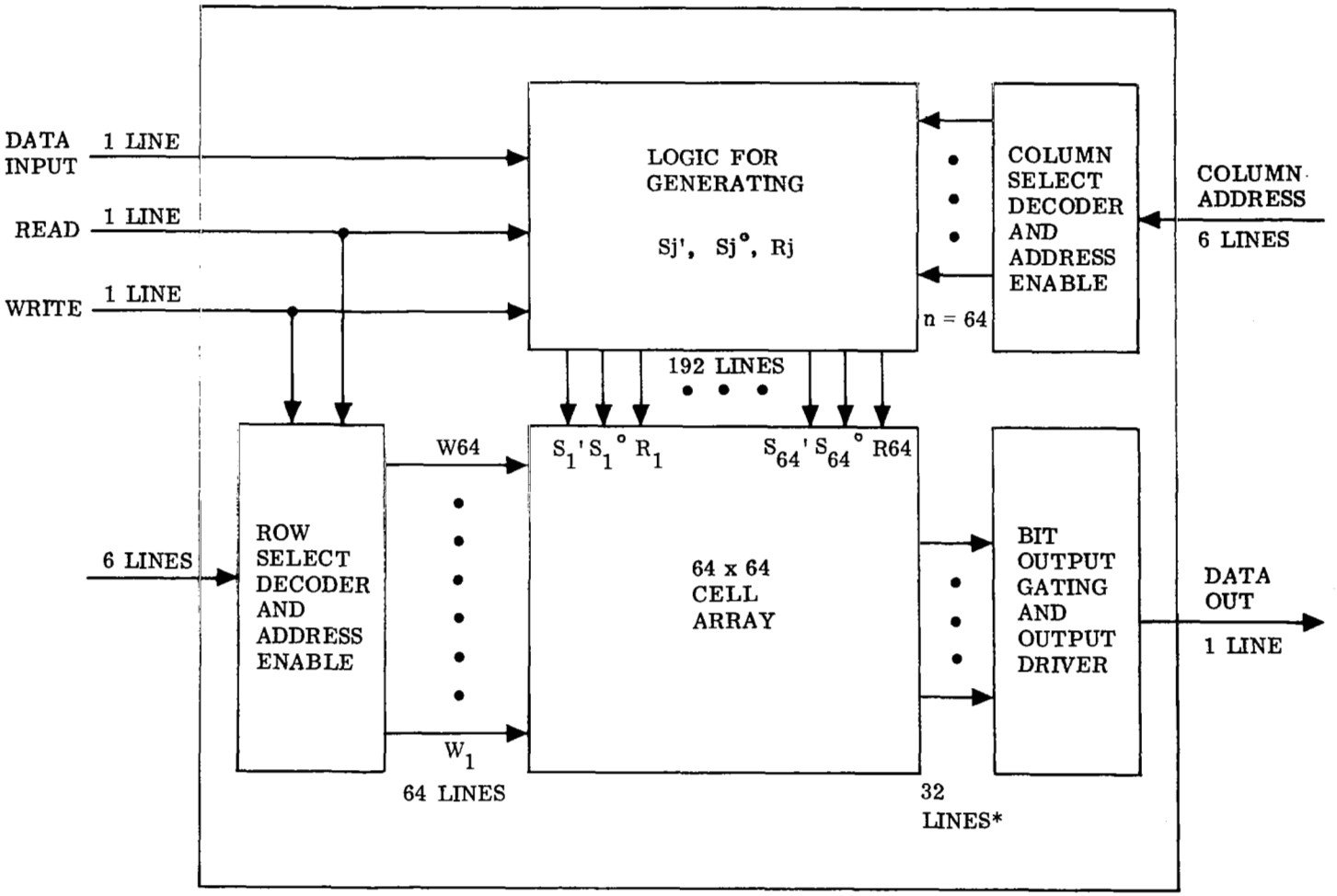
FUNCTION OF ij CELL	Rj	Wi	Sj'	Sj°
NOT SELECTED	0	0	0	0
NOT SELECTED	0	0	0	1
NOT SELECTED	0	0	1	0
NOT SELECTED	0	0	1	1
NO CHANGE	0	1	0	0
WRITE "O"	0	1	0	1
WRITE "1"	0	1	1	0
NOT ALLOWED	0	1	1	1
NOT SELECTED	1	0	0	0
NOT SELECTED	1	0	0	1
NOT SELECTED	1	0	1	0
NOT SELECTED	1	0	1	1
READ	1	1	0	0
NOT ALLOWED	1	1	0	1
NOT ALLOWED	1	1	1	0
NOT ALLOWED	1	1	1	1

Figure 6-9. Logical Operation of a Coincident Select Memory Cell

The organization of a memory cell array is shown in Figure 6-10. The array has 18 external leads as shown. The physical size of the array would be approximatel y 1.2 inches by 1.2 inches by 0.05 inches. The matrix of 64 by 64 memory cells would probably be made by starting with a matrix of 100 by 100 cells, testing the cells to determine the good/bad cell location pattern and then using discretionary wiring techniques to connect a 64 x 64 matrix of good cells. Discretionary wiring techniques will probably be necessary since reasonable yields on this complex of a circuit may be difficult to obtain. All cells, decoding gates, logic gates and output gates would be made simultaneously with batch fabrication techniques. The array is organized as 4, 096 words of one bit. A 100 x 100 cell wafer was chosen since this is consistent with the estimates of MOS/SOS device densities for the processors (5, 500 FET's per 150 mils square).

Figure 6-11 shows how 18 arrays are connected to make a subassembly containing 4, 096 words of 18 bits. Three of these subassemblies are needed to make a module of 12, 000 words, 18 bits. There are 324 interconnections in each subassembly and 52 leads either to or from each subassembly.

The organization of a memory module is shown in Figure 6-12. (Note that if expandability to three processors and three I/O units is desired, two more Input-Output units must simply be added.) Four sets of eighteen time-shared data lines are used from the module to the No. 1 processor, the No. 1 I/O, the No. 2 processor, and the No. 2 I/O. Only one processor or I/O unit can have access to the memory module at any one time. There will be 4 control lines from the memory module to each processor and I/O unit for data transfer control. In order for a unit to write an 18 bit word into the memory module, the following sequence of events must occur. The processor or I/O, assume No. 1 processor, must request a memory cycle from one of the modules. When the module is available, No. 1 processor takes control and the other units are prevented from having access to the module. (There is a simple round-robin scanner in each memory module that takes turns choosing one of the processor or I/O units.) As noted earlier in chapter 6, this means that periodically a processor or I/O may have to wait as long as 14 μs before getting access to a memory module. The frequency of waits can, of course, be minimized by good program placement. After gaining control of the module, a 14 bit address word and a write command are transferred from the No. 1 processor to the address register in the memory module. This operation is enabled by one of the two lines from the data transfer control to the input/output module. These lines enable transmitting or receiving to and from each processor or I/O unit. The appropriate 4, 096 word subassembly is enabled to receive an address by means of two lines going from the Input Data Control and Address Register to each of the three sublines going from the Input Data Control and Address Register to each of the three subassemblies shown in Figure 6-12. (00-disable sub-assembly; 01-enable read; 10-enable write). Twelve of the 30 lines going to the three subassemblies transmit the address of the selected word within the subassembly. The above address setup operations occur in less than 500 ns. The #1processor can now transfer the 18 bit word to be stored in the selected location. As presently shown, the No. 1 processor transmits the word through the input-output gates, through the input data control gates, and to the selected location in the subassembly. Eighteen of the 30 lines going to the three subassemblies transmit the data. The No. 1 processor is required to transmit the data during the full write time which also takes one bit time of 500 ns. The write cycle time is then less than 1 μs (approximately 650 ns). This means the second half of the 2μs memory cycle is spent in standby. The processor is then able to use the two bit times left in the memory cycle to prepare for the next cycle.



*THESE ARE NECESSARY (INSTEAD OF ONE LINE) DUE TO CAPACITANCE LOADING

Figure 6-10. Organization of a Coincident Select Memory Cell Array

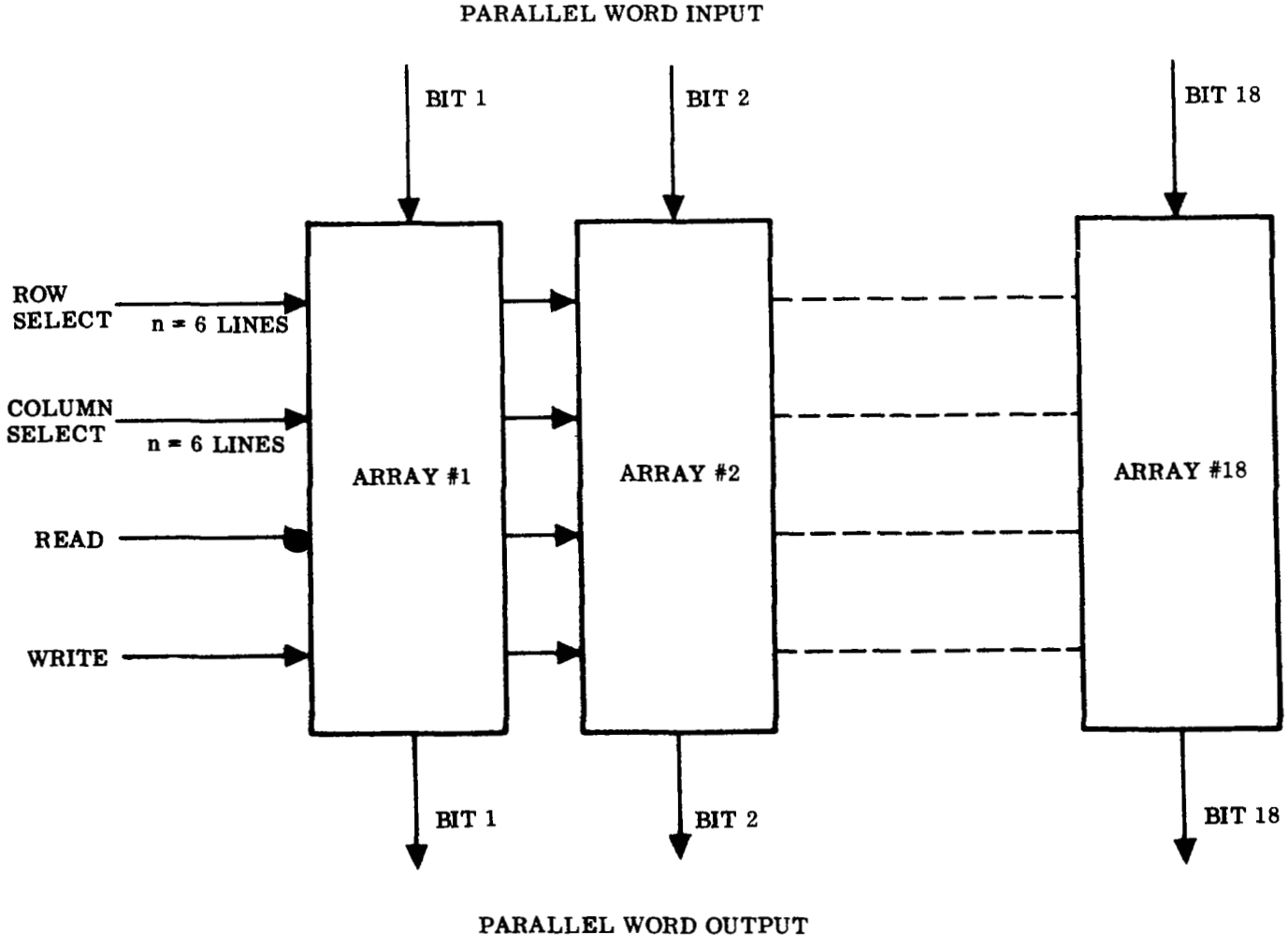


Figure 6-11. Connection of 18 Arrays to Form a 4, 096 Word, 18 Bit, Subassembly for a Memory Module

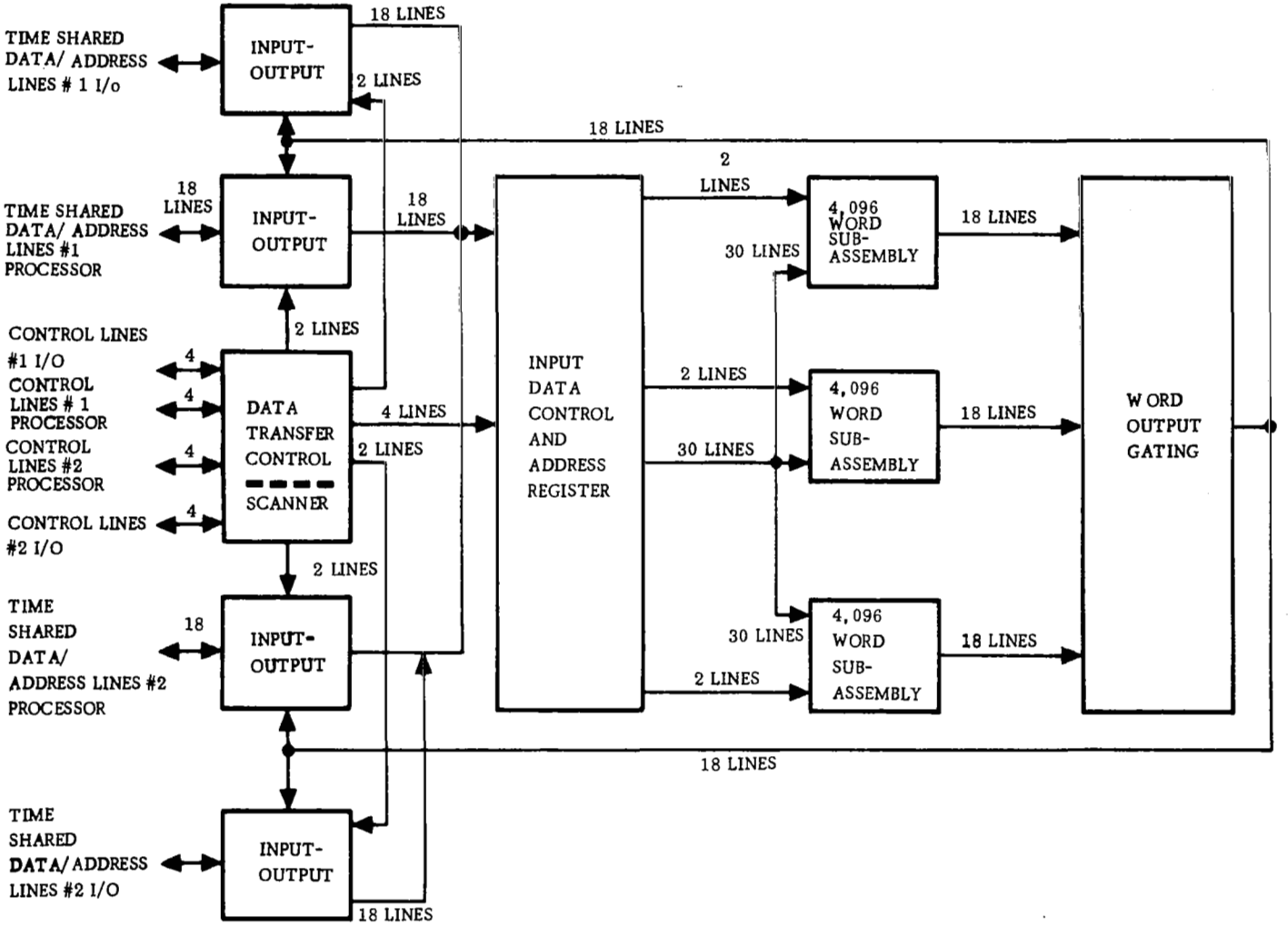


Figure 6-12. Organization of a 12,000 Word, 18 Bit, Memory Module

To read a word in the memory module, the #1 processor must again gain control of the module. As for writing, a 14 bit address word and a read command are transferred to the address register. Since the selected word could be in any of the three subassemblies, a 3-input "or" gate is required for each of 18 bits in a word. These gates are shown in Figure 6-12as Word Output Gating. The selected word is transferred to the Input-Output block, as shown in Figure 6-12, and is setup on the data lines to processor No. 1. At the end of bit time two the processor strobes the word into its memory buffer register. Therefore the read operation occurs in one μs.

The estimated number of arrays in a memory module is given in the following table.

Function	Number of Arrays Per Module	Number of External Connections
Input-Output to #1 processor	2 (1)[1]	60
Input-Output to #2 processor	2 (1)	60
Input-Output to #1 I/O	2 (1)	60
Input-Output to #2 I/O	2 (1)	60
Data Transfer Control	1	30
Input Data Control and Address Register	2	60
Subassembly #1	18	52 + 324[2]
Subassembly #2	18	52 + 324
Subassembly #3	18	52 + 324
Word Output Gating	2	76
Total	67	562 + 962

(1) The number in parenthesis may be more representative of future packaging methods. In fact the I/O units and data transfer control may well be combined into one array. This would put no strain on element densities within the array. If a packaging technique with 100 to 150 connections could be developed the total number of arrays would drop to 56.

(2) Number of connections internal to a subassembly. From the above it can be seen that each memory module has an estimate of about 1500 connections external to the array.

6.1.2.1.4 Volatility Circumvention

The suggested power distribution, voltage regulation, and volatility circumvention for each memory module is shown in Figure 6-13. If a short occurs inside a module, short circuit protection is needed to prevent this short from causing a power failure for the other modules. In the case of an input short circuit, the diode D and transistor Q prevent a reverse power flow through the voltage regulator and a consequent power failure to one or more modules.

For a standby power source, it is recommended that a hermetically sealed nickel-cadmium rechargeable battery be used. Assuming a nominal standby current of 0.5mA per module, the standby current for the three modules is 1.5mA at 25°C and 3mA at 35°C. A battery having an ampere-hour capacity of 0.45 A.H. could supply standby operation for 300 hours at 25°C or 150 hours at 35°C. A hermetically sealed high reliability nickel-cadmium battery electrically similar to a Sonotone S-101 or an Eveready BH450 would provide the following characteristics:

Initial cell voltage	1.45 volts
Nominal cell voltage	1.25 volts
Endpoint cell voltage	1.10 volts
Nominal ampere-hour capacity	0.45 A.H.
Volume per cell	0.42 cu. in.
Weight per cell	0.80 ounces

A battery of eight cells in series would have the following characteristics.

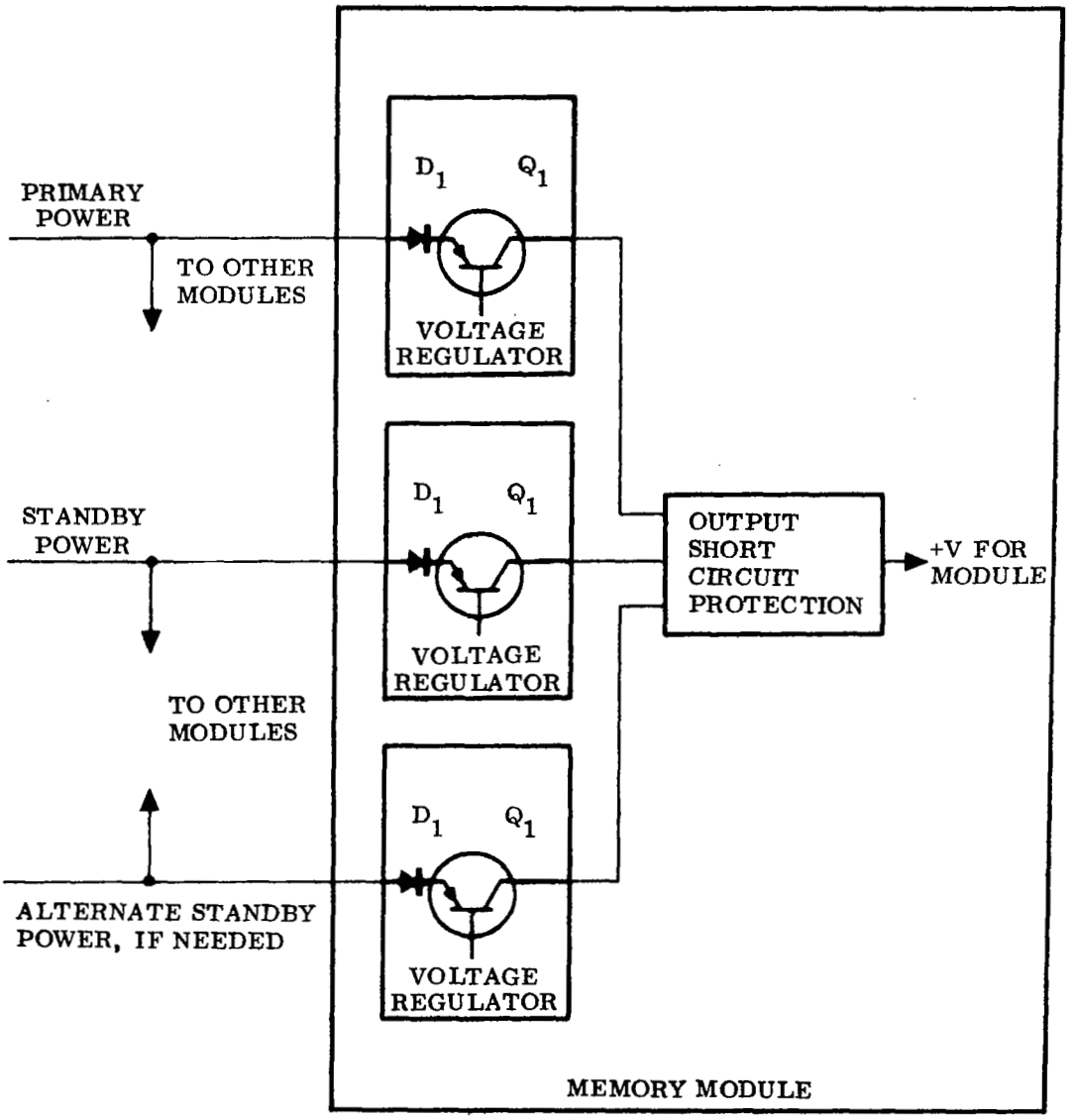
Initial voltage, 8 cells	11.6 volts
Nominal voltage, 8 cells	10.0 volts
Endpoint voltage, 8 cells	8.8 volts
Nominal ampere-hour capacity	0.45 A.H.
Dimensions	1.2" x 2.4" x 2.4"
Volume	7 cu. in.
Weight	8 ounces

Concerning the life expectancy, the "Eveready" Battery Applications and Engineering Data manual states that "Cycle life of the nickel-cadmium sealed cell depends upon the way it is used. The factors affecting life expectancy are:

Amount of overcharge

Depth of discharge

Temperature of charge



NOTE:

D_1 AND Q_1 SERVE AS REDUNDANT INPUT SHORT CIRCUIT PRODUCTION.
Q_1 IS A SERIES REGULATING TRANSISTOR IN THE VOLTAGE REGULATOR.

Figure 6-13. Memory Module Volatility Circumvention

Temperature of overcharge

Temperature of use

A cell which is discharged through only a fraction of its full capacity on each cycle will give many more cycles than a cell which is fully discharged each time. Under conditions of very light or casual service, the expected life is several years. Concerning trickle charging, the Eveready manual states, "A trickle charge is a continuous constant current charge given to a battery to maintain it in a fully charged condition, with no external load connected to it. This may be used for batteries in storage, or in standby service where their use is in an emergency such as failure of the normal power supply." This says that the trickle charged standby batteries in this system should offer high reliability since they are used under optimum conditions.

Gulton Industries, Inc., Metuchen, New Jersey, has published reliability data on some of their hermetically sealed high reliability batteries designed for space applications. For their VO-12HS cell, they report 2, 017, 360 cell-hours of operation with no failures. This data is from the Orbiting Geophysical Observatory Program and is accumulated from life tests, cell evaluation tests, battery development and acceptance tests, spacecraft testing, and battery storage. The data shows a demonstrated failure rate of less than 0. 12% per thousand hours at a 90% confidence level. This is for a large 12 ampere-hour cell. Based on this data, it is assumed that 8 cells having a 0. 45 ampere-hour capacity each will have, in 1973-1975, a failure rate less than 0. 2% per thousand hours at a 90% confidence level.

It should again be noted here that these extra batteries may not even be necessary if a reliable redundant power supply is employed for the spacecraft. The same circuitry as shown in Figure 6-13 would be used with totally isolated primary and standby power lines from the central spacecraft supply.

6. 1. 2. 1. 5 Power Calculations

In order to get a feeling for the memory power dissipation a rough estimate was made of power dissipation during operation. The operating power dissipation of a memory cell array is estimated as follows:

1. Assume a "1" is to be written in a cell.

2. The greatest supply current will be drawn by the array when the W_i line and the S_j line are being charged. This is because of their relatively high capacitance which must be charged.

3. Assume the W_i line has a capacitance of 10pf, the S_j' line has 10pf, the supply voltage is 6. 0 volts and the voltage rise time is 100 nanoseconds.

4. From these estimates,

$$I_s = C \frac{dv}{dt} \text{ per line}$$

$$= C \frac{\Delta v}{\Delta t} \text{ per line}$$

$$= 10 \text{ pf} \frac{6 \text{ volts}}{100 \text{ nanoseconds}}$$

$$= 0.6 \text{ ma per line}$$

$$2I_s = 1.2 \text{ ma}$$

5. To allow for other switching transient currents, such as decoding and output gating, assume the 1.2 ma flows continuously for the 1 μs write period rather than for 100 nanoseconds. This gives a conservative estimate of 7.2 mw per memory cell array. This value will not vary significantly with ambient temperature.

6. The calculations of power dissipation are made using 10 mw for either a logic array or a memory cell array.

7. Because each of the memory modules has a supply voltage regulator, assume the input voltage to the module is 10.0 volts, with a 4 volt drop across the series regulator. This will increase the power dissipation by 2/3. A relative high voltage drop is needed across the regulator because, during standby operations, the battery voltage will drop to the endpoint voltage of 8.8 volts. The regulator can maintain 6.0 volts output with 8.8 volts to the input of the regulator.

A summary of the operating and standby power dissipation is given in the table below.

Item	Standby Power Dissipation, 25°C	Standby Power Dissipation, 35°C	Operating Power Dissipation
Array	80 W	160 μW	17 mW
Module	5.4 mW	10.7 mW	0.53 W(1)

(1) Referring to Figure 6-12 note that only one of the three subassemblies is operating at any one time. The other two are on standby. Therefore 31 arrays are dissipating power.

6.1.2.1.6 Reliability Calculations

The failure rate goal for a memory module is 1.6% per thousand hours. For a 3 module memory, the goal is 4.8% per thousand hours, as stated previously. An estimate of the failure rate goal for an array of 64 by 64 memory cells is determined below. Note that the LSI arrays for performing the logic functions in a memory module are included as equivalent to a memory cell array in determining the failure rate goal of an LSI array.

Item	Failure Rate Per Item	No. Per Module	Failure Rate Per Module	Failure Rate For 3 Modules
Connections	0.00001%	1500	0.015%	0.045%
8 cell battery, 0.45 A.H.	0.2%	2/3 (1)	0.13% (1)	0.40% (1)
Power supply circuits	-	1	0.02%	0.06%
LSI logic and memory arrays	0.0213%	67	1.43	4.29

(1) This assumes 2 standby power sources for 3 memory modules. Only one may be required, or the central spacecraft supplies may be used.

Based on the above failure rate apportionments, the array must have a failure rate no greater than 0.0213% per thousand hours. This is a reasonable goal for 1973-1975. This goal is believed to be reasonable because of the fact that stable complementary MOS devices are being made in the laboratory today and there is 6 to 8 years to mature this technology. As an example of what can be accomplished in 6 years, examine the demonstrated increase in the reliability of discrete bipolar transistors from 1960 to 1966. In 1960, the manufacturers demonstrated failure rate for a small signal NPN silicon transistor was 1% per 1000 hours. This was a transistor used in the Minuteman computer. In 1966, a small signal NPN silicon transistor in the Minuteman computer has demonstrated a failure rate of 0.0062% per thousand hours. This shows better than two orders of magnitude increase in reliability. If the present rough experimental estimates are close to correct, approximately a two orders of magnitude increase on the 2% to 4% per 1000 hours given earlier for a bulk MOS array would meet the needs of the memory arrays for this system.(1) Another reason for expecting the goal to be attained is that the industry is considerably higher on the learning curve for the silicon planar technology in 1967 as compared to 1960. This means that significant improvements can be made in less time. The array also has two features which enhance the reliability. One is the low number of external leads on the LSI array package, 18 in the case of the high usage memory cell array. The other is the very low power dissipation of an array.

6.1.2.2 Magnetic Memory

6.1.2.2.1 Introduction

This section discusses the selection of the magnetic memory for use in the 1975 technology time frame. The memory system requirements (the same requirements as given in Paragraph 6.1.2.1.), i.e., random access, read write at 2 μsec cycle time, reduce the types of memories to be considered to a coincident current DRO core structure and a multiword organized batch fabricated NDRO structure (e.g., plated wire today). Under today's technology, the selection of the most reliable approach is simply a matter of selecting the approach which uses the least number of operational circuits. However, with the advent of LSI techniques and the

(1) It should also be remembered that the 2% to 4% per 1000 hours was obtained from minimal data with almost no failures.

capability of fabricating multifunction circuitry, the problem must now also be viewed in light of what approach is most ammendable to LSI techniques. This point is made since the coincident current approach contains the least number of operational circuits for today's structures but because of the drive levels (both present and projected) appears to be less ammenable to LSI. Therefore today's choice for the most reliable memory would be a DRO core structure; however the choice made here for 1975 is a batch fabricated NDRO structure. This choice has been made for the following four reasons. The reasons are listed in order of importance for this application:

1. The NDRO structure will take maximum advantage of LSI techniques in the 1973-1975 time frame; whereas the DRO core structure does not appear to have the same potential. The NDRO approach therefore offers higher reliability both due to the increase in circuit reliability (less circuits) and the decrease in the number of connections in the system.

