

NASA CONTRACTOR REPORT



NASA CR-1159

NASA CR-1159

GPO PRICE \$ _____

CFSTI PRICE(S) \$ _____

Hard copy _____

Microfiche _____

Microfilm _____

FACILITY FORM 602	(ACCESSION NUMBER)	(THRU)
	34	1
	(PAGES)	(CODE)
	CR 1159	
	(NASA CR OR TMX OR AD NUMBER)	(CATEGORY)

STUDY OF SPACEBORNE MULTIPROCESSING — PHASE II

Volume II — Technical Description

by *L. J. Koczela*

Prepared by

NORTH AMERICAN ROCKWELL CORPORATION

Anaheim, Calif.

for Electronics Research Center



STUDY OF SPACEBORNE MULTIPROCESSING — PHASE II

Volume II — Technical Description

By L. J. Koczela

Distribution of this report is provided in the interest of information exchange. Responsibility for the contents resides in the author or organization that prepared it.

Issued by Originator as Report No. C6-1476.22/33

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

for Electronics Research Center
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

FOREWORD

This final report describes the results of the Phase II portion of a study conducted under NASA contract NAS 12-108, "Spaceborne Multiprocessor Study". It was performed by Autonetics, a division of the Aerospace and Systems Group of the North American Rockwell Corporation. The work was administered under the direction of the National Aeronautics and Space Administration, Electronics Research Center, Computer Research Laboratories, Cambridge, Massachusetts. The NASA project manager was Mr. G. Y. Wang.

The contract participants during this phase and their primary contributions are listed below:

L. J. Koczela	- Principal Investigator, Requirements, Parallelism, Input/Output, Failure Detection and Reconfiguration, Cell and Group Switch Design, Communication Bus, Reliability Simulation, RF Communications
P. N. Bogue	- Group Architecture, Executive Design, Example Program, RF Communications
G. J. Burnett	- Computer Structures, Cell Design
T. T. Carter	- Reliability Simulation
H. W. Copenhagen	- Reliability Simulation
J. A. Luisi	- Technology
J. R. Macy	- RF Communications
W. E. Meyer	- RF Communications