2. The NDRO structure has less sensitivity to transients due to the fact that it does not require a restore cycle.

3. The NDRO structure will be batch fabricated. This offers the opportunity to institute effective on-line quality control and reliability improvement programs. These programs will feedback on the production process in order to modify it to produce more reliable devices.

4. The NDRO structure offers lower power operation than the DRO structure both due to higher speed operation and the ensuing lower duty cycle and to the lack of the need to regenerate after a read cycle.

To illustrate the above statements, discussions and basic block diagrams for both approaches shall be presented. Finally, the multiword NDRO structure shall be detailed (i.e., LSI circuits will be projected and reliability figures assigned) so that a quantitative understanding of the approach and its ability to meet the multiprocessor system requirements may be gained.

The two approaches shall first be described in light of today's technology in order to understand the effects of 1975 technology.

6.1.2.2.2 Present Day Memories

This section shall describe the coincident core memory and the NDRO multiword memory on the basis of today's technology.

Coincident Core Memory

A Coincident Current (CC) structure is shown in Figure 6-14. The CC approach under today's technology has certain organizational constraints which reflect in the system organization. These constraints are as follows:

1. Sense Lines are restricted to 4K elements due to line delay and attenuation, and noise considerations.

2. Bit lines are restricted to 4K elements due to time delay and noise effects.

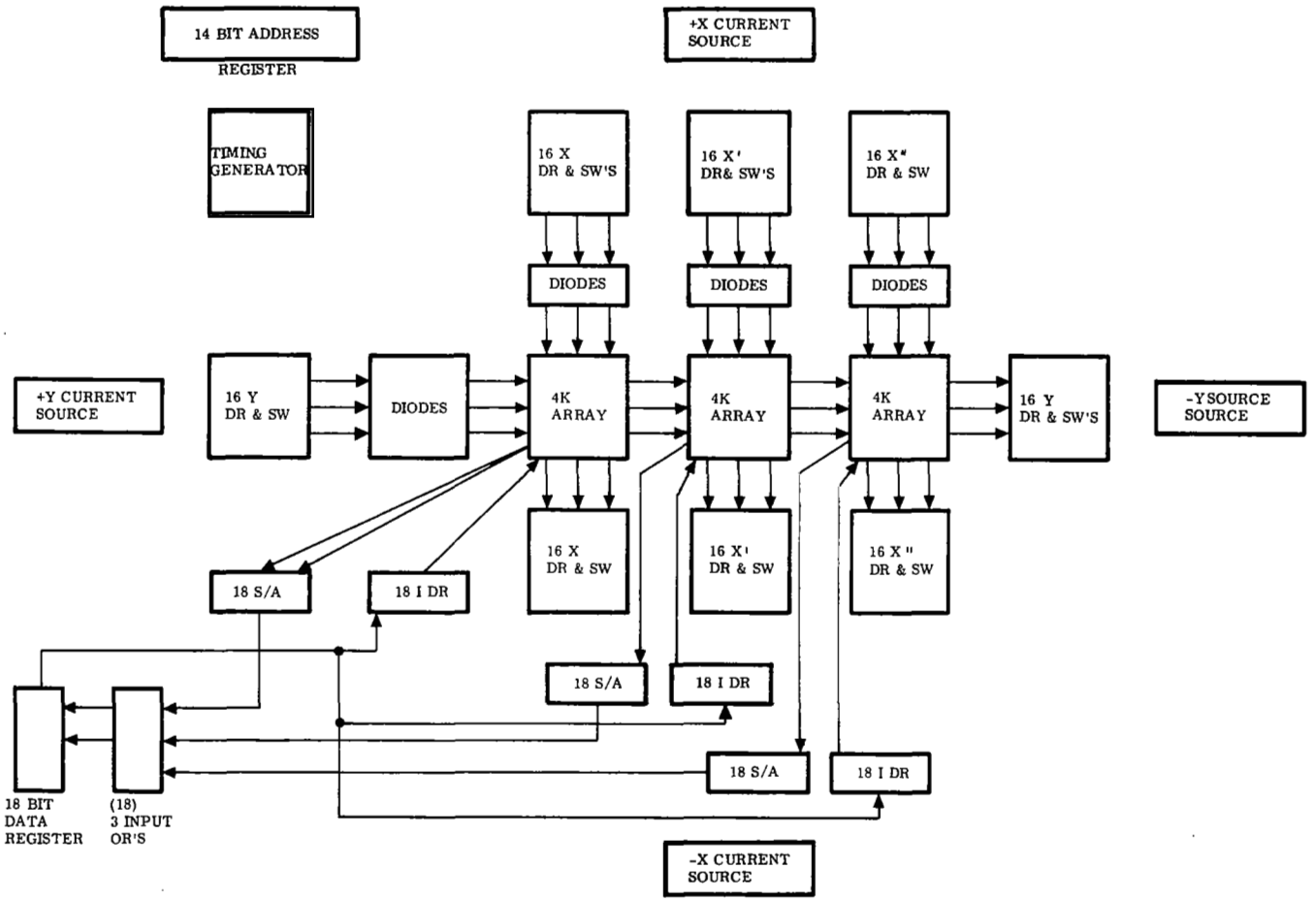


Figure 6-14. 12x18 3D Memory (Todays Technology)

Active Circuits

Word Switches - The drive matrix consists of switches arranged in matrix fashion so that unique word selection is made by selecting two X switches (one on each side of the array) and two Y switches. The number of word switches required = 128 total. (See Figure 6-14). The switches are packaged four per IC with nine connections.

Word Diodes - Discrete diodes are required to isolate the drive electronics. The total required = 512 per module. These are packaged as sixteen per IC with ten connections.

Word Gates - Gates are required to use the address information to select the appropriate X and Y switches. For the organization shown 1 gate per switch is required, Total = 128. They are packaged as two per IC with twelve connections.

Inhibit Drivers - The number of inhibit drivers required is equal to the number of bits times three (the number of 4K arrays) or 18 x 3 = 54. These drivers are packaged separately due to power limitations. Each package has six connections.

Inhibit Gates - The drivers must be selected, depending upon data input, and clocked; as a result each driver requires an input gate. These are packaged with the driver.

Current Sources - Positive and negative X & Y currents are required for a system total of 4. Each source requires approximately 10 IC's with ten pins per IC.

Timing Generator - This generator is required to generate the sequence of timing pulses for the read and write cycles. It requires approximately 10 IC's with five pins per IC.

Sense Amplifiers - The number of S/A required is equal to the number of bits times 3 or 18 x 3 = 54. These are packaged separately with ten connections.

S/A Gates - The three sets of sense amplifiers must be gated into the common data register. Total requirements = 18-3 input "or" gates packaged four to an IC with 14 connections.

Data and address registers along with some interface circuitry must also be included; however, this is not dependent on the memory technology. Therefore, it will not be discussed here.

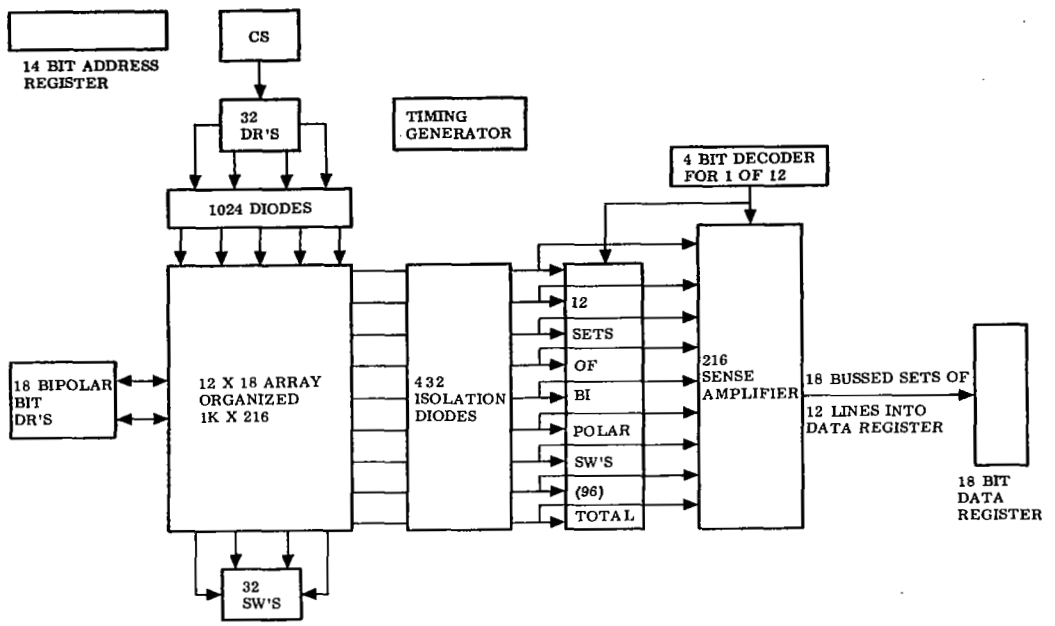


Figure 6-15. NDRO Multiword Memory (Today's Technology)

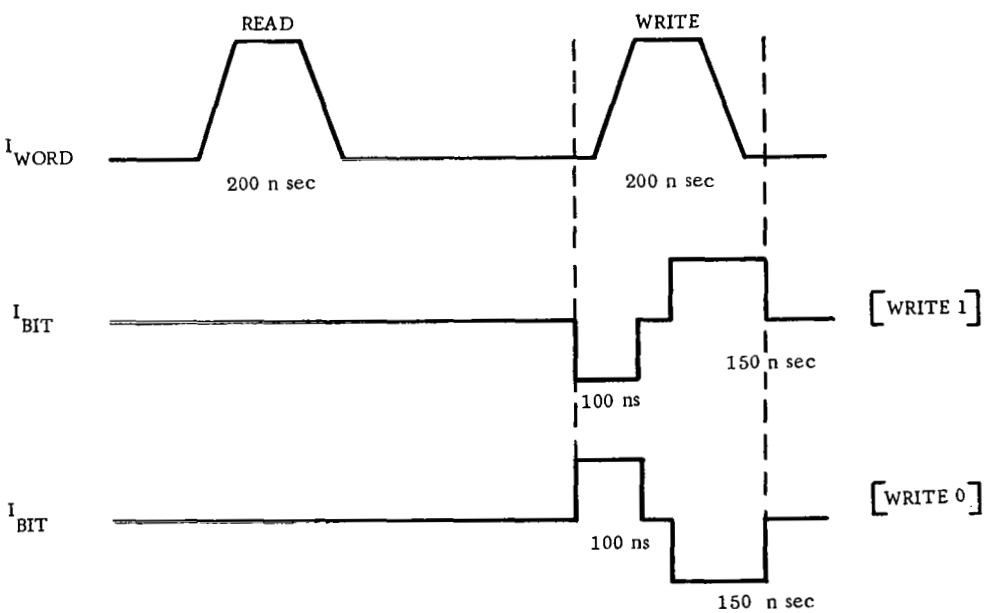


Figure 6-16. NDRO Read and Write Signals

A summary of the coincident current electronics is given below:

Type Circuit	CKTs	IC	Connections
Word Switches	128	32	288
Word Gates	128	64	768
Word Diodes	512	32	320
Inhibit Driver & Gates	54	54	324
S/A's	54	54	540
S/A Gates	18	5	70
Current Sources	4	40	400
Timing Generator	1	10	50
TOTAL		291	2760

This table will be compared in the following section to a similar table for a plated wire NDRO memory.

NDRO Multiword Memory

A good example of an NDRO multiword structure in today's technology is a plated wire memory as shown in Figure 6-15. A multiword structure (more than one 18 bit word on a word line) is used since it saves a considerable amount of electronics. Such structures have the important property that single words on a multiword line can be written into without disturbing the other words on the line.

The optimum multiword organization for the 12K by 18 memory module is one K word lines of 216 bits or 12 words. The two permutations around 1K by 216 (i.e., a) 2K by 108, and b) 512 by 432) are ruled out for the following reasons:

1. The sense/bit line is limited (by bit drive and sense line delay considerations) to 1K elements; hence a duplication of bit drivers and S/A's would be necessary for a 2K by 108 structure.

2. The word line length is limited to about 240 elements by line inductance and delay considerations; thus a duplication of hardware would also be necessary in a 512 by 432 structure.

3. It is also the case that the total amount of circuitry is minimized by using 1K by 216.

This structure is shown in Figure 6-15. The theoretical operation of thin film NDRO magnetic memories is well discussed in the literature and as a result will not be presented here; however, Figure 6-16 gives a diagram of the read and write signals. It should be noted that in todays multiword film structure bipolar bit write signals are required to overcome skew and dispersion effects. As a result the write

cycle time is increased over the read. Future film structures should be of higher quality and thus be able to take advantage of unipolar write signals. One additional useful feature of the multiword devices under consideration, such as plated wire, is that they only require unipolar word line drive. This not only saves circuits but also connections.

Active Circuits

Word Switches - The drive matrix is organized so that unique word selection is obtained by selecting two switches (one on each side of the array). This requires 32 unipolar switches on each side. Total switches = 64. The switches are packaged four per IC with nine connections.

Word Access Diodes - One diode per line for isolation of the unipolar drive electronics total diodes = 1024 per module. These are packaged as eight per I/C with nine connections.

Word Gates - As for the CC memory, one gate per switch is required for selection. Total Gates = 64. They are packaged two per IC with twelve connections.

Word Current Source - The lines are accessed by the 64 matrix switches and the current drive is steered to the selected lines. This requires just one central current source. This source requires approximately 10 IC's with ten connections per IC.

Bit Drivers - There must be one bipolar bit driver for each of the 18 bits in a word. This gives a total of 18 per module. They are packaged as one bipolar driver per IC with 10 connections.

Bit Switches - Since the memory is organized with each word access line containing twelve eighteen bit words, there must be a selection of one of twelve words for the bit drivers to write into. This selection is carried out by bit switches. Ideally only one bipolar bit switch would be required per word; however, with the present plated wire memories, the 100 ma I_B requires four bipolar return switches per word to sink the bit current. Total bipolar bit switches = 48. These are packaged three per IC with 12 connections.

Bit Driver Gates - The bit drivers must be selected depending upon data input and clocking for the write operation. They are not used in the read operation. Each bit driver requires one selection gate for a total of 18 gates. These are packaged as two per IC with 10 connections.

Sense Amplifiers - The sense amplifiers are relatively simple so that one will be connected to each bit line. A one of twelve decoder will then choose the correct eighteen sense amplifiers, Total S/A = 216. They are packaged separately with 10 connections.

Timing Generator - This generator produces the timing pulse for the read and write cycles. It requires 10 IC's.

Type CKT	CKTs	IC	Connections
Word Sw's	64	16	144
Word Gates	64	32	384
Bit DR's	18 (36)	18	180
Bit Gates	18	9	90
Bit Sel Sw's	48	16	192
Current Source	1	10	100
Timing Generator	1	10	50
S/A's	216	216	2, 160
Word Diodes	1024	128	1, 152
TOTAL		455	4, 462

The circuit and connection totals given here for an NDRO memory should be compared to those given earlier for a DRO memory. From this comparison it can be seen that a DRO core structure should be more reliable today than a NDRO plated wire structure since the latter structure contains more electronics and connections. The power dissipation of the two memories is very similar and ranges from 35 to 40 watts; however, over half the power dissipation in the NDRO structure is from stand-by power on the sense amplifiers. This dissipation will be reduced by an order of magnitude in future systems using power strobing of the sense amplifiers. (The amplifiers are turned off for a portion of each cycle.) Of equal importance is the fact that thin film NDRO structures are expected to offer large decreases in word current, bit current, and line impedance in the future. This should enable these structures to take optimum advantage of LSI techniques in the 1973-1975 time frame. For the above reasons a thin film, multiword NDRO structure is chosen over a DRO structure for magnetic main memory in the 1975 time frame.

6.1.2.2.3 Future Memory Organization

This section describes the organization of a 12K word 18 bit NDRO memory for the main multiprocessor memory. A particular memory device is not specified, but the chosen device will be a thin film batch fabricated structure with multiword capability, (e.g., plated wire, bi-core are applicable from today's technology). In order to describe the projected NDRO memory certain assumptions about the performance characteristics and modifications to present circuit approaches are required. These assumptions and modifications are listed below.

Assumptions:

1. **The memory will be organized as one K word lines by 216 bit lines. The reasons for this are the same as those given for the current NDRO structure.**

The line length limitations may be relaxed somewhat in the 1975 time frame, but this organization still requires the minimum number of circuits.

2. One crossover per bit will be used. Orthogonal structures require common mode noise cancellation. This can be achieved in 2 ways a) 2 crossovers per bit operation or b) addition of a common mode cancellation line (dummy non magnetic element). The word line inductance is a function of the number of crossovers down a word line; so that for 2 crossovers per bit the electrical length of the word line is doubled. If the second technique is employed (i. e. dummy line) then the electrical length of the line is only increased by the ratio of active to dummy lines. For example, 1 dummy for 4 active lines gives a 20% increase in length, 1 dummy for 8 active lines gives an 11% increase in length.

The latter approach will be used in this system with one dummy line per on active lines. (n determined by noise)

3. Improvement in present thin film memories or additions of new structures should enable the following properties to be readily achieved by 1975.

   a. Unipolar write current - The signals for the read and write cycles are shown in Figure 6-17. Note the difference between this figure and Figure 6-16 where a bipolar write signal was used to overcome skew and dispersion effects. The read or write operation will take less than 500 ηsec.

   b. I_{word} = 200 ma

   c. I_{bit} = 25 ma

   d. Z_o = 30 Ω's

These properties will enable LSI circuits to be used for the memory electronics.

4. Bipolar bit currents will still be required in order to be able to write "1's" and "0's".

5. The timing and control will be packaged into one LSI chip.

Modifications for LSI

1. One word driver with base to emitter selection will be used for each of the 1024 lines. This approach will be used instead of matricizing drivers and switches because the latter approach would require a large number of isolation diodes. An LSI pack can be fabricated with a number of drivers almost as easily as with a number of diodes. As a result matricizing the switches would actually require more LSI arrays.

2. One sense amplifier per line will be used. This is again for the reason that this approach requires less LSI arrays than an approach using sense amplifiers and gates.

3. One bit driver per line will be used for the same reason as given in 1 above.

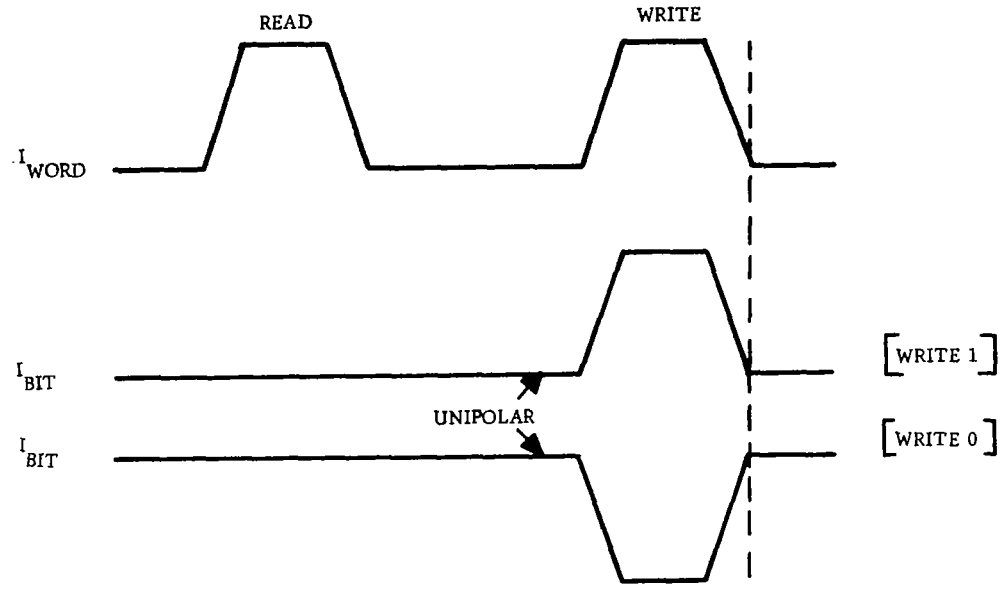


Figure 6-17. Future NDRO Read and Write Signals

The above points enable a memory structure as shown in Figure 6-18 to be specified. This structure takes advantage of LSI techniques to package a number of drivers, switches, and sense amplifiers in arrays. The packaging of these circuits in arrays was estimated for the 1975 time frame and is specified along with connections, and reliability per array in the following material. Figure 6-18 also shows input/output and control sections going to the processors and I/O units. These modules were briefly discussed in Paragraph 6.1.2.1. They will be added to the reliability calculations for this memory as 9 arrays with 270 connections and a reliability of .02% per array. The functions of this interface logic will be discussed again in the following section.

Based on the above discussions, a reliability estimation for this memory is given below:

Word Switches 16 Modules 1024 CKTS

Matrix of 64 SW's arranged 8 x 8 with 64 outputs

17 Logic Inputs 8x, 8y, 1z

2 Power

1 Current Source

This can be accomplished because only one switch is actuated at any one time.

Total Connections = 84

Anticipated Reliability = 0.02%/1000 hrs

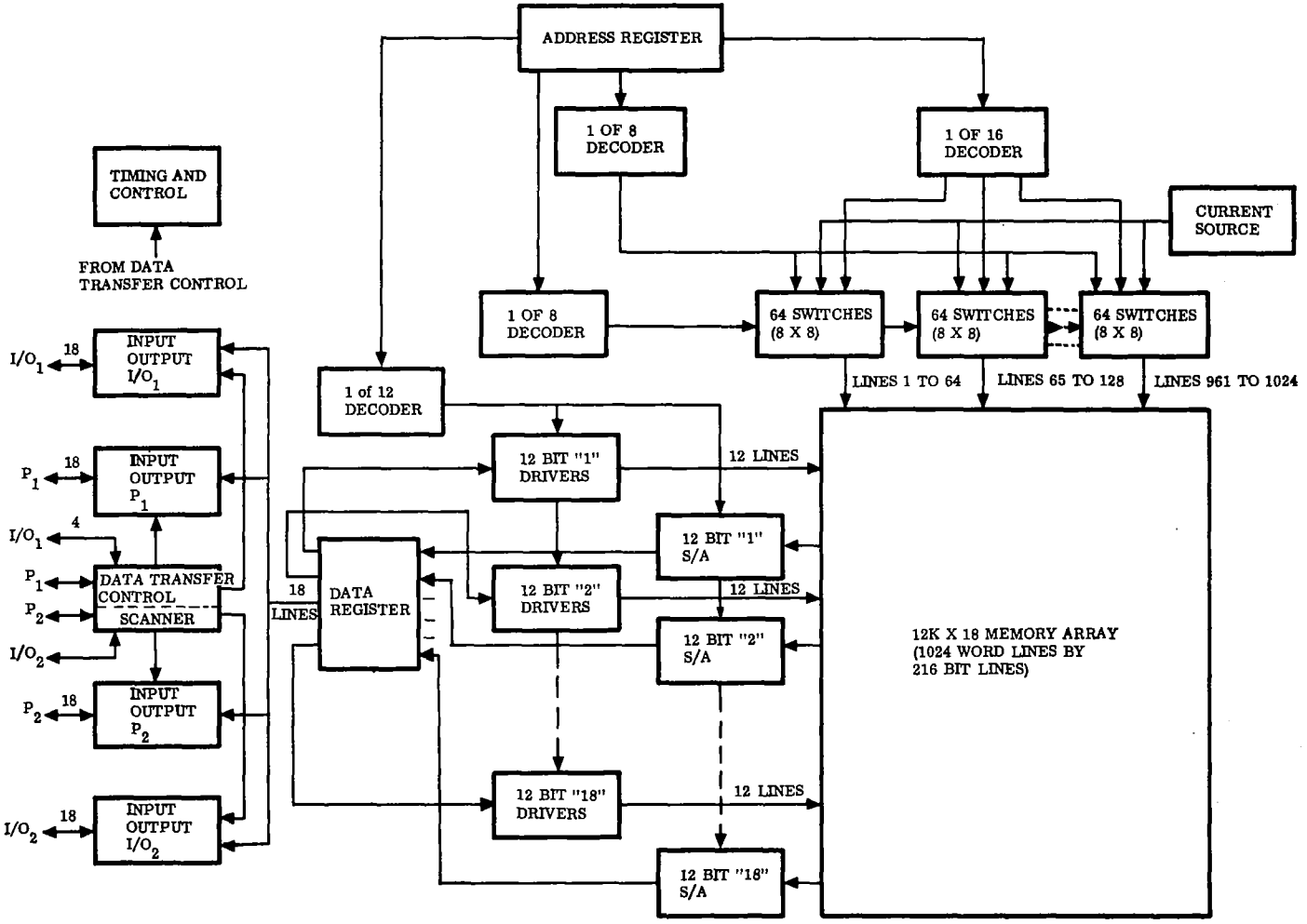


Figure 6-18. Future NDRO Memory

Bit Driver 18 Modules 216 CKTS

Module contains 12 Bipolar Bit Drivers 1 DR. per module is activated per write time

Connections· 12 Logic Line to Select 1 of 12

2 Data Lines (1's & 0's)

2 Timing

3 Power

12 Output Lines

Total Connections = 31

Anticipated Reliability = 0. 01%/1000 hrs

S/A 18 Modules 216 CKTS

12 S/A's per module

All S/A's receive signals for logic select. S/A output to be processed

12 Logic Lines to Select 1 of 12

2 Strobes

4 Power

15 Input

1 Output Total Connections = 34

Anticipated Reliability = 0. 01%/1000 hrs

Timing Generator 1 Module 20 CKTS

4 Input

12 Output

4 Power

Total Connections = 20

Anticipated Reliability = 0. 005%/1000 hrs

Address Register 1 Module 14 CKT

Inputs	15	Logic
Output	28	
Power	4	
Total = 47		

Data Registers — 1 Module

Inputs	36
Outputs	18
Timing	2
Power	4
# Connection	60

Reliability = 0.015 %/1000 hrs

Decoder A (2- 1 of 8's) — 1 Module

Input	24
Output	16
Power	4

Total Connections = 44

Reliability 0.01%/1000 hrs

Decober B (1 of 16) — 1 Module

Input	8
Output	16
Power	4
Total =	28

Reliability = 0.01%/1000

Decoder C (1 of 12)

Input	8
Output	12
Power	4
Total =	24

Reliability
0.01%/1000 hrs

Table

		L. S. I CKTS	# Connections/Mod	Reliability
Array	1		3, 000	
Word Circuits	1024	16	84 (1344)	0. 02%/1000 hrs
Bit DR	216	18	31 (558)	0. 01%
S/A	216	18	34 (612)	0. 01%
Decoders A	2, 1 of 8's	1	44	0. 01%
B	1 of 16	1	28	0. 01%
C	1 of 12	1	24	0. 01%
Address Reg	14	1	47	0. 01%
Data Reg	18	1	60	0. 015%
Timing Gen	1	1	20	0. 005%
Current Source	1	1	10	1. 0%
Logic Arrays	5	9	270 (total)	. 02%/1000 hrs
Total		68	~ 3000	

Reliability	Per Mod		x3
Array	3, 000 x 0. 00001 %	= 0. 030%/1000 hrs	0. 90%/1000 hrs
Word CKTS	16 x 0. 02	0. 32	0. 96
Bit	18 x 0. 01	0. 18	0. 54
S/A	18 x 0. 01	0. 18	0. 54
Decoders	3 x 0. 01	0. 03	0. 09
Register	1 x 0. 01	0. 01	0. 03
Register	1 x 0. 01	0. 01	0. 03
Timing	1 x 0. 005	0. 005	0. 015
*Current Source	1 x 1	1. 0	N/A
Logic Arrays	9 x 0. 02	0. 18	0. 54
Total.		1. 945%/1000 hrs	3. 835%/1000 hrs

* Current Source need not be repeated with increased Modularity

The above calculations show that the NDRO magnetic structure should be able to easily meet the reliability requirement of 4.8% per 1000 hours for the multiprocessor main memory.

A rough power calculation was carried out using the current levels given earlier under "assumptions" in order to get a feeling for the memory power dissipation. Assuming the sense amplifiers will use power strobing, this memory will dissipate about 11 watts per 12K module.

The timing of this memory with a processor is the same as described in paragraph 6.1.2.1. The NDRO magnetic memory will easily be able to setup addressing in 500ns and then read or write within another 500 ns. This will provide information to the processor by the end of bit time two.

It should again be noted here as in the introduction to the memory section that both the NDRO magnetic and semiconductor memories are able with little risk to meet all of the multiprocessor main memory requirements. The semiconductor memory uses substantially less power but it offers a slightly greater risk in being able to meet the reliability requirements in the 1975 time frame. As a result neither memory structure can be chosen at this time. Further development and investigation are necessary in order to make a valid choice.

6.1.2.3 Memory Interface Hardware

The multiprocessor memory must have a good amount of interface hardware in order to handle the communications to the processors and I/O units. The primary parts of this hardware are:

1. A six bit round-robin scanner is used to choose the processor or I/O unit to receive the next memory cycle. For the given mission only four bits of the scanner will be used since only two I/O units and two processors are present. The scanner is simply an asynchronous counter that sequences through all the memory request lines until one is found up. Since the count is done in an asynchronous fashion, it will take less than 200 ns to sequence through all six states.

2. A six bit lockout register is used to hold the I/O or processor modules that are locked out of the memory. The operation of the lockout has been explained in detail in Section IV.

3. A quantity of timing and control circuitry is included to generate the memory timing and the interface signals to the processors and memories. The operation of this hardware is explained in paragraph 6.1.2.1 and shown in Figures 6-12 and 6-18. It amounts to about 9 chips of MOS/SOS hardware. This could be reduced to four to five chips if a 150 pin pack is developed. An interface description of the control signals to the processor is given in paragraph 6.1.1. A similar description for the control signals to the I/O units is given in the following section describing the I/O units.

Both the semiconductor and magnetic memories have been described as capable of completing their read and write transmissions in one microsecond. This is the preferred method of operation since slower operation would not simplify the memory circuitry. As a result the memories will be able to be recycled in 1.5 μs

instead of the required 2 μs. This increased speed can clearly be used to decrease any queueing at the memories. However it could also be used by the processors on instructions that only require three bit times for execution. For example, if an instruction cycle does not require indexing with the T_n registers, the processor control unit can terminate, at the end of bit time 3, all instructions that do not have any functions besides indexing to perform in bit time 4. When the final control sequences for the instructions are laid out, this 1.5 μs memory cycle can be taken advantage of in a number of instructions. As a result the approximate instruction execution times given earlier in the instruction list will be decreased in some cases.

6.1.3 Input/Output Unit

6.1.3.1 Introduction

The input/output section of the multiprocessor is a hierarchical structure with the sensors at the bottom, the conditioners next, and the I/O units on top. This structure is shown in Figure 4-17 and was discussed in Section IV. The sensors are devices that carry out the actual monitoring and control tasks in the spacecraft. The conditioners are specialized to provide the proper control signals to the sensors attached to them. They receive commands, such as read and write, from the I/O units and then use these either to obtain data from the sensors or to send the sensors data or command sequences to be executed. The communication links between the sensors and conditioners and conditioners and I/O units are serial since the sensors pass relatively small blocks of words at low repetition rates. The I/O units themselves are not closely associated with individual sensors or devices. They simply receive calls for I/O actions from the processors (through the memories) and then send a read or write command (along with a data word if appropriate) and sensor name to the appropriate conditioner or to the bulk storage unit. (Communication to the bulk storage is over parallel lines since this unit will have high access rates within a block of storage.) The conditioners then control the sensor operations including sending data back to the I/O unit if necessary.

Many of the techniques that will be used in the Mars Lander Mission for handling guidance and control, status monitoring, and scientific data have been established; however, there will certainly be many new developments. As a result the sensors to be used in such a mission are presently not well defined, especially in the area of scientific experiments. This of course means that the conditioners also cannot be well defined since their primary task is to generate control sequences, carry out analog to digital conversion, etc. for the sensors. However, certain general properties of the programs necessary to operate upon and handle the data from the sensors can be defined. These properties, typical of a wide range of spacecraft programs, will be used to obtain a first approximation to the design of the I/O units.

There are three basic I/O program types: those associated with periodic sensors, the bulk storage unit, and request and background programs. The programs associated with the periodic sensors are characterized by relatively low periodicity rates, a maximum of about 20 repetitions per second, and short to medium sensor waits for single and multiple word data samples. Clearly there are some exceptions where the sensor waits may be hundreds of microseconds, but in these cases the processor should call the I/O data long before the data is needed. Typically, the periodic data will be called from the header of a program in order to waste the minimum amount of processor time waiting for data.

The programs using the bulk storage unit will generally initialize a transfer of a data block to or from the bulk storage unit and will then relinquish control of the processor. The access rate within a block of storage in the bulk storage unit may be as low as 10 μs or less; as a result, the I/O unit must be able to adequately interleave these transmissions with the lower rate I/O programs. The bulk storage unit will be used relatively infrequently for its main task of reloading the memories at phase changes or reconfigurations, but it must also act as a data buffer for TV pictures at Mars (approximately 30, 000 bits/second), for certain scientific experiments, and for telecommunications (approximately 20, 000 bits/second). For these latter tasks during certain phases of the mission, the bulk storage unit may be required to transmit blocks of data to the multiprocessor for computation every few hours; however,

since this is transmission from a buffer, it can simply be handled as a background program and therefore not interrupt the periodic programs. The third type of I/O program is that associated with request and background programs. These programs along with the Executive generate all requests for data from the bulk storage as discussed above. They also require serial sensor data. (This data may experience long sensor delays.) The background and request programs are scheduled as time is available and consequently are interruptible by the periodic programs.

6.1.3.2 I/O Unit Connections and Structure

The I/O unit is shown connected directly to the memory in Figure 4-17; however, consideration was also given to connecting the I/O unit to the processor instead of to the memory. The primary problem with the latter type of connection is that the I/O unit must preempt a processor for all its memory cycles. Note that this is the case in a single computer system, but in a multiprocessor the memory modules can service more memory cycles than the processors can provide. For the periodic type programs I/O-processor connections present no problems; for if the I/O unit was connected directly to the memories, it would most likely be using the same memory as the processor and would consequently steal a memory cycle from the processor anyway. Programs that transfer blocks of data to and from the bulk storage could be set up so that they do not use the same memory that the processor typically uses for program storage. In this case, with connections directly to all memories, the I/O unit would not have to preempt memory cycles from the processors and as a result would take good advantage of the system resources (i.e., the extra memory module). However, if the I/O unit was connected to the processors, it would have to preempt processor memory cycles even though the processor and I/O unit were using different memory modules. There are other less important reasons for using I/O-memory connections, but the above discussion should be sufficient to demonstrate that having the I/O units connected to the memory provides the most flexible multiprocessor system.

Figure 6-19 shows the registers and main connections in the I/O unit. This unit is designed so that transfers from both the bulk storage and the sensors can be conveniently interleaved. To enable this, there is a set of registers to handle bulk storage requests (BR, PMAC, Pc-Ch, PWC) and a set to handle sensor requests (ASR, SMAC, Sc-Ch, C-D, SWC). The basic function of each of these registers will be outlined below.

6.1.3.2.1 Registers

MR - Memory register: The memory register receives data and instructions from the memories and sends data to the memories. The data going to the memories can be from either the sensors (ASR) or from the bulk storage (BR).

ASR - Assembly shift register: This register receives serial data from the conditioners and sends it to the MR for transmission to the memory. It is also used to send serial data and control words to the conditioners.

SMAC - Serial memory address counter: This counter is set up by the I/O control word from the memory. It holds the memory location for reading or writing serial I/O data. This value is decremented by the control circuitry after each serial word is transmitted either to the memory or to the conditioners.

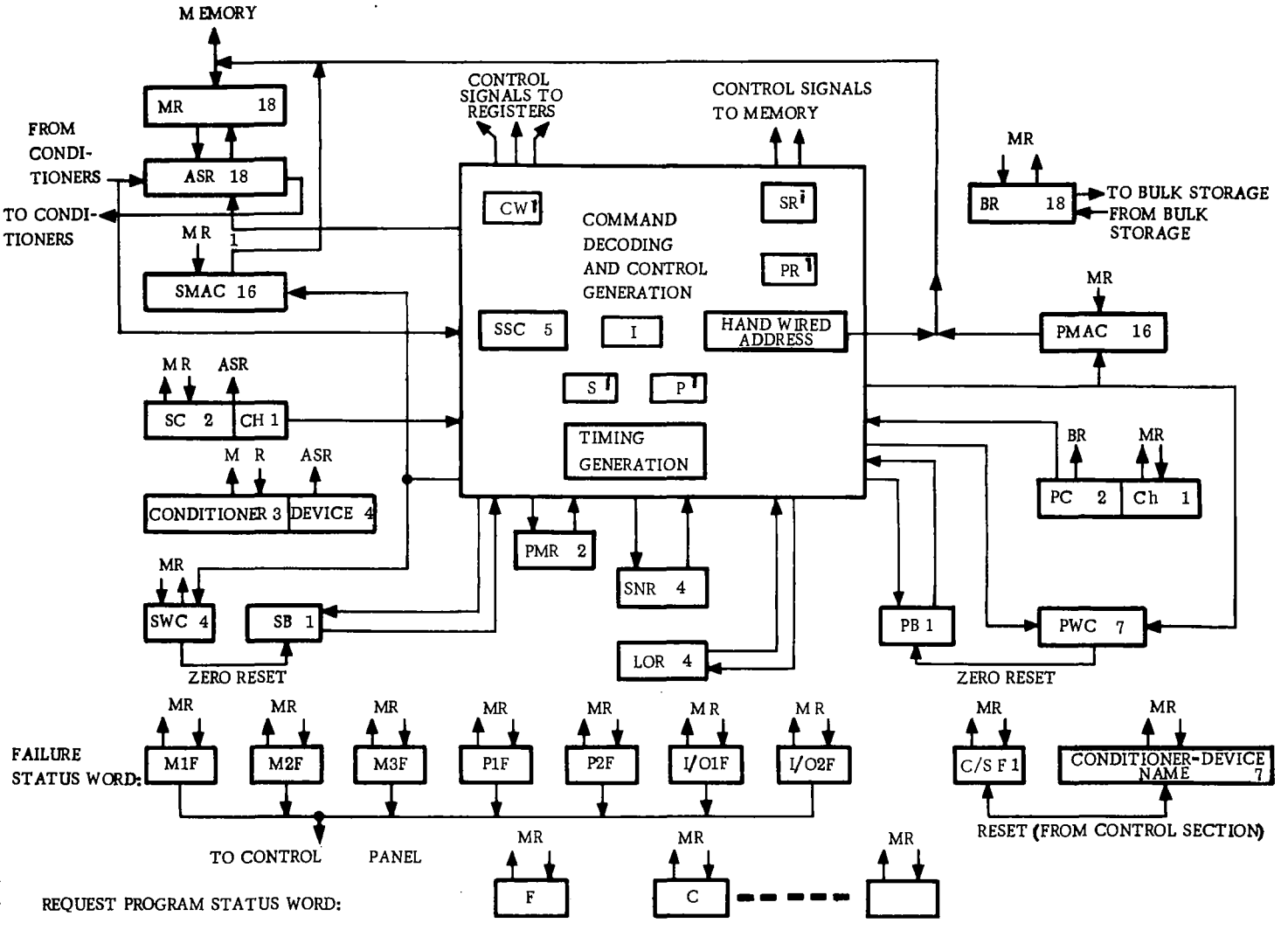


Figure 6-19. Input/Output Module

SC-Ch - Serial command and chain bit: The command and the function of the chain bit will be discussed later in this section. The SC bits are held for controlling memory cycles (read or write) and for sending commands to the conditioners. The Ch bit influences the generation of control signals in the I/O unit.

C-D - Conditioner and device registers: This register holds the name of the conditioner and sensor participating in a serial I/O operation. The register is loaded from the MR. It sends the device name to the specified conditioner as part of the conditioner command word.

SWC - Serial word count register: This register specifies the number of serial I/O words to be transferred. It is loaded from the MR and is decremented along with the SMAC by the control circuitry after each serial word is transmitted. The serial operation is terminated when this register is equal to zero.

SB - Serial busy flip flop: This flip flop is used to notify the memories on request that the I/O unit is busy executing a serial I/O program. It is reset by the SWC register when this register is decremented to zero.

SNR - Scanner register: This register is used to sequentially grant the memories access to the I/O unit. It sequences through the request lines and halts as soon as a line is found up. The scanner will sequence through the request lines as long as either SB or PB are zero, or the lockout register is not set.

LOR - Lockout register: The register is set by a control line from each of the four (three for this mission) memories. A memory will set the lockout register at the start of a critical computation phase or at the start of a periodic program execution. For a memory module to set the lockout, it must set the lockout flip flops associated with the other three memories. This will enable only this memory to use the locked I/O unit for serial or parallel I/O transmission. The setting of the lockout register also causes the I/O unit to be interrupted and store its status in the interrupting memory.

BR - Buffer register: This register receives parallel data from the bulk storage and sends it to the MR for transmission to memory. It is also used to send parallel data and control words to the bulk storage unit.

PMAC - Parallel memory address counter: This counter, set up by the I/O control word, holds the memory location for reading or writing parallel I/O data. This value is decremented by the control circuitry after each parallel word is transmitted.

PC-Ch - Parallel command and chain bit: These bits function just as the SC-Ch bits only for parallel transfers.

PWC - Parallel word count register: This register functions just like the SWC except for parallel I/O operations.

PB - Parallel busy flip flop: This flip flop has the same function as SB except for parallel operations.

There are also a number of control and status flip flops shown in Figure 6-15 that will be explained later in this section.

6.1.3.3 Timing

The timing in the I/O unit is fairly simple and can be carried out by a few one shot multivibrators. These devices must time the memory interface request signal (500 ns), the shifting of the ASR (500 ns or 1 μs), and the acceptance interval for the memory request (14 μs as given in section 6.1.1). Some additional timing may be necessary for timing the requests to the conditioners, but this cannot be specified at this time. The remainder of the actions carried out by the I/O unit are asynchronously timed (the conclusion of one event starts another). If in the 1975 time frame a small simple hardware clock is developed, it could be substituted for the multivibrators. However, the multivibrators will probably prove to provide the least complex timing for the I/O unit.

6.1.3.4 Memory Interface

The lines on the I/O memory interface are given below. It should be noted that except for a few additions and deletions, this is the same as the processor-memory interface.

Component I/O

Interface Memory

Output (to memory)

request	One separate line to each memory. It is used to request memory cycles.
address/data	18-bit two-way bus common to all memory modules. It sends addresses and data to the memory and receives data and control words from the memory.
read/write	Bit 18 of an address on the address/ data bus. This line is used to notify the memory of a read or write request.
lockout	One separate line to each memory. It notifies a memory that it is locked out of this I/O unit.
serial busy	One common line to all memories. This line notifies all the memories that this I/O unit is carrying out a serial I/O program.
parallel busy	One common line to all memories. This line notifies all the memories that this I/O unit is carrying out a parallel I/O program. It should be noted that both a serial and parallel busy are required since it is possible for one to be busy and the other not busy or free to service a request.
accept	One separate line to each memory. This line notifies the memory that its request for I/O access is granted.

Input

busy	One separate line to each I/O unit from each memory. It notifies an I/O unit of acceptance of a request.
request	One separate line to each I/O unit from each memory. It is used to request the I/O unit to receive a control word from the memory.
lockout set	One separate line to each I/O unit from each memory. This line requests the I/O unit to lockout all other memories.
lockout	One separate line to each I/O from each memory. This line notifies an I/O unit if it is locked out of any memories.
address/control	Same as the address/data bus given in output.

The timing and functions of the above lines that are associated with an I/O unit requesting a memory cycle are the same as for a processor requesting a memory cycle. The description is given below. The operation of the I/O unit control section to provide parallel or serial addresses, etc., is described later in this section.

1. The I/O unit sends a request to a memory ("0" to "1" transition occurs on the request line). At the same time it places the memory address and read/write request on the address/data lines.

2. After the memory scanner chooses the requesting I/O unit for a memory cycle, the memory picks up the address and read/write information from the bus. It then sends the I/O unit a not busy signal ("0" to "1" transition occurs on the busy line). This signal is also used to start the 500 ns I/O request one shot. The memory uses the next 500 ns to address the specified memory position and to load its data register if a read is required.

3. For a write operation the I/O unit uses the "0" to "1" transition of the busy signal to load its MR and memory bus with the data word for memory. After 500 ns the I/O unit request signal will turn off. The "1" to "0" transition of this signal causes the requested memory to read the contents of the I/O memory bus into its data register.

4. For a read operation the I/O unit MR is loaded within 1 μs of the acceptance of its request. The load is accomplished by the memory placing the data on its bus to the requesting I/O unit and then turning off its busy line. The I/O MR is loaded from the bus by the "1" to "0" transition on the busy line.

The "lock out set" line causes the I/O unit to be interrupted on its "0" to "1" transition. The lockout set notifies an I/O unit that it is about to become involved in a periodic I/O program execution. The "lockout" line from a memory, on the other hand, notifies an I/O unit that it is locked out of this memory module. The programs will schedule I/O so that an I/O unit will never be sending data to a memory module from which it can be locked out during periodic program executions. This means that I/O unit one, for example, will use memory modules one and three to store I/O variables

but not memory module two since this latter memory will lock out all I/O units except I/O unit two during periodic computations. This programing restriction is necessary so that an I/O program will not be interrupted due to its memory not being available. This situation would require that the serial or parallel I/O program involved store its status in its primary memory; however, the program that initialized this I/O program would not be able to be notified conveniently that its I/O has been interrupted. As a result there is not a convenient method of reinitializing the I/O program when the locking memory is again free. An additional problem is that this type of an interrupt would require nesting since more than one memory could lock out the same I/O unit before the original I/O program is brought back into execution. Future investigation of the above problem should be carried out in order to try to relieve the programing restriction.

In order to explain the rest of the memory interface lines, a discussion of the CIO (call I/O) instruction is necessary. The CIO instruction is used by the processor to initiate I/O operations by sending two control words to the I/O unit from the memory. The I/O control words will be described shortly, but in any case two words are necessary to initiate an I/O operation. The processor sends the memory a request for a memory cycle and also a signal on a separate line (the "I/O" line) to the memory. (This was explained in section 6.1.1 under "Memory Interface.") The memory then responds to the processor in the normal manner except for two changes. The processor is granted two memory cycles in a row (the scanner is inhibited while the "I/O" line is up); and the data is sent to an I/O unit instead of back to the requesting processor. In order to determine which I/O unit to send the data to, the memory looks at the first two bits of the first control word and interprets these as an I/O unit name. The third bit of the control word is also checked to see if a serial or parallel I/O operation is being requested. After the above is accomplished the memory will send a request to the proper I/O unit if all of the following three conditions do not exist:

1. The memory is locked out of the I/O unit. (The lockout line from the I/O would be one in this case.)

2. The memory has a request for a parallel I/O operation and the parallel busy line is up from the specified I/O unit.

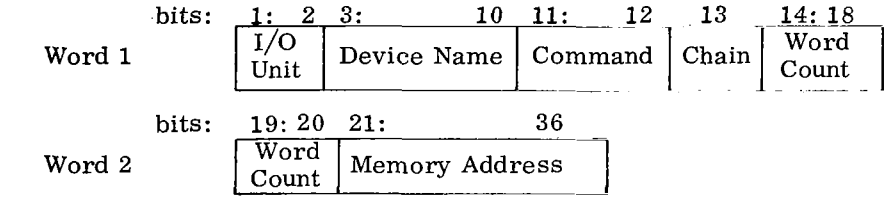
3. The memory has a request for a serial I/O operation and the serial busy line is up from the specified I/O unit.

If any of the above three conditions exists, the memory sends the I/O BL (I/O busy or locked out) signal back to the processor. This terminates the CIO instruction and both the memory and processor are freed. If none of the conditions exists, the memory's request to the I/O unit will be granted within 2 μs. The "0" to "1" transition of the accept signal from the I/O unit will load the memory bus and the I/O MR with the information from the memory's data register. The accept signal will then remain up for the next memory request. In order to generate the next request the processor increments the address of the previous I/O control word by one and sends this new address to the memory. The memory loads the second control word into its data register and sends a request to the I/O unit with its accept line up. The I/O unit will then turn off its accept signal. The "1" to "0" transition of this signal will again load the memory bus and the I/O memory register with the information from the memory's data register. It should also be noted that between the receipt of the first and second control words the I/O unit must distribute the contents of the first control word to the proper I/O registers. The execution of the complete CIO instruction (including processor instruction access) takes six microseconds.

6.1.3.5 I/O Control Word Format

The previous subsection presented a discussion of the CIO instruction. The control words called by this instruction will now be presented and discussed.

The I/O control word format is shown below:



Bits

1:2 I/O Unit: These bits are decoded by the memory during a CIO instruction in order to determine the I/O unit to receive the control words.

3:10 Device Name: If bit three is 1, the device is the bulk storage. Bits 4 to 10 will then be loaded into the BR to be sent to the bulk storage as the upper seven address bits for this unit. If bit three is 0, the device is one of the sensors connected to the conditioners. Bits 4 to 6 will then represent the conditioner and bits 7 to 10 will represent a device (sensor) on the chosen conditioner. In the latter case, bits 4 to 10 will be immediately loaded into the C-D register to prepare for receipt of the next control word. The device name organization mentioned above provides for a total of 128 devices. This should be sufficient since many of the sensors will send more than one word on request. Actually only 126 device names will be usable since one device name will be used to call a request program status word, and a second will be used to call a failure status word. These status words will be discussed later in this section. The correct trade-off between bits for the conditioner name and bits for the sensor name cannot be made at this time, but that shown will probably be close to correct.

11:12 Command: Bit twelve is used to denote read (I/O unit reads from the memory) or write (I/O unit writes into the memory). Bit eleven is used along with the chain bit (bit 13) to specify immediate reading or discrete reading or writing. Immediate reading means that the second control word from memory actually contains a data word rather than a memory address. Discrete words are sent to conditioners in order to provide special control sequences or to provide control signals such as "turn off" to devices. Discrete words are also sent from the conditioners to the memory in order to provide the computation system with status information on the sensors and conditioners. A listing of the above instructions is given below:

B_{11}, B_{12} = 00 - Write: A word or words are obtained from the specified device and written into the memory location given by the SMAC or PMAC.

B_{11}, B_{12} = 01 - Read: A word or words are read from the memory location given by SMAC or PMAC and sent to the specified device.

B_{11}, B_{12}, B_{13} = 100 - Not used.

B_{11}, B_{12}, B_{13} = 110 - Read immediate: The second memory control word is used directly as a single word of data for the specified device. This command will have no real use with the bulk storage unit, but it should be useful for a number of serial devices.

B_{11}, B_{12}, B_{13} = 101 - Discrete write: This command is used to pick up a discrete word from the specified conditioner and device or from bulk storage. The exact contents of the discrete words cannot be specified until the devices themselves have been specified. The chain bit is used for command extension in this case because no more than one discrete word will typically be picked up at a time. In the cases where it would be desirable to chain an additional CIO must be executed.

B_{11}, B_{12}, B_{13} = 111 - Discrete read: This command is used to pick up the second control word as a discrete word and send it directly to the specified device. The chain bit comments given above apply here.

13 Chain: The commands discussed above use this bit for command extension when bit eleven is 1. When bit eleven is 0, this bit is used for command chaining. In this case the I/O command is executed on the specified number of words (see word count below); and then instead of terminating this I/O program when the word count register (SWC or PWC) goes to zero, the word count register is incremented twice and a request is sent to the memory. Two control words are picked up from the two memory locations following the previous I/O program data area. (These words were set up by the processor prior to its execution of the initial CIO instruction.) These control words are then used to continue execution of an I/O program with serial or parallel devices. If serial (parallel) devices were used in the first I/O program, they must also be used in the chained program; otherwise a new control word could be brought in for a busy parallel (serial) device. This chain can be continued as long as desired. The chaining of I/O programs may prove to be particularly useful for transferring large blocks of words between memory and the bulk storage unit. Note that using the chaining feature saves the trouble of requiring the processor to monitor the transfer of data and to execute a number of CIO instructions.

14:20 Word Count: These bits are loaded into the SWC or PWC registers. These registers are then used to count the number of words transmitted in the I/O program and to terminate the program when the count goes to zero. The SWC register only uses bits 17 to 20 since a single serial device will probably never transfer more than sixteen words; however, if the situation arises, the SWC register could easily be increased. The PWC register, on the other hand, is seven bits since the bulk storage unit may want to transfer as many as 128 words quite often. In fact the word count section of the control words could be increased to allow for more words to be transferred from the bulk storage with one set of control words; however, all the other bits in the control words have been put to good use. If in the future less than 64 serial

devices are needed on a single I/O unit another bit could be made available for the word count. However, this problem is actually alleviated by the use of the chain bit. This bit makes transfers of blocks of words longer than 128 words relatively simple. In fact one of the main reasons for adding the chain bit to the control word instead of making the word count eight bits was to enable long word block transfers without the need for processor intervention.

21:36 Memory Address: These bits provide the initial memory address for reading or writing. They are loaded into either the SWC or PWC.

A third I/O control word will be required for I/O programs working with the bulk storage unit. This word will provide eighteen more address bits for the bulk storage location. This word will be picked up by the I/O unit from the first memory location given in the PMAC. A total of twenty-five address bits are then available for bulk storage addressing. (Seven bits were obtained from the device name locations in control word one.) If more bits are necessary for addressing this unit, a fourth control word would have to be obtained. A twenty-five-bit address would be sufficient for a bulk storage of approximately 6×10^8 bits stored as 18-bit words. A number of 10^8 bit bulk storage units are discussed in Appendix II.

6.1.3.6 I/O Device Interface and Word Format

The interfaces to the devices and the control words for the devices cannot be explicitly specified until the devices themselves have been designed. As a result this section will only give a discussion of some of the lines and bits that may be necessary on the interfaces and in the control words.

Since the access of I/O variables is program controlled, the I/O unit should send requests to the devices. The requests will cause the conditioner to immediately begin receiving information over a two-way serial line. There should be no need for the I/O unit to check and see if the device is busy since it would probably be a programming error to use a device twice in succession without allowing it sufficient time to complete its first operation. As a result, if an I/O program is interrupted while a device is busy, the present operation should be halted immediately and the present data or control word will have to be obtained again when the interrupted program is restarted. This means that the devices should receive a halt line from the I/O units. In order to enable loading of I/O variables from the ASR and BR into the devices, each I/O unit could send the bulk storage and each conditioner separate "load" lines. A "0" to "1" transition on one of these lines would then cause the appropriate device, bulk storage, or conditioner, to receive a word or a bit. Similar lines could be run from each conditioner and from the bulk storage to the I/O unit. These lines would enable loading of data from the devices into the I/O unit.

In addition to the above lines, each conditioner and device may have a number of separate lines to the I/O unit for failure notification. These lines would set the C/S flip-flop and conditioner device register. The existence of these lines depends on whether or not the conditioners and sensors have some self-checking hardware. If they are not able to check themselves, they will be checked under program control. In this latter case there would be no need for failure notification lines.

At the present time it appears that the best way for the computation system to handle certain devices and functions is to have them set a request flip flop in the I/O unit when they require servicing. (This of course requires lines from the devices to

each I/O unit.) These flip flops would be periodically monitored by an executive program in order to see if servicing was required. A good example of such a device may be an astronaut input/output console. A request flip-flop, C, is shown in Figure 6-19 for this device. This console will probably be included to enable the astronauts to do some programing and to request a variety of outputs from the computation system. Another example is the use of a failure flip-flop as shown in Figure 6-19. This flip-flop would be set any time one of the failure status word flip flops is set. After noting that the F flip flop was one the executive would reset the flip flop and then read the failure status word in order to determine the proper corrective action.

An example of some of the contents of a control word for a conditioner is given below:

bits:

0	1: 4	5	6
Control/ Data	Device Name	Read/ Write	Discrete

Bits

0 Control/Data: This specifies that the following word is a control word or a data word.

1:4 Device Name: This gives the device name on the conditioner that is receiving the control word.

5 Read/Write: This requests a read (from memory) or write (to memory) operation from the specified device.

6 Discrete: This notifies the conditioner that the read/write command is for a discrete word.

The control word for the bulk storage is also not specified at this time, but the following example is probably accurate:

bits:

Word 1

1	2	3: 9	10: 16
Read/ Write	Discrete	Word Count	Bulk Address

bits:

Word 2

19: 36
Bulk Address

Bits

1 Read/Write: This requests a read (from main memory) or write (to memory) operation.

2 Discrete: This tells if the next control word is a discrete for a read. For a write operation with the discrete bit on, control word two is ignored and the bulk memory sends a status word back to the I/O unit.

3:9 Word Count: This is the word count from the I/O unit's PWC register.

10:16, 19:36 Bulk Address: This is the address of the first bulk word to be transferred or loaded.

Note that bits 17 and 18 of control word one are presently unused. It should also be noted that the bulk storage will be connected to more than one I/O device. As a result, whenever it is busy the I/O unit using it will send all other I/O units a busy signal that will be used to set their PB flip flops. This will inhibit two I/O units from accessing the bulk storage at the same time.

6.1.3.7 Functional Description

A functional description of much of the I/O unit has been given throughout this section; however, additional clarification in certain areas is needed. In particular, a discussion of the control flip-flop sequencing of operations, and the handling of interrupts will be given in this section. In order to best understand this subsection Figure 6-19 should be referenced.

As mentioned earlier, a CIO instruction initializes the I/O unit for a parallel or serial I/O program. The request for access to the I/O unit is honored by the scanner (SNR) which is then locked on the memory for two requests (first and second control words). After the first control word is transferred to the MR, portions of the word are loaded into the appropriate serial or parallel registers and the SB and S (serial) flip-flops or the PB and P (parallel) flip flops are set. The S or P flip-flops are used to tell the control section of the I/O unit which device type receives the next memory word. After the second control word is received and transferred by the control circuitry to the appropriate registers, the I/O unit is no longer under control of the memory. The I/O control section next loads the control word for the device into the ASR or BR and then transmits this word to the device. For a read operation, a memory cycle is requested by setting the SR or PR at the same time the above operation is taking place. (The sequencing of operations with the memory has been discussed earlier.) The memory cycle, when granted, reads a data word from the location specified by the SMAC or PMAC. The memory address is placed directly on the memory bus as shown in Figure 6-19. The data word is loaded into first the MR and then into either the ASR or BR. At this point the S or P flip-flop is turned off and the data word must be sent to the appropriate device. The timing for the transmission of the serial word is generated from the control section timing hardware. The number of bits shifted is counted by the serial shift counter (SSC). After the data word has been transmitted to the device the SMAC or PMAC and SWC or PWC are decremented. If the word counters have not reached zero, the above operation repeats itself, starting with the setting of the SR or PR flip flop in order to send a request to the memory. (There is of course no need to send another control word.) If the counters have reached zero the I/O program is terminated and the appropriate flip flop, PB or SB, is set to zero.

The operation of the I/O unit for a write operation is very similar to that for a read. However, after the control word has been sent to the device the S or P flip flop is turned off and a memory cycle is not requested. First the specified device obtains a data word and sends this back to the I/O unit. The data word is loaded by a "0" to "1" pulse on the "load" line from the devices to the I/O unit. For transmission over the serial line, the I/O unit control section counts the load pulses in the SSC. When the count reaches 18, the word transmission is complete and a memory cycle is requested by setting SR or PR. When the I/O unit obtains a memory cycle the address from the PMAC or SMAC is placed on the memory bus and at the same time the data word to be sent to the memory is transferred from the ASR or BR to the MR. While the data word is being transmitted to the memory, the SMAC or PMAC and SWC or PWC are decremented. The write operation is then terminated or repeated, just like the read operation, depending on whether the counters have reached zero or not.

If the chain bit is set to one by the initial control word, the termination procedure given above for read and write operations is altered. When the word counter reaches zero, the set chain bit flip flop causes the control word flip flop (CW) to be set to one (see Figure 6-19), the word counter (parallel or serial) to be incremented twice, the SR or PR flip flop to be set, and also inhibits the resetting of SB or PB. The next two memory cycles will use the SMAC or PMAC for a memory address, and the words received will be treated as control words. After receipt of the first control word the SWC and SMAC or PWC and PMAC are decremented. After receipt of the second control word, the registers are again decremented. The word counter will then generate a zero signal that will cause the CW flip-flop to be reset. The SB or PB flip-flops will remain set and the I/O unit then proceeds in the same manner as if a CIO instruction has just been carried out.

One additional point should be mentioned about the operation of the control flip-flops SR, PR, S, and P. During the simultaneous operation of a parallel and a serial I/O program by a single I/O unit, one program may request a memory cycle while the other program is in the midst of a memory cycle. For example, SR could be one with a serial device waiting for a memory cycle. If the bulk storage must now also be supplied a memory cycle, the P flip flop will be set, but PR will not be set. As soon as SR returns to zero after its memory cycle, P will go to zero and PR will be set. As a result, no data is lost, and the granting of memory cycles will be sequenced.

One situation has been presented earlier that causes the I/O unit to be interrupted. This occurs when the lockout set line from a memory to an I/O unit goes to one. This notifies an I/O unit that it is about to become involved in a periodic program execution. The I/O unit is interrupted immediately unless a memory cycle is in progress, in which case the cycle is completed first. The interrupt is then carried out in the following fashion. The lockout register is set so that only the interrupting memory will have access to the I/O unit. A serial status word must now be stored if the SB flip flop is one. If this flip flop is zero, the I/O unit is ready to receive a memory request since no serial I/O program is in progress. The serial status word is stored by setting the interrupt flip-flop (I) and thus causing the I/O control circuitry to carry out the following actions:

1. A memory cycle is requested and the primary memory register (PMR) and the hard wired address are used for the memory address. The PMR is used for the two most significant address bits. (The PMR can be set by the executive with the CIO instruction and the device name bits referring to a request status word.)

2. The SMAC is then transferred to bit positions 3 to 18 of the first interrupt memory location and the last two bits of the SWC register are transferred to bits 1 and 2 of the same word. The PMR and hardwired address are next transferred to the SMAC.

3. The second I/O status word as shown below is transferred to the MR and the SMAC is decremented.

bits: 1: 3	4: 10	11: 12	13	14: 16	17: 18
(blank)	Cond-Device	SC	Ch	(blank)	bits 1 and 2 of SWC

4. Another memory cycle is requested and the SMAC is used for the memory address. This is just the hardwired address minus one.

5. This memory cycle loads the status word from the MR into the second interrupt memory location. The I and SB flip flops are reset at the same time.

The I/O unit is now ready to carry out periodic I/O programs.

Note that a parallel status word was not saved since the periodic programs do not use the bulk storage unit. When the periodic programs are completed, the executive must restore the interrupted I/O program by simply giving a CIO instruction with the stored status words as the control words. (These status words were purposefully stored with the registers in the same positions as in the original control words.) Before giving the CIO, the I/O unit name must be entered in status word one, bits 1 and 2.

The failure status word in Figure 6-19 is set by a line to each I/O unit from each module in the system as mentioned earlier. These flip-flops also drive lights on the astronaut's control panel. The failure status word is read by the executive program whenever the request program status word "F" flip-flop is found up. The executive program will then compare the new failure status word with the last value of this status word stored in the memory in order to determine what failure has occurred. The appropriate reconfiguration and software checking actions will then be taken.

The I/O unit as shown in Figure 6-19 contains 150 flip-flops. An approximation for the gates and drivers in the system would give a rough total (including flip-flops) FET or device count of 5,000. This should easily be implemented on a single 150 mil square chip in the 1975 time frame. (This assumes the 5,500 FET's per 150 mils square presented earlier for the processor.)

6.2 FAILURE AND ERROR DETECTION AND CONTROL

A preliminary treatment of failure and error detection and control was given in Section IV for the multiprocessor organization. The coverage in that section of this topic was primarily based on software methods of failure and error detection. Since the multiprocessor was selected for further investigation, hardware methods of failure and error detection were also investigated. This section will therefore cover both hardware and software approaches to failure and error detection. The appropriate use of the two methods or the 'mix' of the two of them to achieve failure and error detection cannot be specified at this time. One of the most important parameters that will influence this mix is the probability of failures or errors being intermittent or transient. If the probability of intermittents is negligible, then software methods may suffice with very little if any hardware methods added and vice-versa. It should also be mentioned here that the maximum time to detect a failure or error is a very important parameter when determining the proper mix. This time was defined as 5 seconds for the application and this is relatively long enough so as not to penalize software methods heavily (the smaller this time, the higher the percentage of time devoted to software self test). The two approaches are presented here and further study is necessary to determine the exact mix of the two that should be employed. A general treatment of the topic of failure and error detection and control is given in Appendix 3; preliminary thoughts and various approaches to this topic are given in that section.

6.2.1 Software Methods

Software self tests are of two general types, problem oriented and machine oriented. Both programs, if properly designed, that is, knowledge of the hardware failure modes used to determine checking values, will be equally complete. Problem oriented programs utilize the operational program by either testing the normal output for reasonableness or running a set of pre-chosen constants. Machine oriented self tests are designed such that the test problem is based on the hardware characteristics independent of the particular sequence they are exercised in the operational mode. Table 6-1 contains the dominant advantages and disadvantages of each approach. Both processes have the common disadvantage that the percentage of lost computation time is directly proportional to the required speed of error detection time and the error reporting is most often the absence of a go signal rather than a positive signal output. They have the common advantage of flexibility as compared to hardware approaches.

Table 6-1. Software Test Characteristics

Problem Oriented	Advantages 1. Minimum extra storage requirements. 2. Errors affecting only that particular program being executed are detected. Useful when computer is performing a very limited set of functions. 3. Running time is short when the operational program has a high cycle rate.

Table 6-1. (Cont)

	Disadvantages 1. Changes each time operational program changes. Added analysis and recertification of completeness required. 2. Special safeguards must be implemented to inhibit outputs when test problem is being executed. In general, this lengthens operational program. 3. Different error response for each computer in the system.
Machine Oriented	Advantages 1. Same program used for all processors. 2. Program independent of problem changes. 3. Added property of distinguishing between operational program mistakes and computer failures. 4. Is generally constructed for in-house use and is available with small modifications for operational use.
	Disadvantages 1. Requires additional storage capacity. 2. Execution time longer when operational program has a high cycle rate.

Following is a listing of machine oriented software tests. These programs can be made complete enough to provide a probability of detecting failures or errors very high, approaching 100%, given that these are solid failures or errors. These tests may or may not detect intermittents or transients, of course as the tests are run at a higher rate and take proportionately larger amounts of computation time away from the operational program, more intermittents or transients will be detected.

6.2.1.1 Memory Check Sum

The memory check sum routine simply adds the contents of fixed storage locations (instructions and constants) without regard to overflow and compares the result with the prestored correct response. The function of the test is to check for potential malfunctions in the computer memory and processor.

The check sum routine could be written to add all of fixed storage at one time. This method was not chosen because of programming inefficiencies which would result from having to keep track of which blocks in memory contain fixed information and which contain variable information. Instead a check sum routine would be built into

each major programming segment and would be performed at the outset of the segment. Parameters such as the starting address, number of locations to be added, and expected check sum response are included as part of the program segment package. Initialization, execution of the check sum, and checking of the response would be handled by a utility routine. With indexing and the appropriate index test, decrement, and transfer instruction the check sum execution can be handled by a two instruction loop.

6.2.1.2 Arithmetic Section Functional Test

This test checks the performance of the arithmetic section logic circuits of the processor. No special test instructions are envisioned, therefore, no additional hardware would be designed into the system to perform this test. Patterns for exhaustively testing the arithmetic logic are prestored in memory and under program control act as stimuli to the logic. The responses of the logic are compared with prestored correct responses to determine the status.

Based on previous experience in writing this type of test, it is estimated that for this application the test would require 425 instructions and 75 constants and temporary storage locations. For a 4 μsec add time the test would run for about 2 msec. The degree of completeness, or the ability of this test to detect arithmetic section errors is expected to be high, (about 99 percent). Of course, proving this would require a thorough analysis which involves determining likely component failure modes and the ability of the test to detect the effects produced by the component failure modes. Such an effort would be in order if further design were performed on the multiprocessor organization.

The test is performed at a periodic rate. Its frequency would be adjusted to insure that the worst case reconfiguration time of 5 seconds during critical phases would be met.

6.2.1.3 Program Control Test

This test checks the ability of the computer to execute instructions in a legitimate operational sequence. Computer malfunctions which produce affects that are described by saying the computer is hung-up within an instruction, within a loop of random size, or wandering aimlessly through instruction sequences, would be detected. Malfunctions producing such effects can originate in the control logic of the processor, the memory, the clocking system, or the power supply.

Efficient implementation of this test requires insertion of built-in test equipment (BITE) to mechanize a timing device. As an example, a digital timer would operate as follows: Under program control, a periodic square wave is set up and acts as input to the timer which consists of counters and associated logic. Tolerances are set on the duration of the "high" and "low" portions of each cycle of the square wave and on the period. The inability of the computer to provide this prescribed square wave, which would occur in the presence of a control error, would be detected by wired-in logic associated with the counter and result in the setting of an error flip flop indicating a computer failure. The period of the square wave and the associated tolerances would be determined to satisfy the worst case reconfiguration time requirement of 5 seconds.

From the programming point of view, periodically, an instruction has to be executed to effect the high portion of the wave, and a prescribed time later another instruction is executed to effect the low portion.

In the multiprocessor configuration the requirement has been established to isolate errors to the processor or memory. The above tests provide this capability only to a limited extent. For example, a processor arithmetic error can be isolated to the processor by executing the arithmetic functional test twice, one from each of two memories. Normally the test would be executed the second time only upon failure of the first test. Similarly, a memory failure is isolable by a check sum where the memory is an operand source, not an instruction source, for two processors. Where a memory is an instruction source at the time of its failure the program control test will detect the error, as it will if the processor contains a control error.

The approach chosen to isolate errors between memories and processors (and between processors and I/O units too) generally takes advantage of the fact that isolation need not be instantaneous and that the space crew is available to perform procedures as required for isolation subsequent to error detection. The penalty of this approach is that more equipment than otherwise necessary may be placed in a "down" condition at the time an error is detected and, of course, also that more crew participation is required. However, an analysis of the mission success and availability requirements shows that those requirements can be adequately met with this approach.

6.2.1.4 Input Signal Tests

Tests performed on input signals can detect failures due to errors in sensors, in data transmission, in input signal conditioning circuitry, or in transferring the signal through the input section of the computer to either the processor section or the memory. Where tests are performed during normal operation of the system (on-line) the stimuli are not "canned" as they are in the case of arithmetic section tests since the sensors are not interrupted to provide prescribed input signals. In place of prescribed sensor values for testing purposes, the validity of these signals can be tested within the arithmetic section of the processor by a combination of the following techniques: reasonableness tests, dual redundant inputs, and BITE. Reasonableness tests use criteria such as the expected range and/or rate of the input parameter for error detection. Redundant inputs allow the disagreement between the inputs to provide error detection. BITE in the form of input conditioner built-in stimuli under program control provides a backup to reasonableness tests and redundancy both for enhancing the error detection capability and for error isolation. The redundancy technique is the least desirable due to reliability and power considerations and would be used selectively, only if a study of the proposed reasonableness tests, BITE, and the criticality of the input signal indicate it is necessary.

Given that errors will be detected by the above mentioned techniques, the isolation problem is to determine if the input device, I/O conditioner, or computer is the error source. It is assumed that the input device cannot monitor its own status completely and will require computer participation for its status determination. It is further assumed that if digital transmission errors represent a significant problem, it would be handled by simple parity checking. A description of the detection and isolation process follows.

Included in the program segment requesting an input is the test required to verify it. If the input is acceptable normal operation continues. If the input is found to be in error, the error status is recorded in an assigned bit position of a status word in memory. (Assume one status word is reserved for each I/O conditioner thereby allowing reference in this description to "I/O conditioner status words"). Normal operation continues, even in this error case, except that the previous value of the input is used in the computations in place of the present value. At a prescribed point in the program, the executive looks at the contents of the I/O conditioner words. If this is the first input cycle in which an error has been detected, the executive permits performance of at least one more input cycle. Note that the number of input cycles resulting in error reports should be greater than one since there is little likelihood that an error will occur at the start of a cycle. But, once having occurred, if it is a solid failure, it will be present throughout all subsequent input cycles and its effect will be truly represented by the I/O conditioner status words.

Next, consider the manner in which the I/O conditioner status words can be used to isolate the failure once the failure history is complete. Basically the process is closely coupled to the function of the failed circuitry. If the failure occurs in circuitry peculiar to a particular input, only that input signal will be affected and only one input will be flagged in one of the I/O conditioner status words. Such errors are either in the sensor, the transmission path between sensor and conditioner, or in the conditioner prior to the point where inputs are multiplexed. If failures are indicated in more than one input signal, the failed point must be in time-shared circuitry. This could be in the conditioner between the point where inputs are multiplexed and its output to the computer, the transmission path to the computer, or in the computer input circuitry. (An additional source could be a gross sensor error where the sensor provides more than one input signal and all have been affected. Such specific cases can be checked for by the executive program if the sensor cannot be depended upon to provide such information.) In the multiprocessor more than one conditioner is tied to the computer input unit, therefore, errors in the computer's input circuitry will affect most input signals.

Thus, it can be seen that the number of input signals affected and their relation to one another can provide a certain degree of isolation of the error. The degree of unambiguous isolation is related to the failure rates of the components within the isolable boxes that can be associated with each effect. If all inputs were bad, one would suspect the computer input unit; if the bad inputs were associated with one conditioner, one would suspect the conditioner first even though there is circuitry within the computer input unit associated only with that one conditioner, etc.

From the programming point of view, each input has associated with it certain parameters and tests employing those parameters. Tests on operational inputs are performed at the rate the operational program requires the inputs. Test on non-operational inputs such as those supplied by BITE test signals are performed at a periodic rate. Detection of failures result in status notification by means of I/O conditioner status words in memory. The executive program interrogates these status words each cycle. A full cycle fault isolation routine is entered after the true failure history has been recorded in the status words. Isolation to a sensor, an I/O conditioner, or the computer input unit is achieved.

6.2.1.5 Output Signal Tests

In order to automatically detect errors in output signals, the loop on these signals must be closed. For this reason, all conditioner outputs are fed back to conditioner inputs and thereby made available for checking within the arithmetic section of the processor. As opposed to input signal varification by means of reasonableness tests, output signals are known at the time they are commanded. Therefore reasonableness tests are not required. All comparisons can be done digitally. Thus, for example, the output voltage derived from a digital output word can be brought back into the conditioner, converted A to D, and the resulting digital input value compared with the original digital output value.

The programming requirements for output signals are similar to those for inputs. Associated with each output signal is a test which involves executing an input command for the I/O conditioner input channel reserved for the feedback of the output, and a comparison of input and output digital values. Test failure results in notification by means of I/O conditioner status words and a possible suspension of this output (note that for input errors past values were used while accumulating the failure history. The same of course, cannot be done for output errors). The executive interrogates the status words each cycle. When the failure history is completed a full cycle fault isolation routine is entered and the error is isolated to either the computer output unit or to the I/O conditioner.

A problem that arises in the multiprocessor is the ability to isolate errors between processors and I/O units. Generally, certain processor failures, and I/O unit failures, will result in the same conditioner status words. The isolation ambiguity is resolved by taking advantage of the built-in flexible communication paths between each of the processors and each of the I/O units. Thus, one processor can attempt to talk to two I/O units, or two processors can attempt to talk to the same I/O unit. The implementation of this test is dependent on the multiprocessor configuration at the time of the failure. It was further discussed in section 4.2.3.2 dealing with reconfiguration.

6.2.2 Hardware Methods

Hardware methods of failure or error detection shall be considered in this section, each of the modules shall be treated separately in the discussion that follows:

6.2.2.1 Memory

Fault detection methods were considered for three different memory approaches: coincident select semiconductor, and DRO ferrite memories, and the NDRO ferrite memory. Each of these was considered unique enough to warrant separate treatment.

6.2.2.1.1 Coincident Select LSI Semiconductor Memory

The coincident select semiconductor memory organization described in 6.1.2.1 was investigated to determine the hardware fault detection methods that may be employed. The memory organization is shown in Figure 6-12. The selected hardware detection methods will be described below.

The selection of the hardware fault detection method was based on a functional evaluation of the memory. Reference should be made to Figure 6-12in the discussion below. Primarily this consisted of determining whether the word has been written

or read correctly. The address is checked as it enters the memory module from the processor for correct parity. The address is further checked at the output of the address register. Since the output is connected to each of the 4K x 18 memory stacks this then guarantees that the correct address is inputted to each stack.

It should now be noted that any failure in the addressing function from this point on will propagate as a failure in one bit position of the selected word and will be detected by a check on the word as it is read out of the memory. This holds true for single failures as is being considered, of course, the probability of certain combinations of multiple failures will reduce the probability of detection. The fact that only one bit will be affected by any failures may be seen by referring to Figure 6-10 the organization of the memory cell array. As shown in the drawing, each cell array contains its own row and column address decoding and selection circuitry. Each cell array represents one bit of a word and also each cell array receives the same address. Thus any failure in one of the arrays will affect only the corresponding bit.

It was stated that the address is checked and verified correct before it enters each 4K stack. This is accomplished in the block labeled Input Data Control and Address Register. A 14 bit address enters this block, it is parity checked in the address register. Twelve of these bits then are sent to the 4K stacks. The remaining two bits are used to control the selection of one out of the three stacks. This selection is combined with control signals from the Data Transfer Control block to control each of the two control lines to each 4K stack. The signals on each of these two control lines are monitored by a series of logic gates and checked against the original two bits in the address register to determine that the proper control signal was activated. This is a feedback type check on the logic decoding and gating circuitry.

A parity check is performed at the interface to the processors (after the I/O blocks) to detect errors in the data read out of the memory. This parity check will detect all single failures and certain multiple failures from the 4K stacks through the Word Output Gating and the I/O blocks. The Data Transfer Control Block will contain some checking circuitry to detect the issuance of the proper gating control signals. However, the request scanning functions of this block can be checked by the processor. The processor will simply check the time to have a request for a memory cycle honored. If the time exceeds a preset amount then the memory is declared faulty. The results of all the fault detecting circuitry are outputted to the line labeled "fault".

It should be noted here that a small number of logic elements can be included in the fault detection circuitry to validate the fault detection capability. This consists of logic for decoding three addresses to provide inputs to the detection circuitry to simulate a bad state. The fault line may then be sampled by the processor to determine whether it is activated. The parity checker circuits can be checked by injecting a word into the module with incorrect parity and monitoring the "fault' line.

To summarize, the following hardware is recommended for hardware fault detection methods on this type of memory organization: parity checking at the interface to the processors and at the address register in the Input Data Control and Address Register block, stack selection address decoding check by feedback in this same block, and comparison of control signal states out of the Data Transfer and Control block with those entering it from the processors. It is felt that this will provide a high degree of confidence in checking the memory (close to 100%).

6.2.2.1.2 Coincident Core DRO Magnetic Memory

The organization of this memory module is the same as that shown in Figure 6-12 for the semiconductor memory. The major differences are in the blocks labeled 4K stacks. The DRO Magnetic memory consists of a different approach to address decoding and switch selection; sense and inhibit circuitry and a data register are also needed. These are shown in Figure 6-20, a detailed description of the 4K stack block.

To detect failures in this memory module the same discussion as given above to functionally checking the semiconductor memory module applies. However, the 4K stack block for this organization requires some additional fault detection circuitry. The reason for the additional detection hardware required is basically that the bits are not separated as in the semiconductor memory. The address decoding and switch selection circuitry is used to select one row and one column for all bits in the stack. Failures may occur which can cause more than one row or more than one column being energized. As an example if one column is energized and two rows were energized; the coincident current at two cores in each bit plane will be $3/4\ I_S$ where I_S is the normal full coincident current used to switch the cores. The cores then may, may not or might possibly switch with this current depending on the core stack design. It is therefore possible to have a random effect with regards to the word read out of the core stack with this type of failure.

Failures such as this dictated special hardware for detection. The approach taken was to monitor the driver and sink switch selection right at the input to the core stack. As shown in Figure 6-20 several logic gates are used to monitor the selection and compare this with the address sent to the 4K stack block. This checking determines that the proper selection has been made, and that not more than one driver or switch has been turned on. Therefore, since the address into this block has been checked for correctness, the monitoring circuitry verifies that the correct address has been selected. This is a feedback type check on decoding circuitry. It should also be noted that six addresses are decoded to be used as a test on the detecting circuitry. Whenever, these addresses are selected a fault output signal from the memory module must be present. The detecting circuitry shown in Figure 6-20 represents only a quadrant of a 4K stack, this circuitry is therefore quadrupled for the entire block.

There may also be faults in this block which are not detected by this monitoring circuitry. However, these faults will be detected by the parity check on the word readout. Examples of these faults are those in the sense or inhibit amps or in diodes associated with the driver and sink switches.

Referring once again to Figure 6-12 this overall memory module block diagram requires some additional hardware for fault detection in the blocks labeled 4K stacks as discussed above; in addition to that hardware and that discussed for the semiconductor memory some additional circuitry has to be placed in the Word Output Gating block. Three additional lines are inputted to this block (the read lines from the Input Data Control and Address Register block), these lines simply control the outputting of the proper 4K stack. This requires an extra gate for every bit line into the Word Output Gating block. What this hardware does is it prevents two stacks from outputting a word simultaneously. This situation could occur if one stack had its read control and timing circuitry frozen true. As noted previously this simultaneous output of more than one word can produce random effects that are

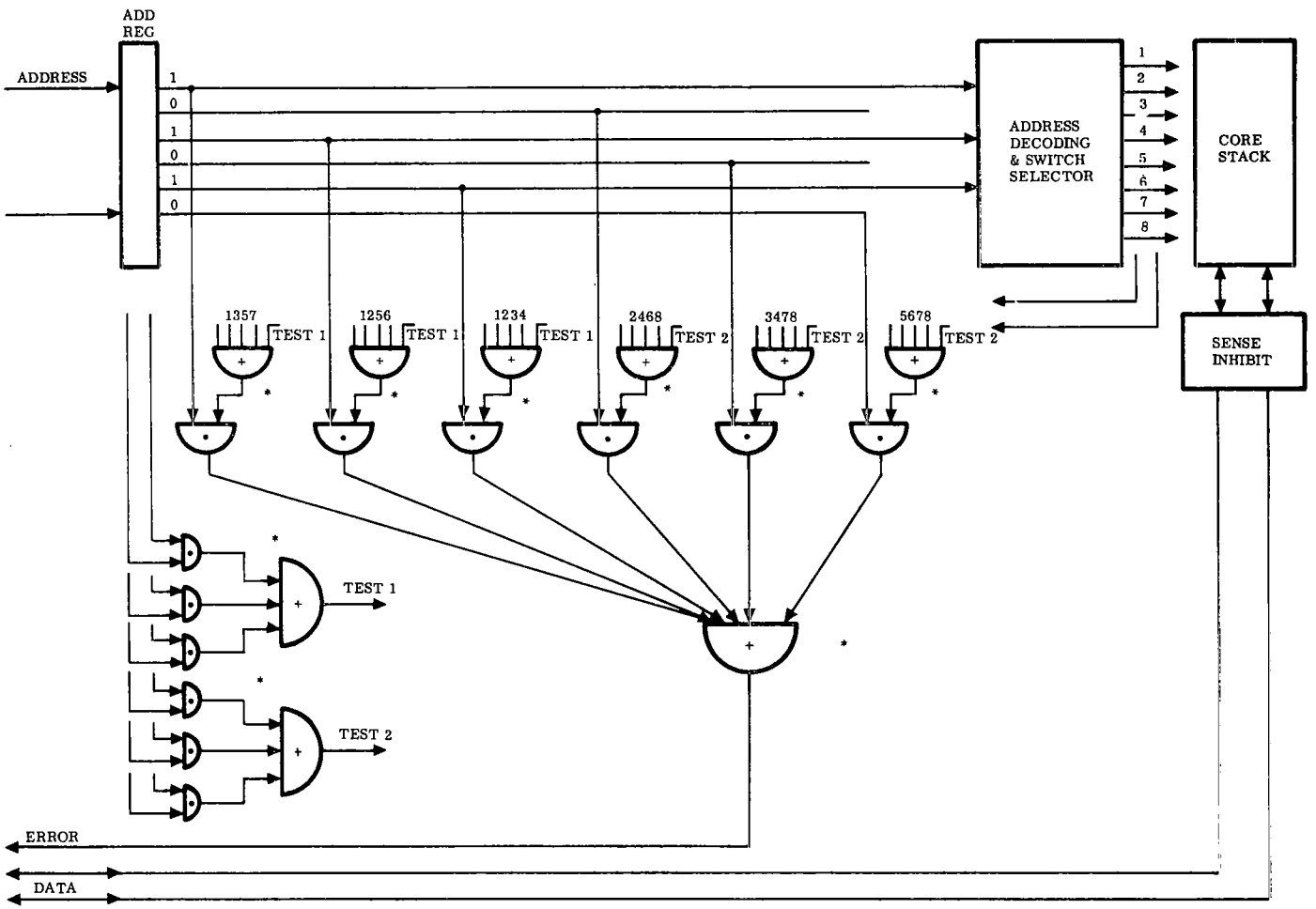


Figure 6-20. Coincident Current Memory -4K Stack Fault Detection

difficult to detect. The circuitry described above essentially prevents the occurrence of these faults.

6.2.2.1.3 Linear Select NDRO Ferrite Memory

The organization of this memory module is given in Figure 6-18. Functionally it is somewhat similiar to that described in Figure 6-12 for the other two memories. The main difference is that a 12K stack is used in place of the three 4K stacks. This eliminates the need for the Word Output Gating block. In addition the Input Data Control and Address Register block is considerably simpler since it is not required to select one out of three stacks.

With regards the fault detection problem, the stack selection address checking circuitry in the Input Data Control and Address Register block is no longer required. The organization of the 12K stack is shown in Figure 6-18. This block is functionally checked by the same approach described for the Coincident Core DRO Memory 4K stack described above. However, the exact structure of the selection circuitry monitoring gates differs slightly from that shown in Figure 6-20.

To check the word line drivers and sink switches the output of each line is checked at the input to the stack by the feedback approach described in paragraph 6.2.2.1.2 to determine that the proper 1 out of 32 drivers and 1 out of 32 sinks was selected. In addition, the selection switches for controlling the write selection gates and the bit output gates are monitored in the same manner and compared with the 4 bits of the address reserved for this function to determine that the proper one out of 12 was selected. Parity checking is performed functionally and in the same places as with the other two memories.

6.2.2.2 Processor

The processor module was investigated to determine the hardware methods that would be used for hardware fault detection. Reference should be made to paragraph 6.1 Figure 6-6 for details on the processor.

6.2.2.2.1 Data Transfers

Data transfers will be checked by testing the parity of the contents of a register after it receives new data. For this purpose, every register whose contents are thus checked will be connected to a parity bus. The parity bus will be connected to a parity checker, which will generate an alarm when incorrect parity occurs.

The segments of an instruction are stored in several registers. The parity of the information transferred to these registers will be checked by concatonating their contents. This parity checking will provide a check on all the registers in the processor section and the communication to and from them (U_1, U_2, L, MB, B_1, B_2, T_1--T_7, OER, IR, TR, SR in Figure 6-6.

6.2.2.2.2 Adder

The operations performed by an adder will be checked and a parity bit will be concatonated with the output operand before the result is transferred to a register. Since some operations will be checked by parity, the output of the adder will be gated into the parity checker.

Since the structure of the adder is not firm, the plans for checking the operations are tentative. However, the following techniques are contemplated.

AND - The results of a logical multiplication can be obtained from the carries if the carry inputs to the adder bits are set equal to zero. The carries will be checked and their parity can be generated and concatonated with the result. Checking parity of the result will detect errors between the generation of carries and their transmission to the adder output. The parity bit will be generated in a parity generator capable of calculating the Mod 2 sum of the parities of the adder inputs and the carries. However, the parities of the adder inputs will in this situation be treated as though they were equal to zero.

OR - The results of a logical addition can be checked by parity. If A and B are two words, then

$$\left.\begin{aligned} P(A) &\equiv \Sigma A_i \\ P(B) &\equiv \Sigma B_i \\ P(AB) &\equiv \Sigma A_i B_i \\ P(A\bar{B}) &\equiv \Sigma A_i \bar{B}_i \\ P(\bar{A}B) &\equiv \Sigma \bar{A}_i B_i \end{aligned}\right\} \quad (\text{mod } 2)$$

It can also be seen that

$$\left.\begin{aligned} P(A \cup B) &\equiv P(A\bar{B}) \oplus P(\bar{A}B) \oplus P(AB) \\ P(A) &\equiv P(A\bar{B}) \oplus P(AB) \\ P(B) &\equiv P(\bar{A}B) \oplus P(AB) \\ P(A) \oplus P(B) \oplus P(AB) &\equiv P(A\bar{B}) \oplus P(\bar{A}\bar{B}) \oplus 3P(AB) \\ &\equiv P(A\bar{B}) \oplus P(\bar{A}B) \oplus P(AB) \end{aligned}\right\} \quad (\text{mod } 2)$$

Hence

$$P(AUB) \equiv P(A) \oplus P(B) \oplus P(AB) \quad (\text{mod } 2)$$

Now P (AB) is the parity of the carries when the carry inputs to the adder bits are zero. Thus, a parity bit can be obtained from the sum of the parities of the carries and the adder inputs.

EXCLUSIVE OR - In exclusive or, the parity of the result is equal to the sum of the parities of the inputs. Hence, the parity bit will be generated by regarding the carries as equal to zero.

ADDITION, SUBTRACTION - In addition or subtraction, the parity of the result is:

$$P(A \pm B) = P(A) + P(B) + P(C)$$

where C is the set of carries. The carries are checked independently of the over-all parity check and then fed into the parity generator along with the parities of the inputs.

The structure of an adder making use of parity checks for addition and subtraction is shown in Figure C-3 of Appendix C.

6.2.2.2.3 Decoders

Decoders will be checked by feedback to determine whether or not the correct signal and only that signal is generated. The approach is identical to that shown in Figure 6-20 used for checking the address decoding for the ferrite memory organizations. If a control memory is used for the instruction decoding and control signal generation function the detection hardware would be organized the same as given in Figure 6-20. It should be noted that conventional logic decoders can be checked in a similar manner; however, some additional detection circuitry would be needed within the logic net.

6.2.2.2.4 Gating Signals

Gating signals will be checked by parity. All the gating signals together with a set of pseudo-gating signals are transmitted to a parity checker. Parity should be odd at all times except for transient intervals during which parity is not examined.

One pseudo-gating signal is a parity signal. This is generated for each clock period of each instruction. Its truth value is determined so that odd parity is generated.

Another type of pseudo-gating signal is used in connection with a conditional gating signal. For a given combination of instruction and bit time, a gating signal will be generated if and only if certain conditions are true. For these combinations of instruction and bit time there should be another gating signal or pseudo-gating signal which is generated if and only if these conditions are false. Then, regardless of the truth value of the set of conditions, exactly one of these gating signals or pseudo-gating signals will be generated for that instruction and bit time. Hence the parity of the entire set of gating signals and pseudo-gating signals will not depend on the truth value of these conditions.

A third type of pseudo-gating signal is used to guarantee that the fan-out of any gate includes an odd number of gating signals which would be in error as a result of an error in the output of the gate.

6.2.2.2.5 Control Flip-Flops

The states of the control flip-flops (RM, PMR, IMR, etc., in Figure 6-6) can be checked by parity, special redundancy and duplication.

The parity of a set of flip-flops is known if one of the following is known:

1. The initial parity and the number of triggerings.

2. The parity of the set of values inserted into the flip-flops.

Thus, if either of these is known, a parity check can be used to test the correctness of the configuration of states.

For some sets of flip-flops parity may not be sufficient. There may be redundancies which would fix the parity even though a single fault is present. For example, a set of flip-flops may be designed so that any one of them should be set. If the wrong one were set, parity would remain odd. In this situation it might be possible to combine the flip-flops into subsets. Then if the flip-flop which is set is in the wrong subset, an error is detected. This is an example of the special use of redundancy.

A preliminary examination of the control flip-flops indicates that they may simply be checked by parity.

6.2.2.2.6 Counters

Counters can be checked by duplication or by the use of unit distance counters. Unit distance counters appear the most attractive in terms of adding little additional hardware; however, problems are encountered since most data loaded into the counters will be in regular binary code. This means that code conversion must be provided either via software or hardware. Time has not permitted investigating this topic further; therefore, a specific recommendation for checking the counters cannot be made.

6.2.2.2.7 Clock Pulse Stream

A fault in the clock-pulse stream is detected by the charging or discharging of a capacitance. A flip-pulse is triggered by every clock pulse. Its output will charge or discharge a capacitance in accordance with the state of the flip-flop. Hence if the flip-flop changes state every pulse period, the resulting voltage will assume a correct stable value. However, if the flip-flop remains in the same state, the voltage will go to a value which is either higher or lower than that correct stable value, depending upon the state of the flip-flop. When the voltage goes above or below a threshold, an alarm is generated.

6.2.2.2.8 BITE Timing Circuitry

This hardware is used in conjunction with software checks. As explained in that section the instruction sequencing test is used to control the setting and resetting of a flip flop which is used to charge a capacitance and deviation from a nominal value will be detected by tolerance circuitry which controls the generation of an alarm.

6.2.2.2.9 Request Timer

A simple one-shot timer can be used to insure that a request signal is honored within some period of time. This time would have to be set to the worst case permissible. It should be noted that this is a check on the memory modules and the interface therein.

The above discussion presented hardware detection methods for the processor. It can be seen that the processor requires relatively more hardware than the memory for fault detection. It is doubtful that it would be necessary to add in all the above hardware checks when hardware and software fault detection methods are combined. A preliminary feeling for this indicates that: (a) parity checking on data transfers, (c) feedback checking on decoders, (e) parity check on the state of control flip-flops, (h) BITE Timing Circuitry, and (i) Request Timer as described above would probably be used. The first three are relatively inexpensive and checkout a large portion of the processor and the last two may almost be required since it is difficult to replace them entirely with software. The remainder of the hardware detection methods would probably be subject to extensive trade offs with software methods.

6.2.2.3 Input/Output

Hardware approaches to fault detection for the I/O module are given below. Reference should be made to Figure 6-19 for details on this module.

6.2.2.3.1 Transfers

Transfers of data words, control words and addresses between the memory and the I/O will be checked by a parallel parity checker. The registers involved in these transfers will be connected to a parity bus, also connected to this bus is a parity checker.

Data transfers between the Assembly Shift Register and the Conditioners are checked by parity. To check or generate parity at the Conditioner in question, a flip-flop can be triggered once for each one transmitted in the serial channel. If necessary, an extra bit position can be filled with a zero to make even the total number of bit positions. Then it would be possible to check odd parity over an even number of bit positions. The parity check would then guarantee that at least one has been transmitted. Otherwise, the parity would be even for an all zero word. It would also guarantee that at least one zero has been transmitted. Otherwise there would be one from each of an even number of bit positions to give even parity.

Data transfers between the Buffer Register and the Bulk Storage would also be protected by parity checks. The thoroughness of the protection will depend upon the final design of the Bulk Storage. These parity checks will also provide fault detection capability for the two aforementioned registers and the MR register

6.2.2.3.2 Counters

The same comments as given for the processor apply here. No final recommendation has been reached for the counters.

6.2.2.3.3 Decoders

Decoders for selecting specific conditioners or devices are checked by feedback of the acknowledge signal. Each feedback signal is compared with each bit of the register if any bit of the register is inconsistent with the feedback signal, an alarm is generated.

6.2.3.3.4 Acknowledgements

Control signals transmitted between the I/O and another module to initiate cooperative actions are protected by acknowledgements. Associated with each control signal of this type, there is a time delay. If the acknowledgement does not arrive before the time delay expires, an alarm is initiated. If an extraneous acknowledgement is received, an alarm should be generated. The control flip-flops whose states determine the appropriateness of the acknowledgement are used to detect the extraneous acknowledgement. This is accomplished as explained under "Decoders."

The time delay associated with an acknowledgement is designed to expire when

1. The delay in the acknowledgement is longer than the maximum delay in a good device.

2. The delay in the acknowledgement is longer than the requesting device can tolerate.

Situation (1) always calls for an alarm leading to a roll-back or reconfiguration. Situation (2) may call for such an alarm. On the other hand, it could result from congestion or latency.

6.2.3.3.5 Control Flip-Flops

Same comments as given in the processor section apply here.

6.3 EXECUTIVE PROGRAM

In the following paragraphs, flow diagrams of the major executive functions are presented (Figure 6-21). Several areas are not included: the effect of I/O and error detection management on the I/O Supervisor, message processors, and the self-test routines.

Following is a list of the entries referenced in the diagrams; those that are not designed are marked with an asterisk (*): (reference should be made to Paragraph 4.2 for general discussions on these routines).

PROGRAM SEQUENCER OR SCHEDULER

1. PIE (Program Interrupt Entry) – Entered from the RTC zero-transition interrupt. Performs basic scheduling of periodic, request and background programs.
2. PPT (Periodic Programs Termination) – Point at which periodic programs return to PIE when completed.
3. RPT (Request Programs Termination) – Point at which request programs return to PIE when completed.
4. BPT (Background Programs Termination) – Point at which background programs return to PIE when completed.
5. FSE (Fill Start Entry) – Entered from the fill function. Status is saved and PIE is entered to schedule background computation.
6. FEE (Fill Exit Entry) – Entered from the fill clock zero transition interrupt. The background status is saved and the program issuing the fill request is resumed.

RECONFIGURATION

7. PLE (Phased Loading Entry) – Entered as a NOW request program. Accomplishes loading and initialization of a new program configuration without closing down the current periodic computations until the new periodic programs are ready to take over.
8. CSE (Cold Start Entry) – When a dormant processor-memory-I/O configuration is activated, control is eventually passed to this entry with RTC already at zero. Preparations are made for entry into PLE for loading of a program configuration.
9. RLE (Request Loading Entry) – Used to load request programs on a priority basis.

REQUEST PROCESSOR

*10. RSE (Request Select Entry) – Entry for selecting request programs for execution with specified priorities.

11. RME (Request Monitor Entry) – Places selected request programs in the request scheduling queue according to priority.

*12. CMP (Console Message Processor) – Reads, verifies and initiates action in response to directives from the console.

SELF TEST

*13. PST (Periodic Self-Test) – Performs whatever is necessary at the RTC frequency.

*14. CSC (Comprehensive Self-Check) – Complete validation of the system (hardware and software). Used only from CSE and fault isolation programs.

15. FFE (Forced Fault Entry) – Creates a failure condition for the processor - memory combination by issuing a pulse stream flip-flop signal out of synchronization.

I/O

*16. (READ) – Used to transmit information from Mass storage to main memory.

A data base is assumed in these designs. Table 6-2 lists these parameters with cross-references to the entries which use them:

Table 6-2. Executive Data Base Table

Parameter	Description	Entries
$P(i)_{consists\ of}$	Entry for i^{th} periodic program in the Periodic Scheduling Table (P-table)	PIE, PLE, CSE
$t_p(i)$	iteration count for period control	
$f_p(i)$	frequency count	
$e_p(i)$	early I/O entry	
$x_p(i)$	execution entry	
$c_p(i)$, $m_p(i)$, & $l_p(i)$	checksum data: checksum value, check start point & check segment length	
$S_p(i)$	interrupt register save area	
$R(i)_{consists\ of}$	Entry for i^{th} request program in the Request Scheduling Table (R-table)	PIE, PLE, RLE
$x_r(i)$	execution entry	
$Q_r(i)$	queue Pointer	
$c_r(i)$ $m_r(i)$ $l_r(i)$	checksum data: checksum value, check start point & check segment length	

Table 6-2. (Cont)

Parameter	Description	Entries
$B(i)_{consists\ of}$	Entry for i^{th} background program in the Background Table (B-table)	PIE, PLE
$X_b(i)$	execution entry	
$Q_b(i)$	pointer to next background	
$c_b(i)$ $m_b(i)$ $l_b(i)$	checksum data: Checksum value, check start point & check segment length	
S_I	Hard-wired area for interrupt system storage of registers	PIE, FSE, FEE
S_R, S_R^*	Primary and secondary areas for request programs status retention	PIE
S_B	Status retention area for background programs	PIE, FEE
S_X	Status retention area for fill function users	FSE, FEE
E_S	Empty Storage capacity	RLE
RTC	Primary interrupt system clock	PIE, CSE
FC	Fill function interrupt clock	PIE
p*	Index of periodic program currently in execution	PIE
p**	Index of periodic program last interrupted	PIE
n_p	Number of periodic programs	PIE
r*, r**	Indices of currently executing and last interrupted request programs	PIE
r_o	Index of NOW request program	PIE, RME, RLE
r_n	Index of NEXT request program	PIE, RME, RLE
r_a, r_x	Indices of first request programs on the ASAP and non-priority queues	PIE, RME, RLE

Table 6-2. *(Cont)*

Parameter	Description	Entries
r_i^*	Index of request program being executed by "this" processor	PIE
r_j^*	Index of request program being executed by "opposing" processor	PIE
r_c	Index of request program selected for execution	PIE
b*	Index of background program currently in execution	PIE
b_r	Index of next background program to be executed	PIE
$P^{\#}$	Next phase number	PLE
$R^{\#}(j)_{consists\ of}$	Entry for j^{th} request program in the request board	RME
$R_p(j)$	Priority assigned	
$R_j(j)$	Select/no-select flag	
$R_n(j)$	Request program index	
r_v	Index of request program being loaded	RLE
r_r	Number of entries in the Request Board	RME

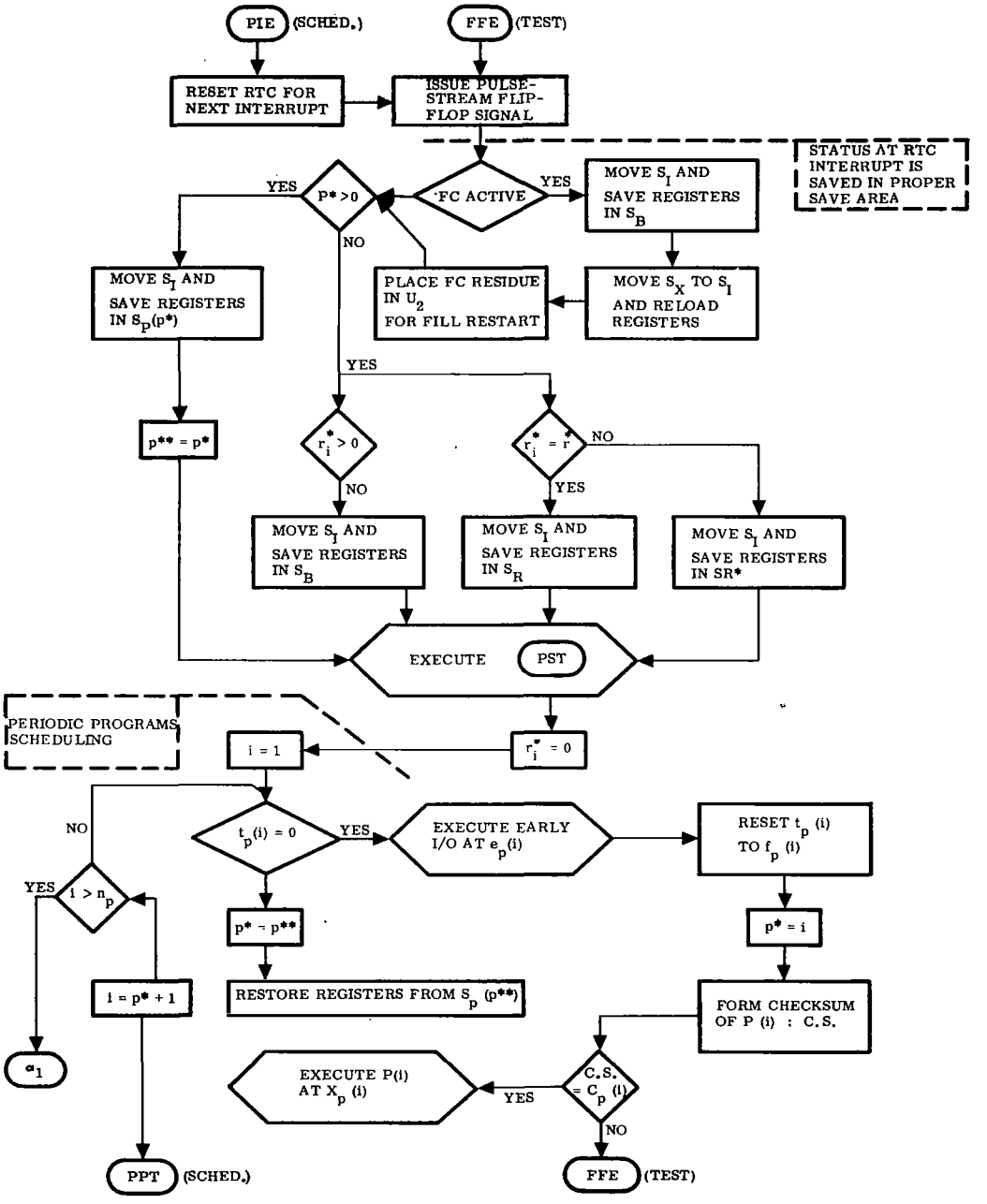


Figure 6-21. Executive Flow Diagrams (Sheet 1 of 8)

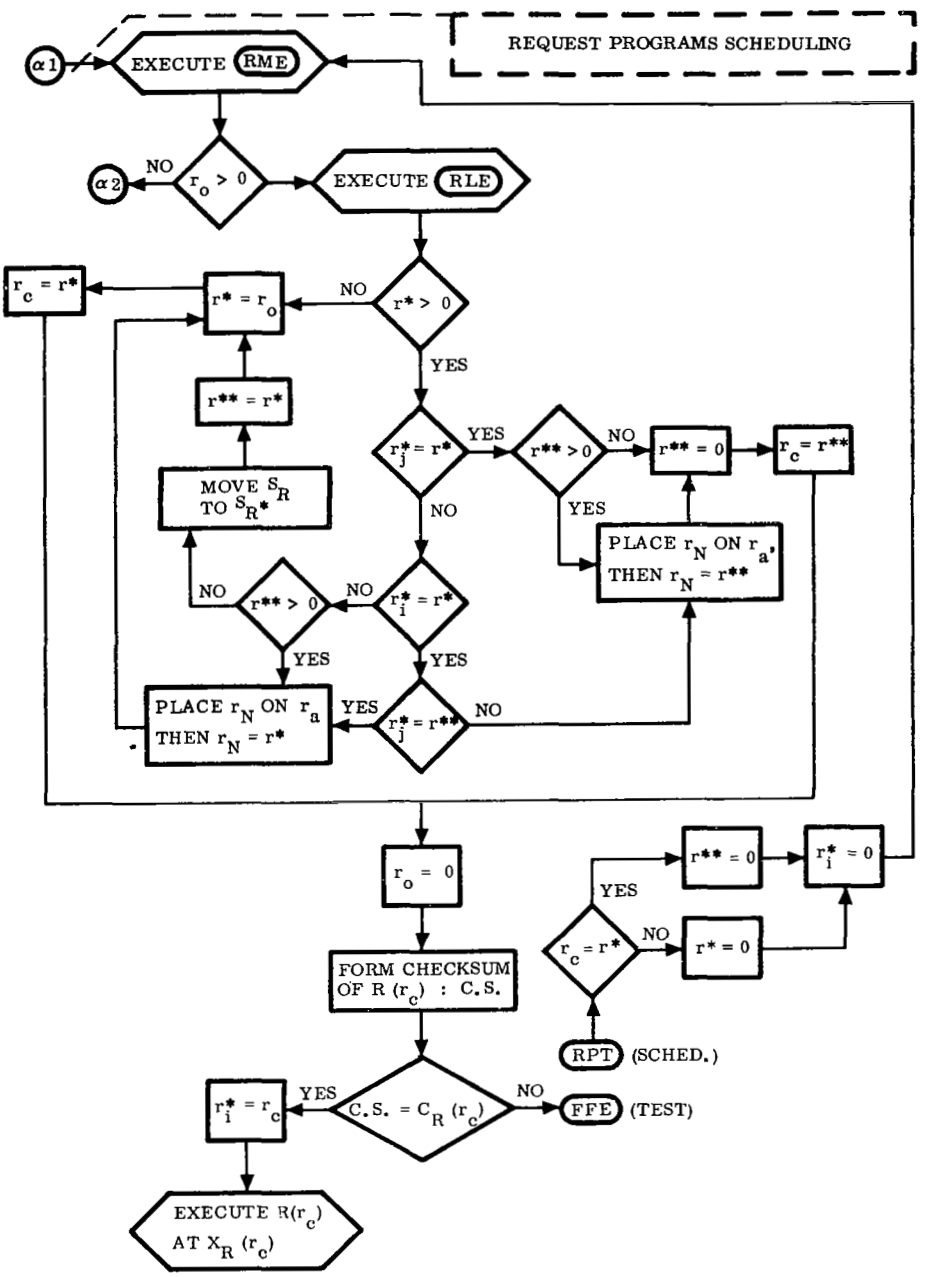


Figure 6-21. Executive Flow Diagrams (Sheet 2 of 8)

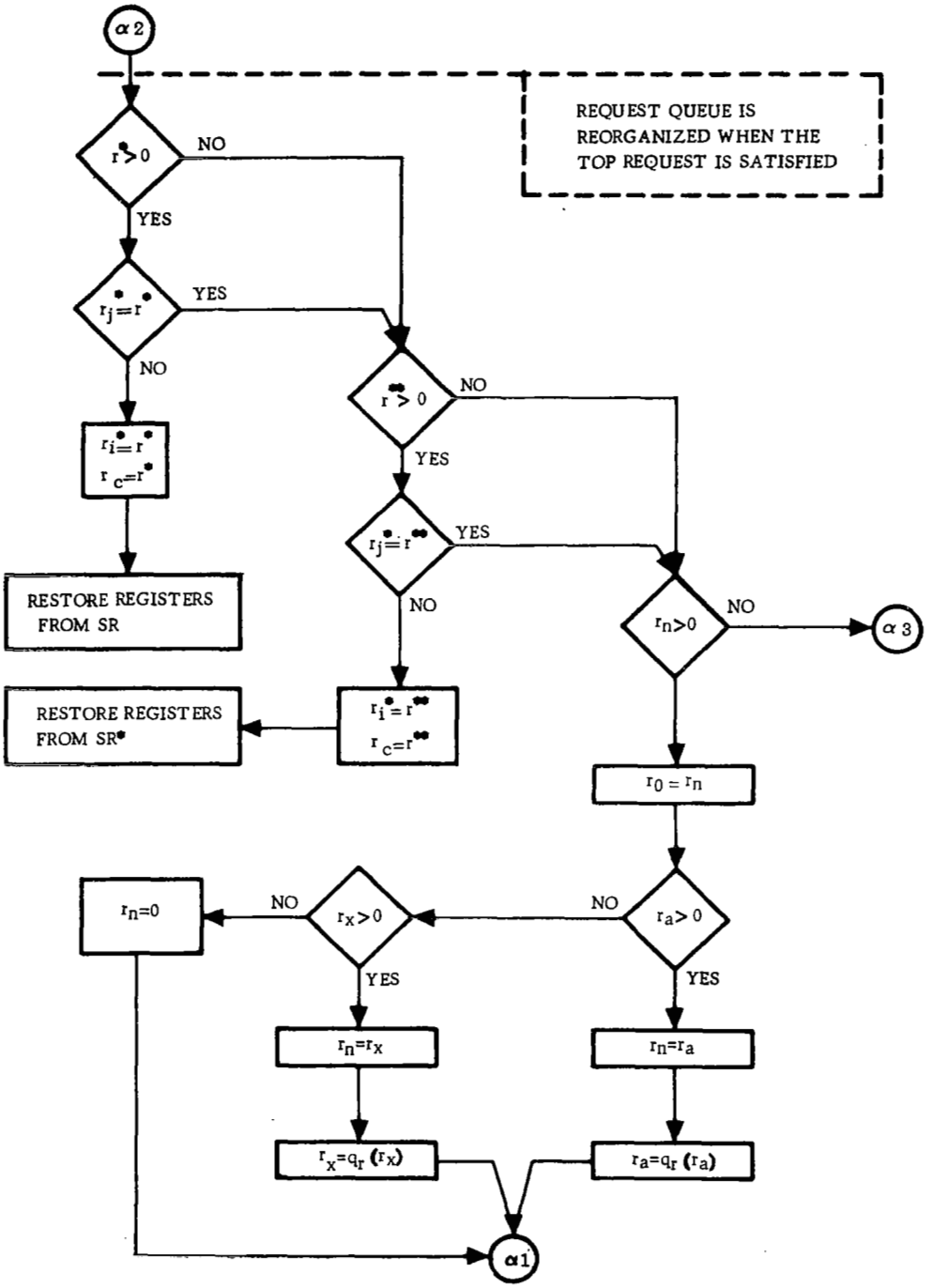


Figure 6-21. Executive Flow Diagrams (Sheet 3 of 8)

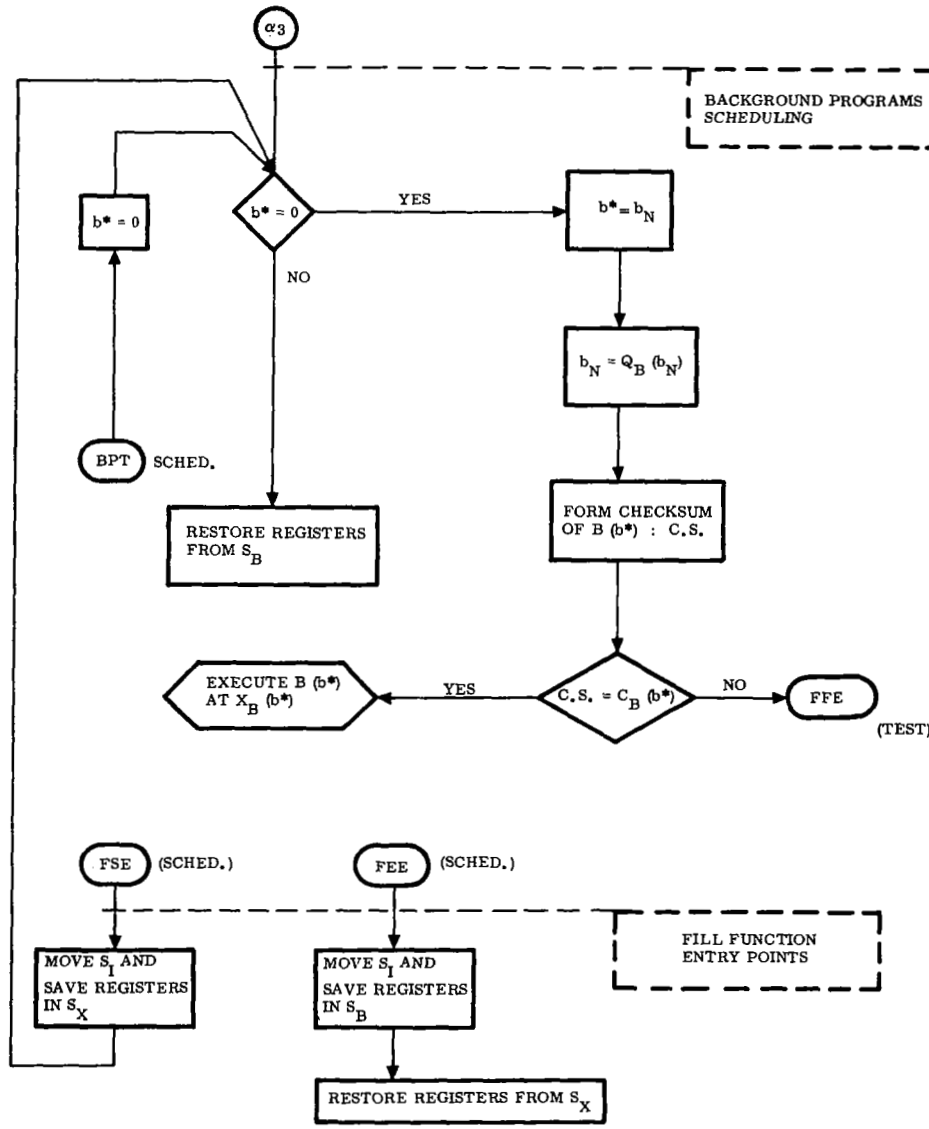


Figure 6-21. Executive Flow Diagrams (Sheet 4 of 8)

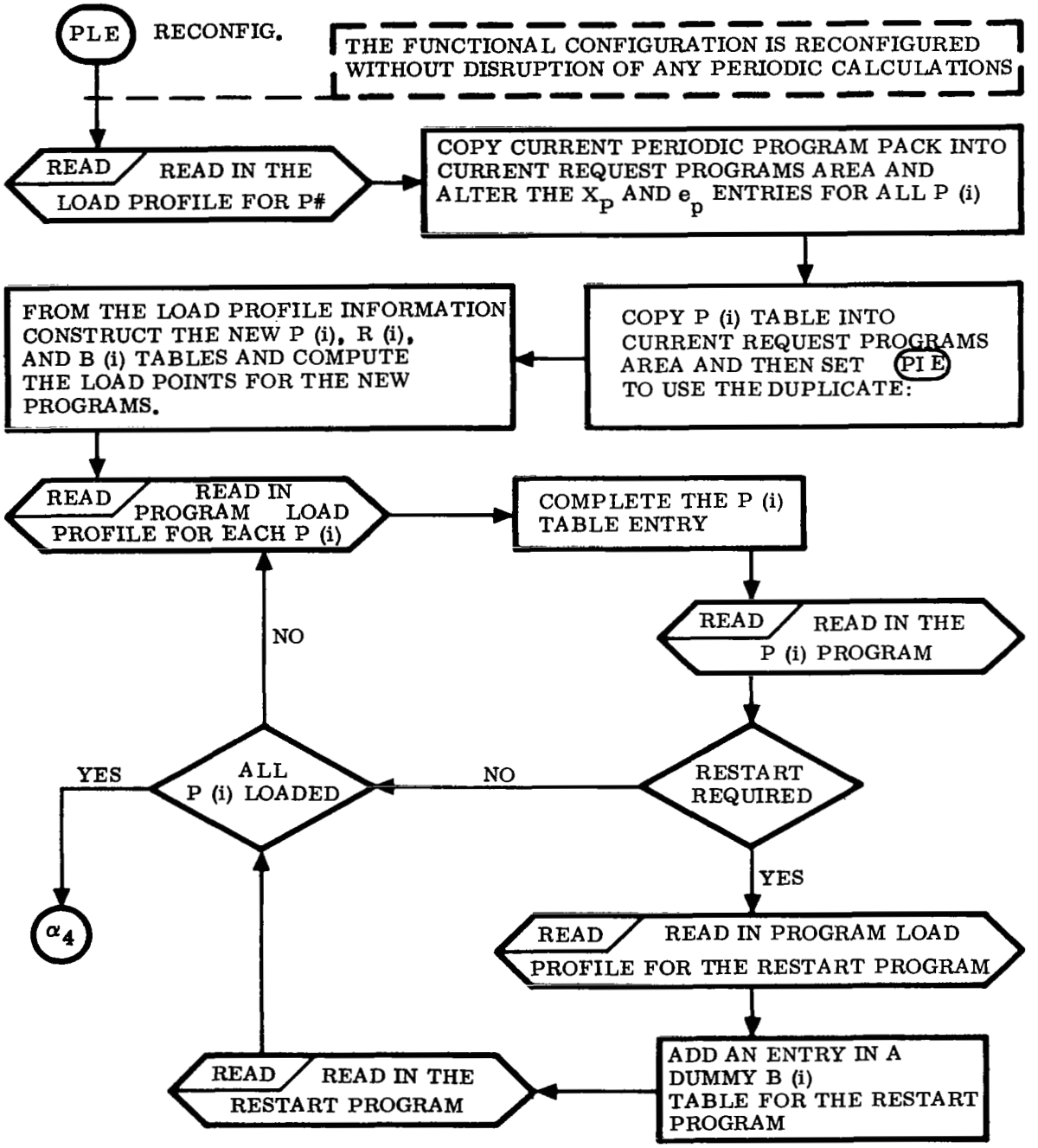


Figure 6-21. Executive Flow Diagrams (Sheet 5 of 8)

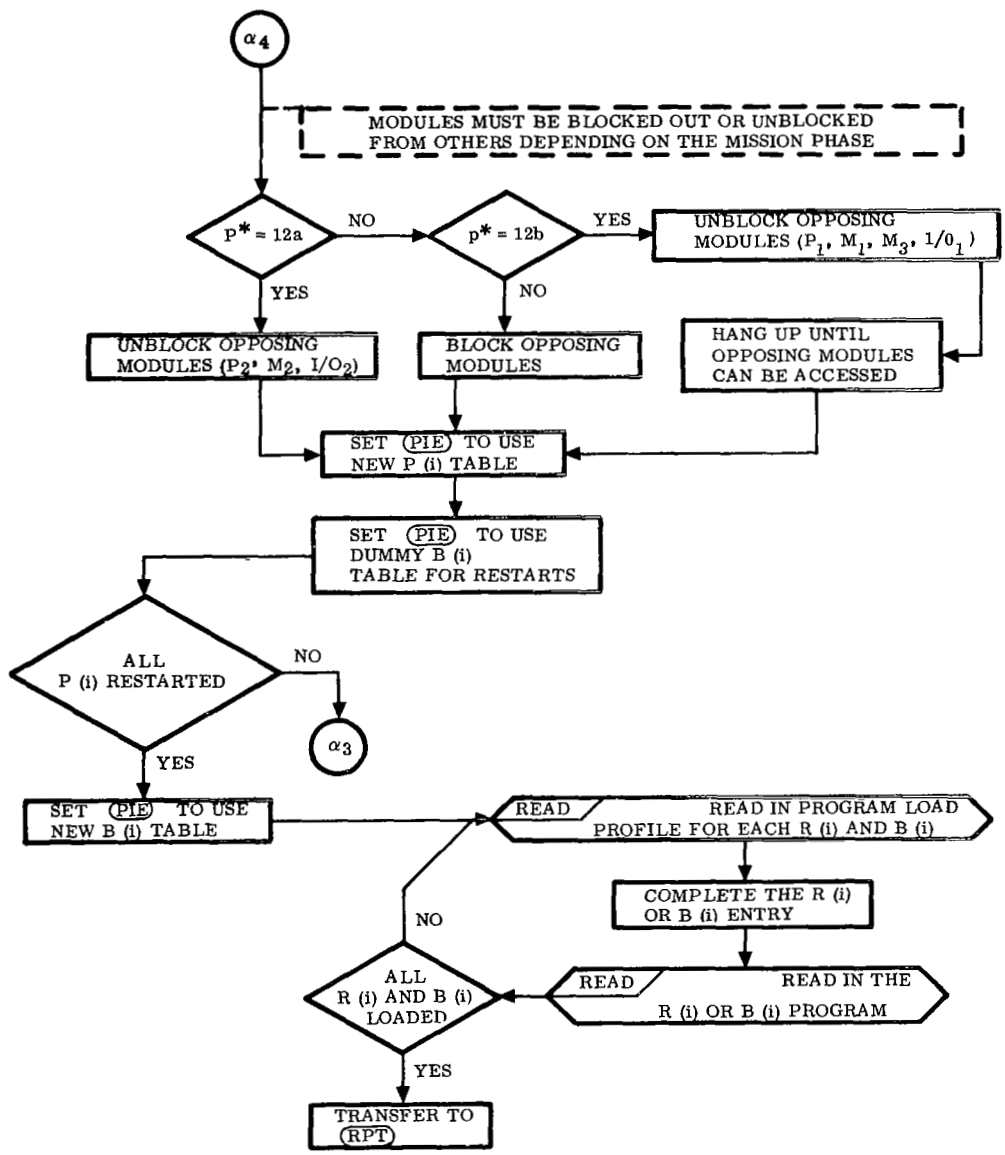


Figure 6-21. Executive Flow Diagrams (Sheet 6 of 8)

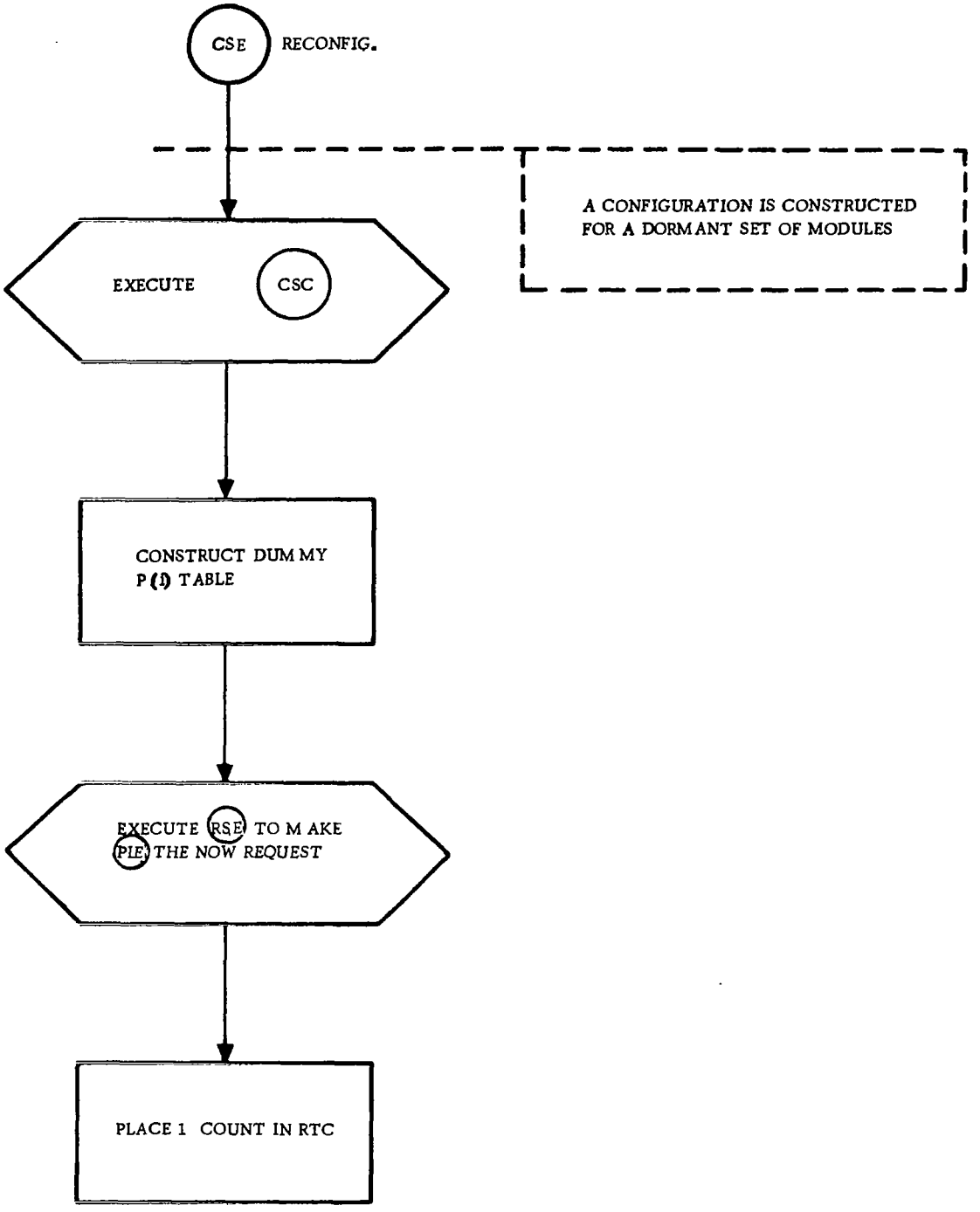


Figure 6-21. Executive Flow Diagrams (Sheet 7 of 8)

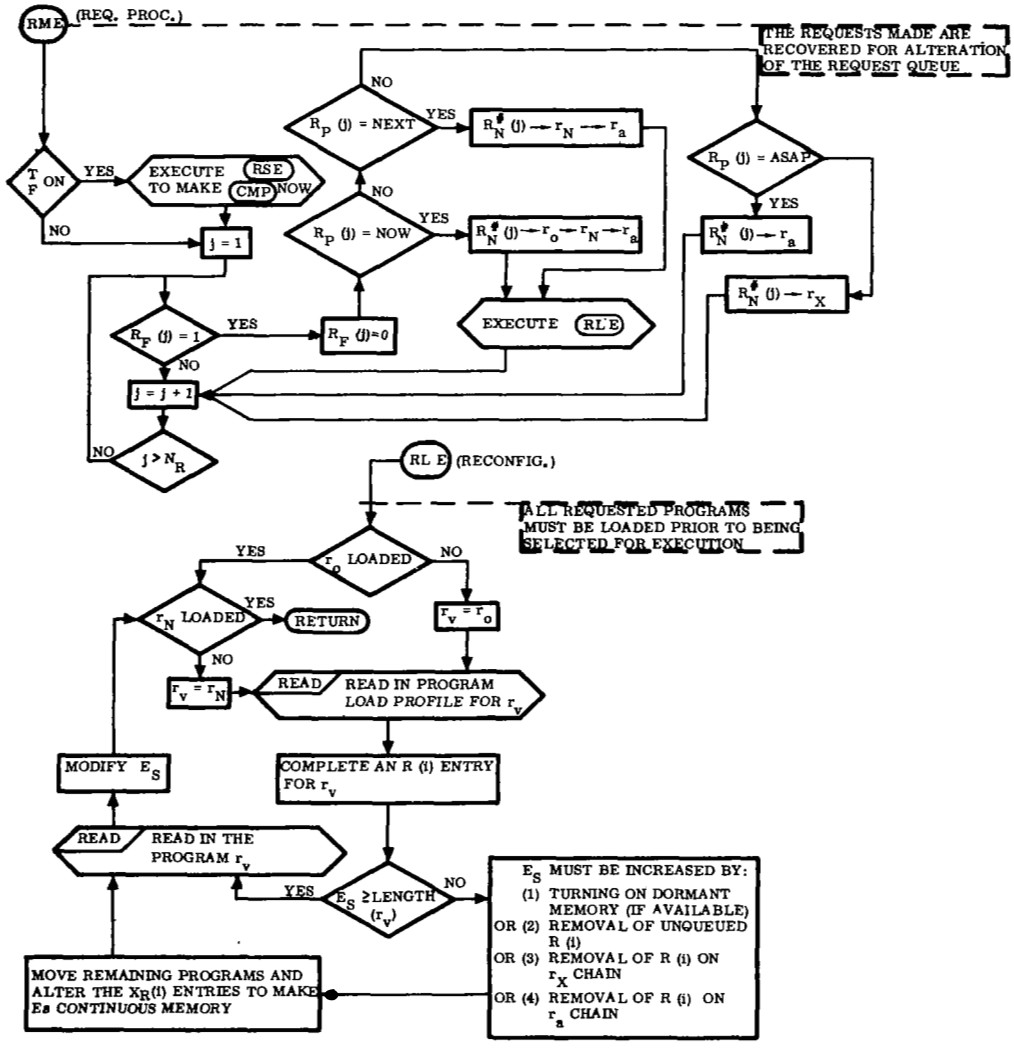


Figure 6-21. Executive Flow Diagrams (Sheet 8 of 8)

VII. SUMMARY AND RECOMMENDATIONS

A summary of the work accomplished during this study along with areas requiring further investigation is given below. Computer requirements were defined for what is considered as a representative future manned space mission, the Mars Lander Mission. Some significant points about the requirements are the widely varying computer requirements in terms of speed and storage from phase to phase and the critical nature of the computations during certain phases such as atmospheric entry. The requirements have a large influence on the computer design and there are many areas which could not be covered completely primarily due to a lack of data. Some of the more important areas are: (a) the interface between the computer and the sensors, information is lacking as to the nature of the signals expected, (b) precision requirements for various computations needs to be firmly defined, also the question as to whether floating point is needed for some navigation and guidance functions should be decided, (c) the structure of a reliable bulk storage unit for the time period of interest should be investigated and (d) the investigation and design of ultra reliability switch networks. Any future requirements studies should take into account the above points.

Three types of multiprocessing organizations were presented. These organizations span technology that may be considered as state of the art to that expected over the next 10 years. The modular multiprocessor organization was selected for a detailed investigation. All the functional features of the memory, processor, and Input/Output modules have been designed. The next step in the design of the computer would be a detailed logic design and layout. The modular multiprocessor was shown to meet the requirements of this mission quite efficiently. Its organization permits the turning on and off of any module and thereby providing a good match to the diverse computational requirements from phase to phase. It also provides the capability for a significant enhancement in probability of success and availability as was shown in the simulation results. This is due to the reconfiguration around failures at the module level and the increase in reliability due to turning modules off given that dormant failure rates are lower than operating failure rates.

The simulation pointed out two points that need further investigation: (a) the relative difference between dormant and operating failure rates needs to be determined, this factor has a profound effect on the computer design and (b) monte carlo methods become increasingly expensive in terms of computer time as the organizational complexity is increased and analytical methods should be developed for complex missions such as this one.

With regards to the functional design of the three organizations, the distributed processor is the one which needs more work in all areas (logic design, failure analysis, software) to further assess its advantages and disadvantages. The area of the modular multiprocessor which should receive further study is the input/output software mechanization and in particular the software utilization of I/O failure detection information.

Software and hardware failure detection methods have been investigated, it is felt that the relative importance of intermittent type failures is an area in need of immediate attention to determine the relative use of these two methods for failure detection.

APPENDIX A. DETAILED COMPUTER REQUIREMENTS

This appendix contains a tabulation of computer requirements on a per phase basis (Table A-1). The requirements for each phase are given for each of the four basic functions: (a) Navigation and Guidance, (b) Tele-Communications, (c) Experiment Data Processing, and (d) System Checkout. These four functions are further broken down into sub functions for each phase. Section II, 2.8 contained a discussion of each of these sub functions. It should be noted that the requirements are given as sub totals for each of the four functions for each phase and also as totals for each phase.

In addition a "Cumulative Total" is tabulated for each phase. This cumulative total is the total storage that would be required in the computer if the programs for each phase were commulatively stored in the computer (this is the case if a bulk storage unit is not used). The storage requirements are not directly added on to each other from phase to phase since many of the functions are identical in more than one phase; as an example the tele-communications functions of the trans-Mars coast and trajectory correction are identical. The storage requirement tabulated as cumulative would exist in a computer that is not reloaded with new programs from phase to phase.

It should be noted that these requirements were derived before an 18 bit computer with indexing and other features previously discussed was decided upon. Therefore, these requirements do not take into account word length required in arriving at storage and speed, that is, no half length or double length considerations are made and no special features such as indexing, indirecting, or multiple accumulators are assumed available. The same general discussion as given in Section II, 2.9 holds true for deriving the computer requirements and for overhead requirements (executive, I/O, etc.).

Figure A-1 contains the requirements in a graphic form. The storage is plotted as a solid line and a dotted line; the dotted line is the cumulative storage plot.

Table A-1. Computer Requirements

Requirements		Storage			Speed		Word Length
		Instr	Const (Words)	Var	Short (Operations/second)	Long	Bits
1.	Phase 1 Atmospheric Ascent						
1.1	Navigation and Guidance						
1.1.1	Process Accelerometer Outputs	320	26	3	240	37	
1.1.2	Navigation Computation	290	16	9	488	29	
	Subtotal	610	42	12	728	66	

Table A-1. (Cont)

Requirements		Storage			Speed		Word Length
		Instr	Const	Var	Short	Long	Bits
		(Words)			(Operations/second)		
1.2	Telecommunications	800	50	50	1000	200	
1.4	Status Monitoring	1000	300	60	1000	120	
	Total	2410	392	122	2728	386	
2.	Earth Orbital Coast						
2.1	Navigation and Guidance						
2.1.1	Attitude Reference	1756	422	183	11520	3320	30
2.1.2	Landmark Tracker Operation	517	313	8	9483	3000	30
2.1.3	Orbit Determination Computation	1478	88	230	7576	433	30
2.1.4	Navigation Computation	573	18	39	493	47	30
	Subtotal	4324	841	460	29072	6800	
2.2	Telecommunications	Same as 1.2			1000	200	
2.4	Status Monitoring	1.4 plus 3000	1000	200	5000	1000	
	Total	9124	2191	770	35072	8000	
	Cummulative Total	9734	2233	782			
3.	Trans Mars Injection						
3.1	Navigation and Guidance						
3.1.1	Process Accelerometer Outputs	Same as 1.1.1			480	74	
3.1.2	Navigation Computation	Same as 1.1.2			976	58	
3.1.3	Required Velocity Computation	1000	40	60	10000	3000	
3.1.4	Velocity to be Gained Steering	200	10	20	3000	700	
	Subtotal	1810	92	92	14456	3832	

Table A-1. (Cont)

Requirements		Storage			Speed		Word Length
		Instr (Words)	Const	Var	Short (Operations/second)	Long	Bits
3.2	Telecommunications	Same as 1.2			1000	200	
3.4	Status Monitoring	1.4 plus 1000	300	60	3000	350	
	Total	4610	742	262	18456	4382	
	Cummulative Total	11934	2583	922			
4.	Trans Mars Coast						
4.1	Navigation and Guidance						
4.1.1	Attitude Reference	Same as 2.1.1			11520	3320	
4.1.2	Navigation Computation	1000	30	50	3000	1000	
4.1.3	Velocity to be Gained (Monitor)	1500	25	60	2500	900	
	Subtotal	4256	477	293	17020	5220	
4.2	Telecommunications	1.2 plus 2000	100	1000	5000	500	
4.3	Scientific Experiments						
4.3.1	Data Compression	1121	414	2009	6591	525	12
4.3.2	Sequencing	750	125	150	200	2	16
4.3.3	Pointing and Control	500	50	25	1000	400	16
	Subtotal	2371	589	2184	7791	927	
4.4	Status Monitoring	Same as 2.4			5000	1000	
	Total	13427	2516	3727	34811	7647	
	Cummulative Total	18805	3327	4216			
5.	Trajectory Correction						
5.1	Navigation and Guidance						
5.1.1	Process Accelerometer Outputs	Same as 1.1.1			480	74	
5.1.2	Navigation Computation	Same as 1.1.2			976	58	

Table A-1. (Cont)

Requirements		Storage			Speed		Word Length
		Instr	Const	Var	Short	Long	Bits
5.1.3	Velocity to be Gained	Same as 4.1.3			30000	10000	
5.1.4	Velocity to be Gained Steering	Same as 3.1.4			3000	700	
	Subtotal	2310	77	92	34456	10832	
5.2	Telecommunications	Same as 4.2			5000	500	
5.3	Scientific Experiments	Same as 4.3			7791	927	
5.4	Status Monitoring	Same as 3.4			3000	350	
	Total	9481	1416	3446	50247	12609	
	Cummulative Total	18805	3327	4216			
6.	Spin Up						
6.1	Navigation and Guidance						
6.1.1	Angular Velocity to be Gained	500	40	50	5000	1500	
6.1.2	Steering	400	30	40	10000	4000	
	Subtotal	900	70	90	15000	5500	
6.2	Telecommunications	Same as 4.2			5000	500	
6.3	Scientific Experiments	Same as 4.3			7791	927	
6.4	Status Monitoring	Same as 3.4			3000	350	
	Total	8071	1359	3444	30791	7277	
	Cummulative Total	19705	3397	4306			
7.	Spin Cruise						
7.1	Navigation and Guidance						
7.1.1	Attitude Reference	Same as 2.1.1			11520	3320	
7.1.2	Navigation Computation	Same as 4.1.2			3000	1000	
7.1.3	Velocity to be Gained (Monitor)	Same as 4.1.3			2500	900	

Table A-1. (Cont)

Requirements		Storage			Speed		Word Length
		Instr	Const	Var	Short	Long	Bits
7.1.4	Angular Velocity to be Gained (Monitor)	Same as 6.1.1			1000	300	
	Subtotal	4756	517	343	18020	5520	
7.2	Telecommunications	Same as 4.2			5000	500	
7.3	Scientific Experiments	Same as 4.3			7791	927	
7.4	Status Monitoring	Same as 2.4			5000	1000	
	Total	13927	2556	3837	35811	7947	
	Cummulative Total	19705	3397	4306			
8.	De Spin	8071	1359	3444	30791	7277	
	(Same as 6. Spin Up)	19705	3397	4306			
9.	Mars Approach Correction						
9.1	Navigation and Guidance						
9.1.1	Process Accelerometer Outputs	Same as 1.1.1			480	74	
9.1.2	Navigation Computation	Same as 1.1.2			976	58	
9.1.3	Velocity to be Gained	4.1.3 plus 300	15	5	30000	10000	
9.1.4	Steering	Same as 3.1.4			3000	700	
	Subtotal	2610	92	97	34456	10832	
9.2	Telecommunications	Same as 4.2			5000	500	
9.3	Scientific Experiments	Same as 4.3			7791	927	
9.4	Status Monitoring	Same as 3.4			3000	350	
	Total	9781	1431	3451	50247	12609	
	Cummulative Total	20005	3412	4311			

Table A-1. (Cont)

Requirements		Storage			Speed		Word Length
		Instr	Const	Var	Short	Long	Bits
10.	Aerobraking						
10.1	Navigation and Guidance						
	Entry Constraint and Steering	3000	200	200	12000	3000	
10.2	Telecommunications	Same as 4.2			5000	500	
10.3	Scientific Experiments	Same as 4.3			7791	927	
10.4	Status Monitoring	Same as 3.4			3000	350	
	Total	10171	1539	3554	27791	4777	
	Cummulative Total	23005	3612	4511			
11.	Mars Orbit Injection						
11.1	Navigation and Guidance						
11.1.1	Process Accelerometer Outputs	Same as 1.1.1			480	74	
11.1.2	Navigation Computation	Same as 1.1.2			976	58	
11.1.3	Required Velocity	400	20	15	7000	2000	
11.1.4	Steering	Same as 3.1.4			3000	700	
	Subtotal	1210	72	47	11456	2832	
11.2	Telecommunications	Same as 4.2			5000	500	
11.3	Scientific Experiments	Same as 4.3			7791	927	
11.4	Status Monitoring	Same as 3.4			3000	350	
	Total	8381	1411	3401	27247	4609	
	Cummulative Total	23405	3632	4526			
12.	Mars Orbital Coast						
12.1	Navigation and Guidance	Same as 2.1			22800	4516	30
12.2	Telecommunications	4.2 plus 1500	200	1000	10000	1000	

Table A-1. (Cont)

Requirements		Storage			Speed		Word Length
		Instr	Const	Var	Short	Long	Bits
12.3	Scientific Experiments						
12.3.1	Data Compression	4.3 plus 500	445	1546	194840	36120	12
12.3.2	Sequencing	200	50	50	500	5	16
12.3.3	Pointing	1000	50	50	10000	4000	16
	Subtotal	4071	1134	3830	205340	40125	
12.4	Status Monitoring	Same as 2.4			5000	1000	
	Total	16695	3625	6600	243140	46641	
	Cummulative Total	26605	4377	7172			
13.	Trans Earth Injection						
13.1	Navigation and Guidance						
13.1.1	Process Accelerometer Outputs	Same as 1.1.1			480	74	
13.1.2	Navigation Computation	Same as 1.1.2			976	58	
13.1.3	Required Velocity	700	30	25	10000	3000	
13.1.4	Steering	300	15	15	4000	1000	
	Subtotal	1610	87	52	15456	4132	
13.2	Telecommunications	Same as 4.2			5000	500	
13.3	Scientific Experiments	Same as 4.3			7791	927	
13.4	Status Monitoring	Same as 3.4			3000	350	
	Total	8781	1426	3406	31247	5909	
	Cummulative Total	27605	4422	7212			
14.	Trans Earth Coast						
	Same as 4.						
15.	Trajectory Correction						
	(Same as 5.)						

Table A-1. (Cont)

Requirements		Storage			Speed		Word Length
		Instr	Const	Var	Short	Long	Bits
16.	Spin Up						
	(Same as 6.)						
17.	Spin Cruise						
	(Same as 7.)						
18.	De Spin						
	(Same as 8.)						
19.	Earth Approach Correction						
19.1	Process Accelerometer Outputs	Same as 1.1.1			480	74	
19.1.2	Navigation Computation	Same as 1.1.2			976	58	
19.1.3	Velocity to be Gained	9.1.3 plus 600	25	20	45000	15000	
19.1.4	Steering	Same as 3.1.4			3000	700	
	Subtotal	3210	117	117	49456	15832	
19.2	Telecommunications	Same as 1.2			1000	200	
19.4	Status Monitoring	Same as 3.4			3000	350	
	Total	6010	767	287	53456	16382	
	Cummulative Total	28205	4447	7232			
20.	Earth Re-entry						
20.1	Re-entry Energy Management and Guidance	10.1 plus 2300	250	250	18300	4460	
	Subtotal	5300	450	450			
20.2	Telecommunications	Same as 1.2			1000	200	
20.4	Status Monitoring	Same as 3.4			3000	350	
	Total	8100	1100	620	22300	5010	
	Cummulative Total	30505	4697	7482			

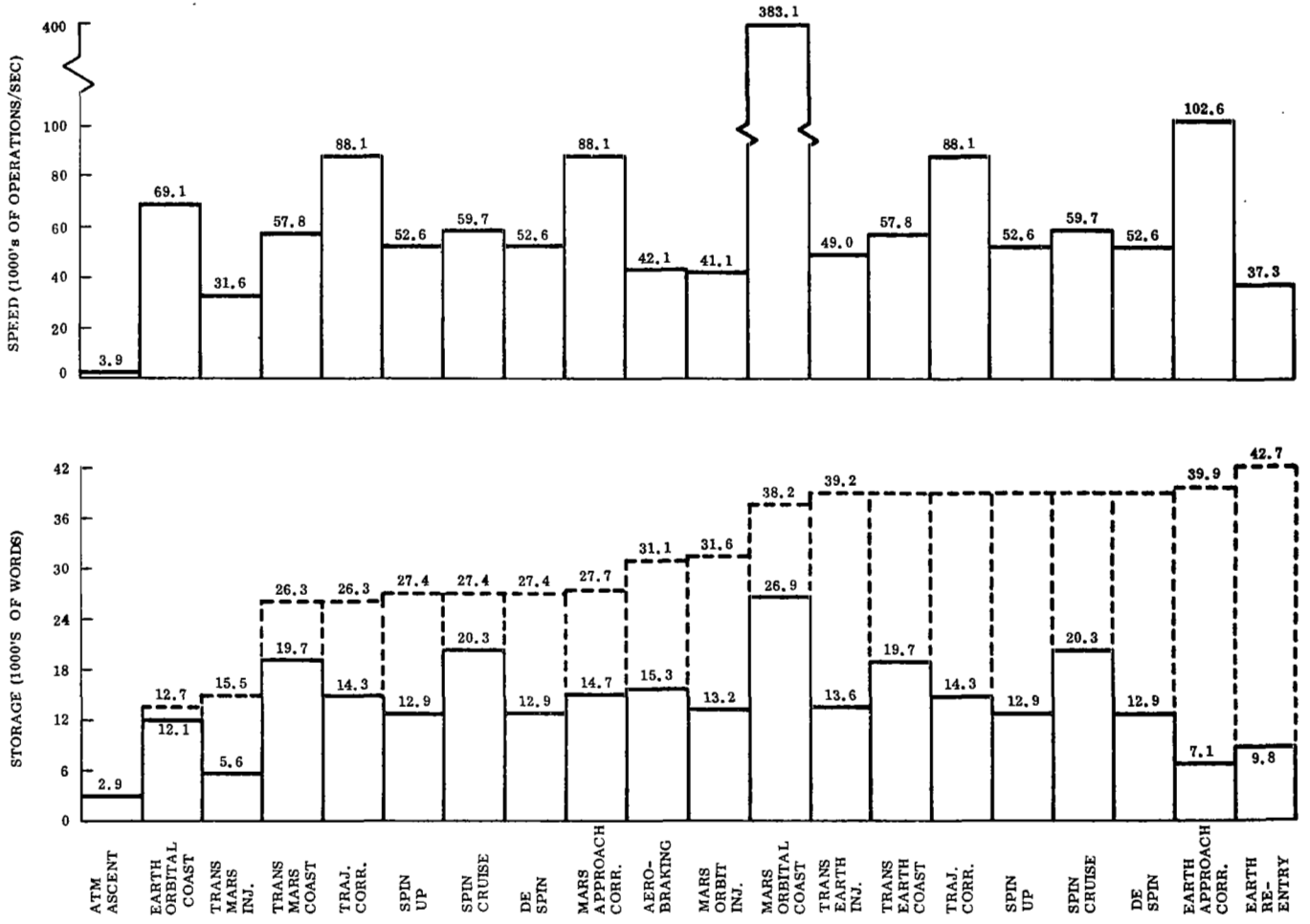


Figure A-1. Computer Requirements per Phase

APPENDIX B. MASS STORAGE CONSIDERATIONS

B1 INTRODUCTION

There exists the need for providing a mass storage medium in the computer system. This is dictated by the fact that it is desired to store the mission programs in some device assuring the integrity of approximately 10^6 bits. In addition, it is desired to provide a data buffer between sensors and the computer. This is used to store data in bulk quantity prior to processing either due to the desirability for burst processing on accumulated sensor data such as video scans and/or buffer bursts or hi-rates from sensors. The total storage required is expected to be on the order of 10^8 bits for data and mission programs with the majority of this for sensor data. The discussion in this section will present technology trends that may be applicable to the mass storage medium required for the manned Mars mission.

Much of the difficulty of selecting an optimum approach to a 10^8 bit memory system arised because no non-mechanical systems that large have been constructed. The general undesirability of massive moving elements in a spaceborne system, the maintenance requirements, and the dependence on a mechanical determination of data rate are factors which have eliminated electro-mechanical systems such as tape and drum storage from consideration.

Several groups within the industry have been funded under government contracts for the development of 10^8 bit memory systems. While none of the systems have yet been completed, the development of portions of such systems have been carried to the point where full-system problems are illuminated and reasonable predictions of system characteristics can be made.

The three outstanding approaches to mass storage include assemblies of magnetic plated wire arrays, stacks of planes of toroidal cores etched from thin permalloy sheets, and an assembly of long flat strips of glass coated with permalloy and with copper.

All of these approaches to mass memory utilize a random-access word organized configuration. The application herein considered could use a different type of organization in which random access is provided to blocks of information with serial access to the bits constituting the block (BORAM). Non-mechanical versions of this type of memory are currently under development; however, their progress and application to mass memory is several years behind the previously mentioned systems.

Some details of these various approaches are described in the following paragraphs.

B1.1 Plated Wire Memory

The fundamental operation of a plated wire memory cell was described in sections 3 and 6. A recent study contract, funded by the Rome Air Development Center, resulted in a proposed 10^8 bit memory design (characteristics of the system have been simulated) with the following characteristics: (Reference 26)

The system will be organized as ten 10^7 bit modules. Each ten-million-bit memory plane constituting a module contains 2048 word-line solenoids encircling 4608 plated wires (plus a small number of common mode cancelling wires). The modules

will be organized with 64 words of 72 bits on each word line, and each time all the bits in such a word group line are interrogated, only the bits belonging to the selected word are routed by a set of gates to the sense amplifiers. This mode of operation is possible because of the non-destructive characteristics of the interrogation, thus making restoration of information in interrogated bits unnecessary. This property is very important because it allows a memory configuration to be chosen which leads to a minimal number of bit and word drivers and sense amplifiers. (The organization could easily be changed to 256 words of 18 bits or other combinations whose product totals 4608 bits.)

The design word rate for reading and writing will be 100 kilocycles per second which corresponds to a serial bit rate of 7.2 megacycles per second.

The output signals, appearing on the bit lines are of the order of 5 millivolts in amplitude and 70 nanoseconds wide.

The word lines will be spaced at 0.045 inch centers and the bit lines spaced at 0.015 inch centers, resulting in a storage density of approximately 1500 bits per square inch.

Ultimately all of the circuits in the system are expected to be of the microelectronic type. Near future designs use a combination of integrated circuits and cordwood packages using discrete components.

Interconnection will be made through multilayered boards with soldering and wire-wrap technique.

The size of the 10^8 bit memory will be 4 x 5 x 1.5 feet, or 30 cubic feet in volume. The total weight will be approximately 750 pounds. It is predicted that future efforts will reduce the volume to 20 cubic feet.

The power consumption decreases with lower speed operation, and with an increasing ratio of read to write cycles. At the 7.2 megacycles per second bit rate the power consumption is estimated to be 72.5 watts, but decreases to 30 watts at a 0.72 megacycle per second bit rate. The latter rate is entirely acceptable for this application. Increasing the number of read cycles per write cycle would effect up to a 14% reduction in the power consumed.

A reliability computation considers the failures arising only from the joints in the memory stack and from the bit and word access matrices. Assuming a transistor failure rate of 10^{-9} per hour and a failure rate of joints of 10^{-10} per hour, the estimated mean time between failures is 7600 hours for a 10^8 bit memory. According to Autonetics experience with transistor reliability, 10^{-9} per hour failure rate is optimistic. A more realistic figure of 10^{-8} reduces the MTBF to 3300 hours. If one assumes a flat pack reliability of 10^{-9}, the MTBF rises to 22,000 hours. Manufacturing cost per bit for the 10^8 bit system is projected to be 0.108 cents per bit in 1970.

B1.2 Etched Permalloy Toroid Memory

Rome Air Development Center has sponsored the development of a mass memory technique which utilizes toroids etched from sheets of permalloy as the storage elements which are batch fabricated, along with the associated conductors, into 256 x 256 bit memory planes. (Reference 27)

A coincident-current memory organization is used. The writing system is conventional requiring the coincidence of pulsed fields on the X and Y lines with the writing of a zero or one determined by the presence or absence of the inhibit line current in each plane. The reading operation uses a two-frequency selection scheme which is nondestructive. In the reading operation only a toroid at the intersection of the selected row and column drive line is energized by fields of both frequencies W_1 and W_2. The core acts as a non-linear mixing element to produce a sum frequency component on the sense line. The signal is amplified, narrow-band filtered, and phase detected against a reference which yields a signal whose output polarity depends on the state of the core. The read drive frequencies of 570 kilohertz for W_1 and 930 kilohertz for W_2 result in a sum frequency of 1.5 megahertz. The result is a read time of 10 μsec with a nominal signal amplitude of 50 μv rms.

The batch fabricated memory planes consist of flat toroids etched from sheet permalloy with the associated wiring formed by etching and plating copper. The topology is such that wires never cross on the same plane, thus allowing the wiring pattern to be formed by two layers of etched copper insulated from each other and the permalloy toroids, but connected by means of plated regions through the interior of the toroids. Using highly developed photoetching procedures the toroids are fabricated on 25 mil centers which results in 1600 bit per sq. inch density within the plane.

The 10^8 bit system requires 1616 planes of 256 x 256 bits per plane. They will be assembled into 16 block with 101 plane per block. The word length is 100 bits.

The design package has a volume of 5 cubic feet corresponding to a weight of 468 pounds. Power figures range from 98 watts for a read-only mode to a maximum of 179 watts for write-only, at a maximum write-in word rate of one word every 23 microseconds.

Trade-offs among the parameters can reduce memory size and power consumption. If the word length is extended to 200 bits, for example, and the write-in rate decreased to one word every 80 microseconds, the maximum power would decrease to 110 watts, the size to 4 cubic feet and the weight to 374 pounds.

The present status of this development is that of attempting to fabricate the 256 x 256 bit planes. Hence, no system reliability data is available. Therefore an attempt to estimate a MTBF figure by considering only the plane edge interconnections and the number of semiconductors will be made. The interconnections will be 1,600,000 for the 10^8 bit memory (versus 180,000 for the plated wire memory) and the number of integrated circuit packages plus discrete semiconductors total 4481. Assuming 10^{-9} per hour failure rate for interconnections, the interconnection are the dominant factor, so that changing the failure rate for semiconductors from 10^{-8} to 10^{-9} only changes the calculated MTBF from 4900 to 6000 hours.

The small signal amplitudes and the nature of the read-out system requires a consideration of effect on reliability of coherent noise in the sense lines and random noise in the sense amplifier. This is calculated to correspond to an erroneous reading of one bit every 10,000 hours, which may be detected and corrected by repetition of the interrogation.

A cost figure of 0.083 cents for electronics plus 0.056 cents for memory planes is estimated for a total cost of 0.139 cents per bit.

B1.3 Flat Film Strip Memory

Lincoln Laboratories is engaged in the development of a mass memory using flat magentic films at very high storage density.

The storage elements are formed of rectangular glass strips coated first with permalloy and then copper. These strips are etched into 2 mil lines on 4 mil centers terminating in a pattern of lands which provide fan-out to a pressure connector. Twenty-four of these substrates, each 2 ft by 1 inch, are placed side by side forming a square plane and another similar plane is placed above it with its strips lying perpendicular to the first set, with an insulating layer between. A square array results with 6K bits on a side for a total of 36 x 10^6 bits.

Current work on the project is centered about the assembly of a one million bit model using 10 inch substrates. Recent experiments center about a bit density of 250 words per inch and 50 digits per inch, i.e., 12,500 bits per square inch. (Reference 28) Cost projections are in the order of 0.3 cents per bit.

While this approach achieves extremely high density for the memory stack, it also has several unfavorable characteristics. High coercive force films are used to reduce the effects of demagnetizing fields at the high bit density, which require currents in the 100 to 500 ma. range. Read-out is destructive and, therefore, all bits of a word must be rewritten each time after reading, which results in excessive power consumption in this application when the ratio of read to write cycles exceeds one. The destructive readout characteristic precludes the sharing of bit line circuits so that the linear select organization requires many more circuits than the plated wire or etched toroid approaches. Since reliability of the system is so strongly dependent on the electronics, this approach to the mass memory requirement does not appear favorable.

B1.4 BORAM

A type of mass memory that seems to be well adapted to this requirement is that being developed under the BORAM concept. BORAM, an acronym from Block Oriented Random Access Memory, has been promulgated by the U.S. Army Electronics Command at Fort Monmouth, New Jersey.

The basic idea of BORAM is that the mechanization be an all-electronic or static type memory with rapid random access to the blocks of data (1 microsecond access time) with the subsequent sequential transfer of a block on a character-by-character basis. A 1.5 to 3 million character per second transfer rate is envisioned. Other requirements in the total concept include removable storage media and asynchronous transfer capability.

From the viewpoint of the mass storage requirement of the study, the memory would appear as a group of serial storage devices with random access to each device and serial access to the bits within the selected device. The important advantage is the large factor of minimization of electronics and interconnections and the consequent gain in reliability.

A specific implementation of the mass memory under this concept cannot be delineated at the present time because applicable devices are in the early stages of development. Among these developments the most interesting ones include a ferroacoustic delay line, a thin film storage strip which responds to the field from a

propagating domain wall in an adjacent strip, and a new technique which utilizes the controlled propagation and interaction of domain tips through a pattern of magnetic film channels (Reference 29).

It is reasonable to assume operating goals for these devices, with respect to speed and density, of 10,000 bits per square inch and 1 megacycle serial data transfer rate. Then 20 of these devices each storing 5×10^6 bits could provide the required 10^8 bit storage. Since random access to any of the twenty blocks could be attained very rapidly, the maximum access time would be approximately five seconds. Since the total circuitry would be reduced to the selection circuits for the twenty blocks, driver circuits (probably a maximum of four per block) and a read amplifier per block, system failure due to the electronics could be made very small.

The aspect of this approach that is unfavorable is the uncertainty of the developmental time scale which will allow mechanization of the system.

B1.5 Reliable Backup Mass Storage

The backup mass storage is used to store the guidance and control functions that would be necessary to complete the mission if the primary mass storage was to fail. This memory should, of course, be very reliable and in fact, it is probably reasonable to carry a spare due to its relatively small size, approximately 2×10^5 bits. Although extensive investigation of the mass memory was not planned for this study, it appears that the backup storage could be read only. This, of course, would provide obvious reliability advantages due to the lesser numbers of circuits.

A number of possibilities exist for this memory. One is a fixed memory, such as the "Silicon on Sapphire Diode Array". A second is a memory of the same technology as the primary mass storage only without write circuits. The best solution will depend on the relative reliabilities and power and on the ability of the second approach to use spares from the primary mass storage.

B1.6 Conclusion

The above discussion demonstrates that there are a number of memories on the order of 10^8 bits either under development or being considered for development; however, the near term systems are large and also dissipate a good quantity of power. If any of these approaches are to be made applicable to this study, both the power dissipation and size will need to be substantially reduced. Future developments along the lines of "BORAM" may offer the best system for space.

APPENDIX C. FAULT AND ERROR CONTROL

Faults can lead to the generation of errors and can cause down time. It is therefore necessary to consider the risks associated with the possible occurrence of faults as well as methods of remedying or at least reducing their harmful effects. The harmful effects of errors can be reduced by error correction or by special treatment of erroneous results. Down time as a result of faults can be reduced by reconfiguring the system to evade the effects of known faults and by diagnosing faults so that repair is facilitated.

Techniques for overcoming these harmful effects may require an enlargement of the system. Two questions regarding this enlargment of the system arise. First, the enlarged system contains more elements which can become faulty and therefore is more likely to contain a fault. Hence, it is natural to ask whether or not there will indeed be a net enhancement in reliability. This question will be treated in section C2.1.

The second question involves the additional costs, the additional weight and the other penalties associated with the enlargement of the system. Will improvements in the system be worth the price? To answer this question rationally, it is necessary to optimize the relationship between the penalties, the probability of mission success and the expected productivity of the mission.

Three processes used in overcoming these harmful effects will be consdiered—error or fault detection, error treatment, and system reconfiguration. The techniques used will usually involve more than one of these processes. Also, it is sometimes difficult to determine where one leaves off and the other begins. However, it is useful to consider the characteristics of these processes separately. Error or fault detection involves the determination that an error or fault is or is not present and also the initiation of an alarm or corrective procedure when appropriate. Error or fault detection may be performed for every step of an operation, it may be performed after the completion of a set of operations, or it may be performed periodically. When error detection is performed for every step of an operation, redundant computing elements are usually introduced into the system. When it is performed upon the completion of a set of operations, redundant computations are performed. When fault detection is performed periodically, a self-test program is used. This self-test program will not detect faults causing intermittent errors unless they occur during the self-test program.

Treatment of errors may involve error correction by redundant logic, error correction by roll-back, or special handling of erroneous results. Error correction does not necessarily involve error detection. Therefore, it may be necessary to include in check-out procedures some provision for the detection of faults whose errors are corrected. This could be done by injecting errors into the system to determine whether or not they are corrected or by disabling redundant features during check-out.

Roll-back procedures are simplest when they involve transient errors. In this case, a recalculation performed in exactly the same manner as the original computation will produce the correct result when the transient error does not recur. For reproducible errors, the roll-back procedure would be more complicated. The recalculation then must use different logic paths to evade the fault causing the error or to correct the error. This will be treated in further detail in section C. 2.2.

System reconfiguration involves the removal of a fault from the active system or the introduction into the system of a means of correcting the errors due to the fault. The most simple techniques involve the removal of the faulty element from the active system. However, it is possible to resequence the operations of the system so that the faulty element is not used in a manner that will cause an uncorrected error.

Faults of the following types will be referred to as "standard faults" hereafter in this report

1. An open input to a gate
2. The inability of a gate to drive to zero the node to which its output is connected.
3. The inability of a node to go to one.

C 1. TECHNIQUES FOR FAULT OF ERROR DETECTION

This section will treat methods of detecting errors in data processing operations and methods of detecting faults thru the errors they cause.

Most of the error detection techniques treated in this section involve the use of redundant hardware for checking each step of a computation as it is performed. However, some consideration will be given to error detection techniques in which some of the computations in a program are used to check the results of other computations.

Most of the fault detection techniques used involve the use of self-test or diagnostic programs. These programs are used at various times to verify that there are no faults in the system or to help locate faults which may be present in the system. Usually self-test or diagnostic programs in a general purpose computer do not involve special circuits other than circuits for responding to the detection of a fault, although provisions for disabling self-correcting features or alternately injecting errors may be required for checking out redundant circuits. Other fault detection techniques utilize special hardware to detect specific faults, for example, a circuit which will generate an alarm if clock pulses do not occur regularly.

The error patterns which are to be detected by specific fault or error detection techniques are established by one of two approaches. In the first hardware oriented approach, an effort is made to consider the faults or combinations of faults which may occur and, from these, to determine a set of possible error patterns. This method has the advantage of making it possible in principle to assign a failure rate to each error pattern. Hence, if any combination of faults is considered to be so improbable that the associated error pattern can be neglected, then a failure rate can be assigned to that neglected pattern. From this, a summation can be made to obtain the combined failure rate of all known neglected error patterns. However, to obtain the failure rates for unknown faults, experimental techniques are required.

In the second functional oriented approach, the set of error patterns to be detected is established without considering the logical details of the systems or the faults which may occur. This method makes it possible to design the error detection procedure prior to the completion of the logical design. If intelligently used, it could detect most of the reasonably probable error patterns and some others besides. However, for fault detection, this approach is inefficient. To be certain that all

possible faults have had opportunities to generate errors, it is necessary either to examine the logic structure or to check many functional situations to be confident that all logic paths have been used.

Fault detection and error detection techniques have different principal objectives. Fault detection equipment is designed primarily to prevent a fault from causing future errors and to initiate its removal from the active system. It may also invalidate erroneous results before they are used. Error detection equipment is designed primarily to initiate the correction of an error or special handling of erroneous results. However, an alarm from an fault detection system may also perform some or all of the principal functions of an alarm from an error detection system and vice versa.

With these general concepts in mind, specific fault detection and error detection techniques will be considered.

C1.1 Continuous Error Detection or Fault Detection

C1.1.1 Data Transfers

Self-checking codes can be used to verify the accuracy of a data transfer. The theory of self-checking codes has been intensively developed. Therefore further comments of a general nature on this topic are not in orcer.

C1.1.2 Memory

Self-checking codes can be used to verify the accuracy of transfers between memory and the other modules.

Self-checking codes can also be used to verify the address. Extra bits can be hard-wired into each memory location. These bits could be concatenated with the address to form a self-checking code. For these extra bits, sense amplifiers would be required, but inhibit drivers would not be required.

Decoding circuits can be checked by feedback and comparison with the original code or address to verify that the proper signal alone is generated.

The memory cells of a DRO memory could be checked out before information is stored into the memory location. This could be accomplished by adding an extra read operation at the beginning of a write cycle. This would add a half cycle to every write cycle, but would not affect the read cycle. The sequence of operations for writing with the memory cells checked before writing are as follows:

1. Insert ones in all bits of the memory data register
2. Write
3. Insert zeros in all bits of the memory data register.
4. Read (thus inserting ones in the memory data register bits corresponding to ones in memory cells of the addressed location)
5. Test the memory data register. If any bit contains a zero go to an error routine.

6. If all bits of the memory data register contain ones, transfer the word to be stored to the memory data register.

7. Write

This fault-checking routine will protect the stored data from faults in the memory cells themselves. However, it will not provide complete protection from faults in the write circuitry. Reading back the stored word would provide this protection, especially in NDRO memories.

C1.1.3 Adder

Various methods are available for checking the results of an addition. Examples of specific technqiues are given below.

C1.1.3.1 Residue Checks

Residue checks are performed by adding in mod n arithmetic the addend and the augend used in an addition. The result should be congruent (mod n) to the sum. If the sum is in error by an amount which is divisible by n, the residue check will not detect the error.

In many adders, most faults will cause errors which are a power of two. Thus, if $n = 2^k-1$, most errors will be detected.

If the adder operates in mod 2^p arithmetic, overflow is equivalent to a subtraction of 2^p from the answer. Therefore some provision must be made for overflow in designing a residue check unless the modulus of the arithmetic is divisible by the modulus of the residue check. If the adder operated in 2^r-1 arithmetic, the modulus of the adder will be divisible by the modulus of the residue check if r is a multiple of k.

C1.1.3.2 Duplication Check

Addition can be checked by duplication of the adder. The result obtained from the two adders is compared to see if they are the same. A check of this nature will catch any faults which may occur in one adder provided that the other adder makes no errors. This technique is especially advantageous if reconfiguration is provided for. If the self-checking doctrine is abandoned, one of the adders could carry on if the other failed.

C1.1.3.3 Complementation Checks

In mod 2^n arithmetic there are three inputs: the addend, the augend and the input carry to the units bit. Now, if

$$s = A + B + Co$$

$$(2^n - 1) - s = ((2^n - 1) - A) + ((2^n - 1) - B) + (1 - C) + 2^n$$

or

$$((2^n - 1) - s) = ((2^n - 1) - A) + ((2^n - 1) - B) + (1 - C) \qquad (\text{mod } 2^n)$$

In otherwords, if the ones complement of the input operands to a mod 2^n addition is formed, then the ones complement of the sum is formed. Similarly, in mod 2^n-1 addition

$$s = A + B$$

$$s' = A' + B' \qquad (\text{mod } 2^n-1)$$

One can therefore check an addition by calculating the sum, and then calculating the sum of the complements and finally seeing if the results are ones complements.

It should be noted (See Figure C-1) that it is possible to construct an adder so that the signals at all the nodes of the adder are complemented if all the inputs of the adder are complemented. If an adder is thus constructed, a "standard" fault will cause an error in one but not both of either the sum or the sum of the complements. Thus a complementation check will detect all standard errors in an adder of this structure.

C1.1.3.4 Self-Checking Adder

Figure C-2 shows one bit of an adder with internal self-checks. An error caused by a single standard fault will be detected.

If an input to any decoding gate is open, for some values of the operands there will be two sum-carry combinations indicated. If these two combinations have the same carry, both s and $\overline{s}$ will be indicated. This will cause the gate connected to s and $\overline{s}$ to go low and indicate an error. When the inputs are complemented, this fault will have no effect so that the correct answer will be obtained and no error will be indicated. It can then be stated that the fault has been evaded by complementation.

If the two combinations of sum and carry simultaneously indicated have the same sum but different carries, all the sum-carry combinations of the next stage will be inhibited. Hence all four sum-carry nodes will be high and the "no output" error for the next node will be indicated.

If standard faults of type b and c appear, the errors associated therewith will cause two sum-carry nodes to be zeros or all sum-carry nodes to be ones.

If the Acc. Bit does not go into the correct state, a fault will be indicated because the state of the flip-flop does not match the s or $\overline{s}$ signal. If neither s or $\overline{s}$ is high a similar alarm will appear.

Faults inhibiting the detection of an alarm will not cause an error unless some other fault is present. However, these faults may stifle alarms. To avoid degradation of the system leading ultimately to undetected errors, it would be desirable to be able to verify that there are no faults inhibiting alarms. The gate for causing one of the nodes to equal zero incorrectly can be used in checkout to demonstrate that having two gates equal to zero can be detected.

Similarly, it is possible to verify that there is no fault preventing the detection of a condition in which all sum-carry nodes equal one. This is accomplished by a fifth input on one of the adder gates to make that gate have an incorrect one output. This is required for the units bit only.

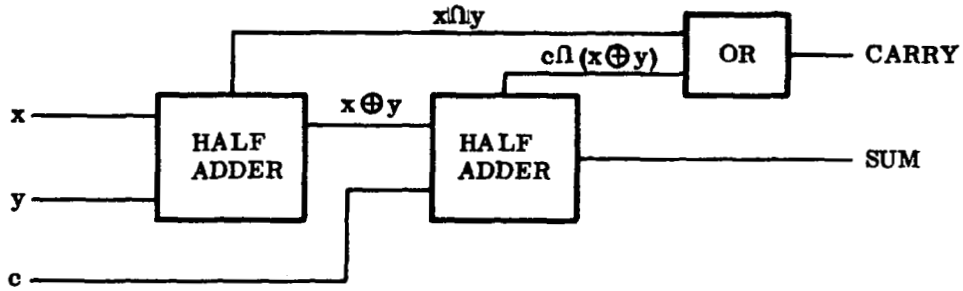


FIG. C-1A . ADDER WHOSE FAULTS CANNOT BE DETECTED BY COMPLEMENTATION, THE SIGNALS x∩y, x⊕y AND c∩(x⊕y) ARE NOT ALWAYS COMPLEMENTED WHEN x, y, AND c are COMPLEMENTED

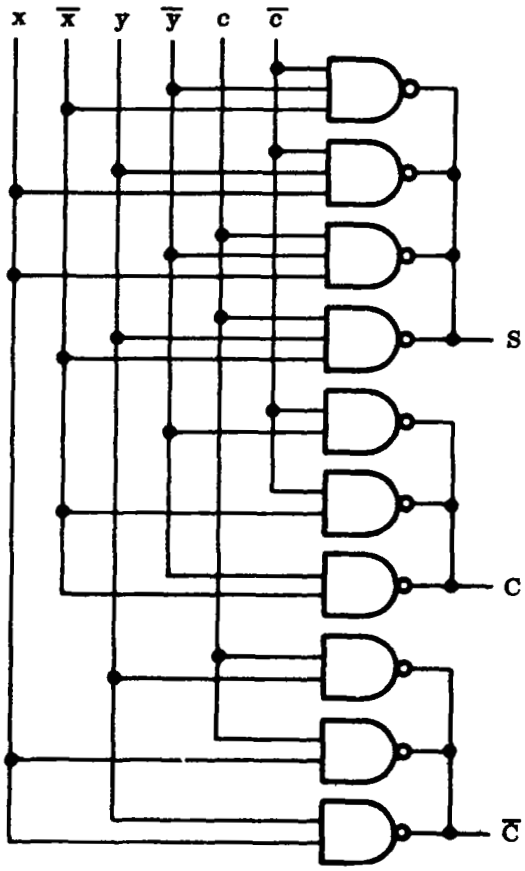


FIG. C-1B ADDER IN WHICH STANDARD FAULTS CAN BE DETECTED BY COMPLEMENTATION. THE SIGNALS AT ALL NODES ARE COMPLEMENTED WHEN THE INPUTS ARE COMPLEMENTED.

Figure C-1. Detection of Standard Faults by Complementation

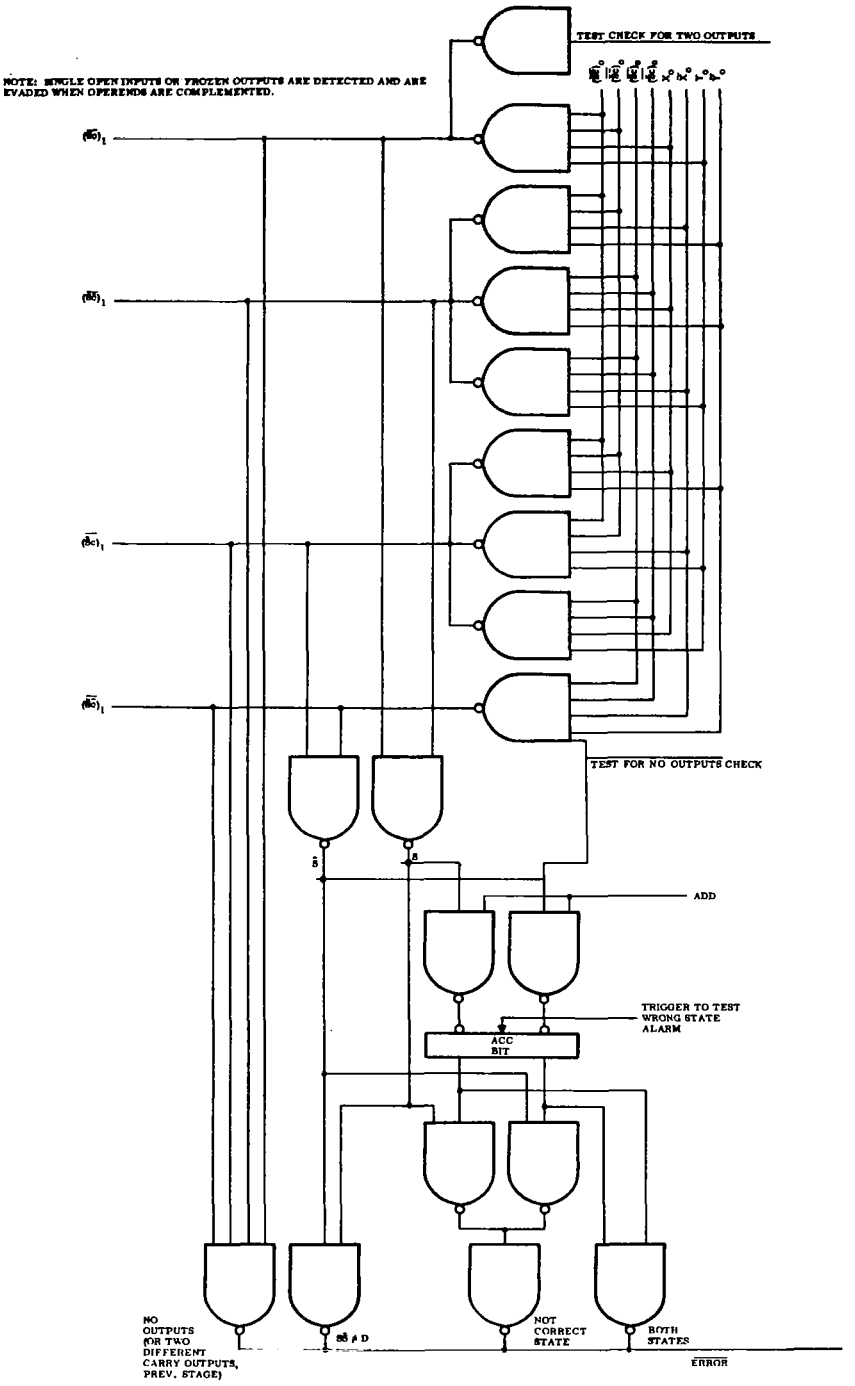


Figure C-2. Self-Checking Adder Without High-Speed Carry

An additional test device to force the Accumulator bit into the wrong state verifies that the "wrong state" alarm is not inhibited.

Faults freezing the error node into the zero state cannot be evated by complementation. Note that the signal on this node is not complemented when all the inputs to the adder are complemented.

C1.1.3.5 Parity Check

Figures C-3 and C-4 show a technique for performing a parity check on an addition. It should be noted that the ith bit of the sum is given by the formula

$$S_i = A_i \oplus B_i \oplus C_i$$

The parities of the augend, addend, and carries are

$$P(A) = A_0 \oplus A_1 \oplus \ldots \oplus A_n$$

$$P(B) = B_0 \oplus B_1 \oplus \ldots \oplus B_n$$

$$P(C) = C_0 \oplus C_1 \oplus \ldots \oplus C_n$$

Also, the parity of the sum is

$$P(S) = S_0 \oplus S_1 \oplus \ldots \oplus S_n$$

$$= (A_0 \oplus B_0 \oplus C_0) \oplus (A_1 \oplus B_1 \oplus C_1) \oplus \ldots \oplus (A_n \oplus B_n \oplus C_n)$$

$$= (A_0 \oplus A_1 \oplus \ldots \oplus A_n) \oplus (B_0 \oplus B_1 \oplus \ldots \oplus B_n) \oplus (C_0 \oplus C_1 \oplus \ldots \oplus C_n)$$

$$P(S) = P(A) \oplus P(B) \oplus P(C)$$

Now, when parity bits are used for checking data transfers, P(A) and P(B) are carried along with A and B. Therefore P(S) can be calculated from P(A), P(B) and the carries into the adder bits as shown in Figure C-4. Also shown in Figure C-4 is the parity checker used to verify that the addition is correct. The parity check is not sufficient to guarantee that the addition is correct. The carries may have been incorrect. Since the parity check uses the carries to compute the required parity of the sum, an error in a carry would cause not only an error in the sum but also an equivalent error in the sum parity bit. Thus a further check on the carries. In Figure C-3 the carries are checked by generating both carry and not carry for each bit. Then the results are compared by exclusive or networks.

Note that in Figure C-5 either the parity checker or the parity generator has an even number of bits. Thus one of these must have at least one node which is not complemented when the inputs are complemented. In the example shown in the figure, there are an even number of data bits. Therefore the parity bit for the sum is the same as the parity bit for the complement of the sum.

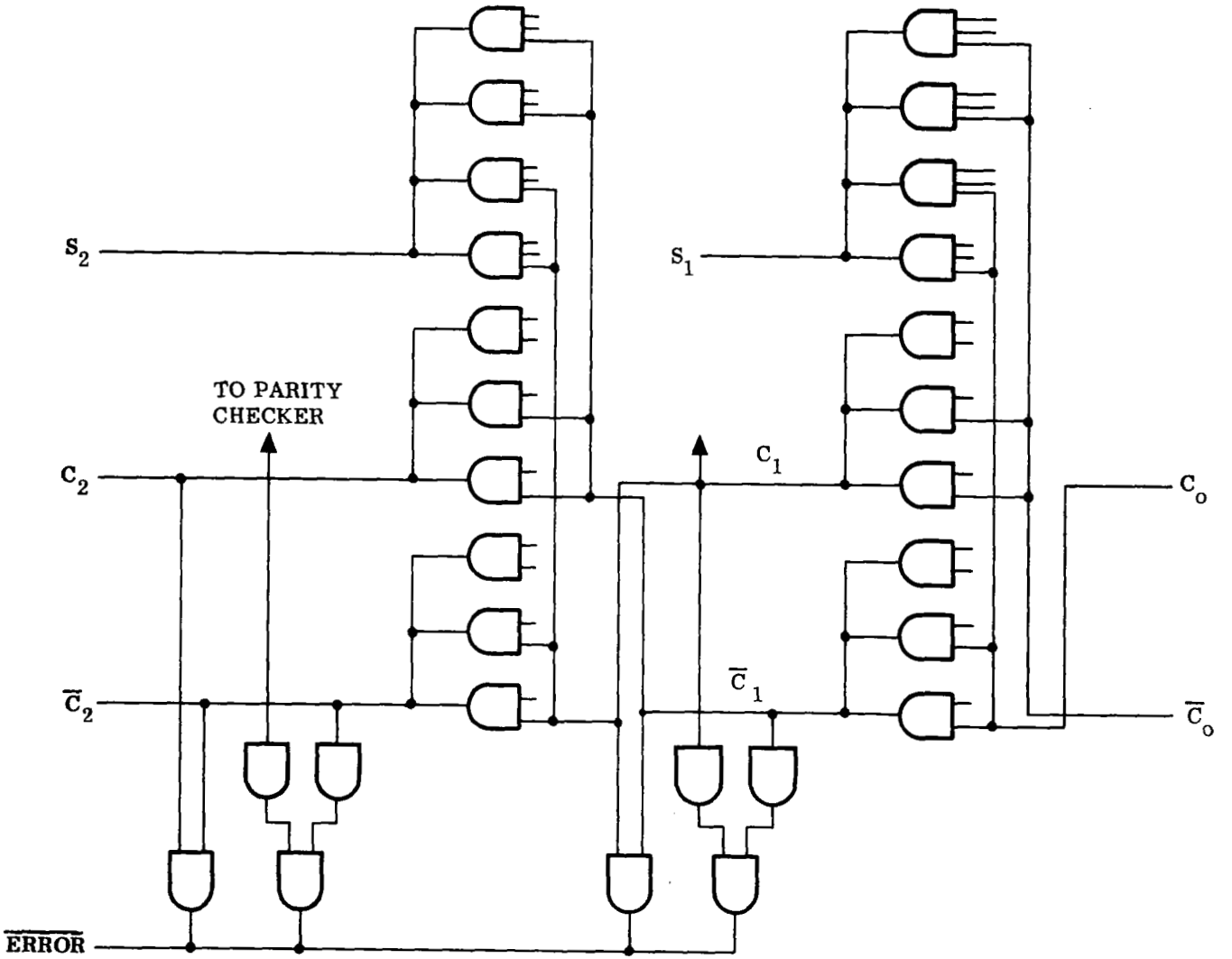


Figure C-3. Two Bits of Adder Used in Parity Checking Addition

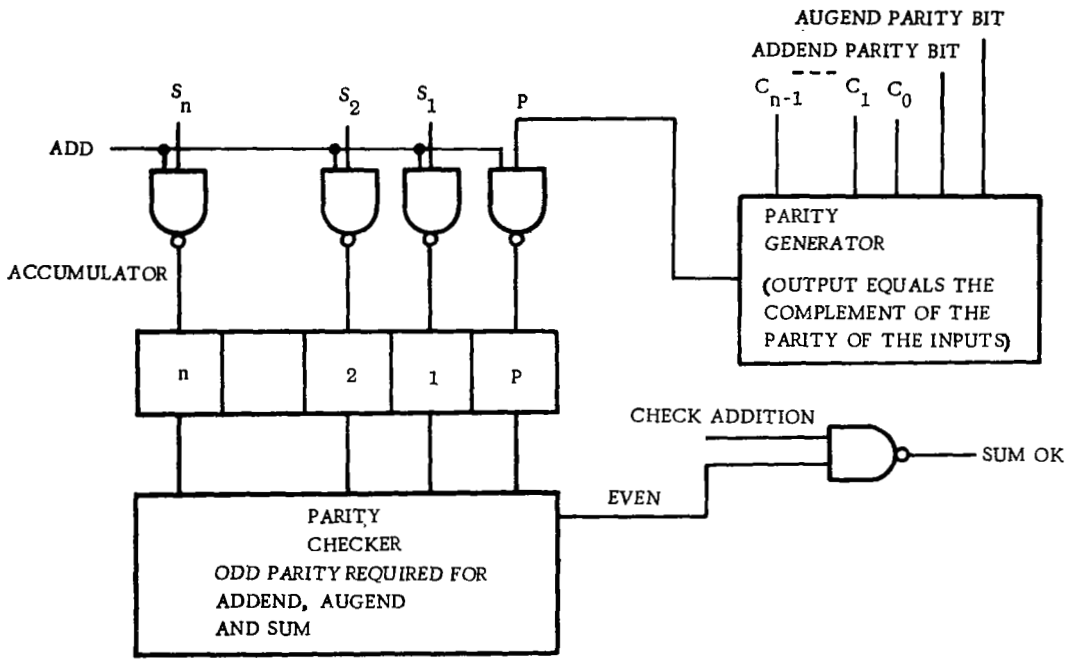


Figure C-4. Parity Checker Used in Parity Checking Addition

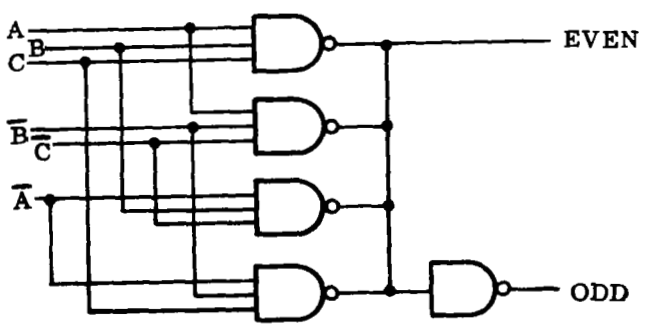


Figure C-5. 3-Bit Parity Checker

C1.2 Periodic Fault Detection

Fault detection is performed by self-test programs executed at regular intervals. The purpose of these programs is to detect the presence of faults so that they can be removed from the active system. Fault detection will not determine whether or not a discovered fault had caused an error. However, it can verify that the computations made up the last time the computer was found to be fault-free are free from reproducible errors. On the other hand, it is doubtful that fault detection techniques will check for marginal conditions so thoroughly that protection from transient errors could be guaranteed.

The self-test program carried to its ultimate conclusion will check each logic path thru the equipment so that if any fault could effect the signals thru any logic path, an error will be detected.

Practical limitations may interfere with the achievement of this objective. For example, the errors due to some faults do not always have reproducible effects. For example, a spurious signal might attempt to set a flip-flop while a correct signal attempts to reset the flip-flop. If there are no loading conditions or permanent circuit parameters permanently biasing the response of the flip-flop to the racing condition, the effect of the spurious signal on the state of the flip-flop is not reproducible.

The self-test program may attempt to detect faults causing non-reproducible effects. This might be done by exercising these marginal logic paths under a variety of conditions or at least exercising them repeatedly in the hope that conditions which could cause an error would arise accidentally.

A limitation, which also applies to error detection and error correction, involves the difficulty of listing all possible combinations of faults and determining their effects. First, there is the theoretical difficulty of conceiving all the possible failure modes. Second, there is the effort involved in determining the effects of each of an overwhelming number of possible fault patterns. Thus, it is necessary to limit the number of fault patterns investigated. A good approach is to limit the investigation to single faults plus possibly a few multiple fault patterns caused by the same nucleus of failure.

By arranging the self-test program so that the logic paths involved in each test involve many previously checked logic paths and only a few new logic paths, the effect of multiple failures upon the execution of the test program is reduced.

Performing the self-test program frequently, reduces the probability that two or more statistically independent faults will develop between two check-out operations.

It should be noted that even through uninvestigated failure patterns may occur, there is a good chance that they may be detected even though reliable diagnostic information for locating this exotic fault pattern is not available.

C1.3 Programmed Error Checks

It is possible for the programmer to verify the accuracy of many of his results by adding extra instructions to his program.

An example of how results can be checked can be drawn from the matrix multiplication

$$\sum_{j=1}^{h} A_{i_j} b_j = C_i \qquad (i = 1, 2, \ldots, m)$$

The programmer can add instructions for computing

$$\sum_{i=1}^{m} A_{i_j} = A_i \qquad (j = 1, 2, \ldots, h)$$

$$\sum_{j=1}^{h} A_j b_j = C$$

and determining whether or not

$$\sum_{i=1}^{m} C_i = C$$

Other checks can be designed for other types of computations.

These error checks provide good protection from faults. However, there is always the possibility that some fault might cause an error in the results and an equivalent error in the check. To design the check calculation to avoid this risk would require an extensive logical analysis of the error patterns due to possible faults. Through extensive software development, a procedure for automating this analysis might be developed.

C.2 TECHNIQUES FOR TREATMENT OF ERRORS

Errors may be treated in two ways - they may be corrected or the erroneous results may be subject to special handling (probably rejection). When erroneous results are given special handling, the manner in which they should be handled is normally more closely related to the application than to the computer hardware. Therefore, the remainder of this section will be devoted exclusively to error correction.

C.2.1 Error Correction by Redundant Logic

C.2.1.1 Self-Correcting Codes

The most thoroughly explored area of error correction thru redundancy is the self-correcting code. The theory of self-correcting codes is related to the theory of self-checking codes. Since the general theory is widely known, it will not be discussed further.

When self-correcting codes are used, it is possible to correct the error without setting an alarm indicating the existence of the error.

C2.1.2 Self-Correcting Logic Circuits

Logic networks can be so constructed so that a single standard fault in a segment of logic will not cause an error. Furthermore, many other types of faults causing an erroneous signal at only one node within that segment of logic will not cause an error at the output of the network. Many techniques have been developed for this type of redundancy. Among them is Quadded Logic.

The reliability advantage of self-correcting logic will now be discussed. Assume that the probability of a fault in a segment of logic is p and that all faults are statistically independent of one another. Also assume that a new redundant system will contain one logic segment and will fail if there is a single fault and that a redundant system will contain n logic segments and will fail if there are faults in two or more segments. The probability that the non-redundant system will not fail is

$$(1 - p)$$

The redundant system will not fail if none of the logic segments contains a fault. The probability that none of the logic segments contains a fault is

$$(1 - p)^n$$

Also, it will not fail if only one logic segment contains a fault. The probability that exactly one logic contains a fault is

$$n(1 - p)^{n-1}p$$

Hence, the probability that the redundant system will not fail is

$$(1 - p)^n + n(1 - p)^{n-1}p$$

so that the probability that it will fail is

$$1 - (1 - p)^n - np(1 - p)^{n-1} = \frac{n(n-1)}{2!}p^2 - \frac{2n(n-1)(n-2)}{3!}p^3 + \frac{3n(n-1)(n-2)}{4!}p^3 - \ldots$$

Hence the probability of failure for the redundant system is not greater than

$$\frac{n(n-1)}{2}p^2$$

Therefore the probability of failure is less for the redundant system than for the non-redundant system if

$$p < \frac{2}{n(n-1)}$$

If faults are not eliminated from a redundant system during check-out, its advantage in enhanced freedom from uncorrected errors is diminished. Suppose that there are N logic segments in the non-redundant system and N corresponding sets of n segments in an equivalent redundant system. Then there will be an uncorrected

error in the non-redundant system if there is a fault in any segment and there will be an uncorrected error in the redundant system if there are two faults in any segment.

The probability that there are no faults in any of the N segments of the non-redundant system is

$$(1 - p)^N$$

If each of M sets of segments in the redundant system has a single fault, the probability that none of these M sets of n segments has more than one fault is

$$\left[(1 - p)^{n-1}\right]^M$$

The probability that none of the remaining N-M sets of n segments has two faults is

$$\left[(1 - p)^n + np(1 - p)^{n-1}\right]^{N-M}$$

Therefore, the probability that there will be no uncorrected error in N sets of n segments in which m sets of segments contain a single fault is

$$\left[(1 - p)^{n-1}\right]^M \left[(1 - p)^n + np(1 - p)^{n-1}\right]^{N-M} = (1 - p)^{(n-1)N} \left[1 + (n - 1)p\right]^{N-M}$$

The probability of no uncorrected error will be greater for the redundant system if

$$1 < \frac{(1 - p)^{(n-1)N} \left[1 + (N - 1)p\right]^{N-M}}{(1 - p)^N} = (1 - p)^{(n-2)N} \left[1 + (n - 1)p\right]^{N-M}$$

Let

$$N-M = \frac{(n - 2)N}{n - 1}(1 + B)$$

Then this condition becomes

$$1 < \left\{(1 - p)^{n-1} \left[1 + (n - 1)p\right]^{1+B}\right\}^{\frac{(n-2)N}{n-1}}$$

This is equivalent to the condition

$$1 < (1 - p)^{n-1} \left[1 + (n - 1)p\right]^{1+B} = \left[(1 - p)^n + np(1 - p)^{n-1}\right] \left[(1 + (n - 1)p\right]^B$$

Now the first factor is the probability that there will not be two faults in a set of n segments. This is less than one. The second factor is greater than one. Hence B must be positive and it must be large enough to compensate for the first factor. Note that $B > 0$ if

$$N > (n - 1)M.$$

C2.2 Error Correction by Rollback

If an error is detected it is often possible to correct the error by repeating the calculation.

Usually rollback procedures repeat the calculation using exactly the same logic paths as were used when the error occurred. Such procedures can correct transient errors. The number of times the same calculation can be rolled back before an alarm is set is usually specified. If this number is exceeded, it could be decided that the error was not transient.

It is also possible to roll back some computations so that the logic paths used in the second computation are different from those used in the first. If an adder is structured so that complementing the inputs will complement the signals at all nodes, a simple roll back procedure suggests itself. If an error arises in the performance of an addition, complement all the inputs to the adder. Then the output of the adder would be complemented to obtain the desired sum. As was previously explained, if an adder has a structure satisfying this condition, no standard fault will cause both the original addition and the addition with complemented inputs to be incorrect.

Other rollback procedures for addition are possible. For example, the addend and the augend can be shifted so that the faulty adder bits are confronted with a combination of inputs which do not exercise the faults. However, for an appropriately structured adder this technique is less powerful than complementation.

Rollback procedures often can be designed to take advantage of the many redundant paths in most computers to correct an erroneous computation. However, the design of most of these procedures depends upon the detailed structure of the system.

C3. SYSTEM RECONFIGURATION OR REPAIR

Three courses of action are possible after a system is discovered to be faulty:

1. Discontinue operations
2. Continue operations, accepting the possible consequences of the faults
3. Reconfigure or repair the system

The first solution is unthinkable. It is the situation which would have to be faced if all other available courses of action are impossible or excessively perilous. In most situations, this solution would not be accepted unless the computer were a failure.

The second solution is better. If the computer has built-in error correction features this solution could be entirely satisfactory. If the system is designed so that operations can tolerate some errors, continuation of operations might still be acceptable as a form graceful degradation. Finally, it might be possible to reprogram the system or redesign the sequences of operations in some of the instructions so that the harmful effects of errors due to known faults will be remedied. This approach, however, is a form of reconfiguration.

The third solution is reconfiguration or repair. In this solution, either the hardware or the software is modified to eliminate or reduce the harmful effects of known faults.

C3.1 Repairs

Repairs performed with perfection will not degrade operations. However, their performance requires spare parts, tools, skills and/or time. When these are available, repairs provide the best maintenance.

To make repairs with a minimum amount of test equipment, it is necessary to have accurate knowledge of the location of a fault which is to be removed from the system. This knowledge will be provided whenever possible by diagnostic programs and built-in fault detection circuits.

The simplest repair procedure involves the removal and replacement of a faulty module. This method is the simplest to perform. However, this means that spare parts provisioning must be provided at the modular level.

Another repair technique would involve simple adjustments to remove from the active system faulty elements within a module. Such adjustments also would involve a knowledge of the location of the fault. Furthermore, in this case, the fault would have to be located more closely than to the module.

As an example of a repair technique by adjustment, consider a memory with a spare bit position. If one of the active bit positions was found to be defective, the fault could be remedied by disconnecting the faulty bit position and connecting the spare bit position into its place.

The facility with which this could be performed would depend upon packaging techniques. One technique which would facilitate this type of repair would be a circular connector. All bit positions of the memory would be connected to the memory side of the connector. However, on the computer side of the connector, one pin would not be connected. If one bit of the memory were found to be faulty, the connector could be rotated so that that faulty bit position would be connected to the disconnected pin on the computer side.

C3.2 Reconfiguration

C3.2.1 Reconfiguration at the Model Level

The simplest form of reconfiguration involves switching out a defective module and possibly switching in a sound one. For this technique, it is necessary to divide the system into modules and to have more than one module of each type. Some modules may be spares. In this type of reconfiguration, no operational degradation occurs until the spares are consumed.

C3.2.2 Reconfiguration at the Sub-Module Level

The reconfiguration need not be done at the module level. Spare logic circuits can exist within a module. Then, a register can be used to control the selection of circuits used. If error detection or error correction circuitry are used, it may be desirable to provide test modes for switching out some of the redundant circuitry while the cooperating logic elements are being checked out. In systems with this feature a test mode might also be used to remove defective redundant circuitry from the active circuit.

If a memory or a transfer bus had a faulty bit position and a spare bit position, the faulty bit position could be switched out and the sound bit position could be switched in.

If a mod 2^n-1 adder had a faulty bit position, the adder could be converted to a mod 2^n-1 adder by breaking the connections between the defective bit and its neighbors. In this case, it would also be necessary to reconfigure the sign tests so that the proper bit would be tested as the sign. An equivalent effect could be obtained by inserting a spare bit in the adder and disabling it. Then, if a bit were found to be defective, the spare bit would be enabled and the defective bit disabled.

In a micro-programmed computer, a defect in the memory storing the micro-program could be evaded by changing the operation code of the instruction whose micro-program was located in the defective memory locations.

C3.2.3 Reconfiguration by Instruction Sequence Modification

The defects of an adder will usually cause an erroneous power of two to be added to or subtracted from the results whenever the operands cause certain logic paths to be used. Thus, if an adder is known to contain a fault (or several faults) the error patterns which might occur can be listed. Hence a roll back routine can be designed for correcting the error pattern associated with the given fault.

As the error pattern will not occur in every addition, this roll back technique cannot be used with every addition. It would therefore be necessary to use some error detection scheme to determine whether or not the operands currently being combined cause an error. Thus the roll-back routine should be used only after an error has been detected.

REFERENCES

1. Study of Subsystems Required for a Mars Mission Module SID64-1-1 through 5. Contract NAS-9-1748, Space and Information Systems Division, North American Aviation, Inc. (2 January 1964)

2. Manned Mars Landing and Return Mission Study. SID 64-619-3. Contract NAS-2-1408, Space and Information Systems Division, North American Aviation, Inc. (April, 1964).

3. A Study of Mission Requirements for Manned Mars and Venus Exploration FZM-4366-3 vol 3. Contract NAS-8-1131-8, General Dynamics/Fort Worth (30 May 1965).

4. Manned Mars and Venus Exploration Study. GD/C AOK 65-002, Vol. 1, 2 and 3. Contract NAS-8-11327, General Dynamics/Convair (21 May 1965).

5. Space Navigation Guidance and Control, R-500 vol. 1 and 2. Contract NAS-9-4065, MIT Instrumentation Laboratory (June 1965).

6. Study of Conjunction Class Manned Mars Trips. Douglas Report SM-48662. Contract NASw-1028, Douglas Missiles and Space Systems Division (June 1965).

7. Sohn, Robert L., "A Chance for an Early Manned Mars Mission" Astronautics and Aeronautics, Vol. 3, No. 5, pp 28-33, May 1965.

8. Bell, M. W. J., An Evolutionary Program for Manned Interplanetary Exploration, AIAA/AAS Stepping Stones to Mars Meeting, Baltimore, Maryland, March 28-30, 1966, pp. 87-98.

9. Wood, E. C. and Greene, D. W., On-Board Checkout System Concept AIAA/AAS Stepping Stones to Mars Meeting, Baltimore, March 28-30, 1966. pp. 263-268

10. Spaceborne Memory Organization (Appendices) Interim Report 120171RI. Contract NAS-12-38, Honeywell Systems and Research Division (15 December 1965).

11. S-IIB Orbital Launch Vehicle SID 65-895. Space and Information Systems Division, North American Aviation, Inc. (Sept. 1965).

12. Standardized Space Guidance System, Autonetics, Division of North American Aviation, Report #SSD-TDR-64-129 Annex G

13. Study of an Advanced Energy Management System for Re-Entry Vehicles, Bell Aerosystems Company, FDL-TDR-64-79

14. Manned Mars and/or Venus Flyby Vehicle Systems Study, North American Avaition, SID 65-761-4

15. Data Compression by Quantiles, JPL Space Program Summary, No. 37-17, Volume IV

16. Study of Spaceborne Multiprocessing, Autonetics, Anaheim, Calif., 1st Quarterly Report, C6-1476.1/33.

17. Study of Spaceborne Multiprocessing, Autonetics, Anaheim, Calif., 2nd Quarterly Report, Volumes 1 and 2, C6-1476.4/33.

18. Study of Spaceborne Multiprocessing, Autonetics, Anaheim, Calif., 3rd Quarterly Report, C6-1476.8/33.

19. Modern Probability Theory and Its Applications, E. Parzen, Chapter, 5, Wiley and Sons, 1960.

20. Survey of Highly Parallel Information Processing Technology and Systems, Westinghouse, Baltimore, Maryland.

21. Advanced Computer Organization Study Vol I and Vol II, Goodyear Aerospace Corp., Akron, Ohio.

22. The Solomon Computer, Slotnick, et. al., Proc. - Fall Joint Computer Conference/1962.

23. A Study of Iterative Circuit Computers, TDR AL-TDR-64-24, Air Force Avionics Laboratory, Wright Patterson AFB.

24. A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously, John Holland, Proc. - Eastern Joint Computer Conf./1959.

25. Intercommunicating Cells, Basis for a Distributed Logic Computer, C.Y. Lee, Proc. - Fall Joint Computer Conf./1962.

26. Medium-Speed Mass Random-Access Memory, C. Chong, G. Reid, A. Turezyn, Tech. Rpt. No. RADC-TR-64-571, March, 1965.

27. Study and Investigation of Technique for Constructing Medium-Speed Random Access Mass Memory, Tech. Rpt. No. RADC-TR-64-538, March, 1965.

28. A Magnetic Film Memory Development Program, I.I. Raffel, Lincoln Laboratory, Mass. Inst. Tech., March, 1965.

29. Controlled Domain Tip Propagation Part II, R.I. Spain and H.I. Jauvtis, Journ. Appl. Phys. 37, 2584, June, 1966.

NASA-Langley, 1970 — 8 CR-1446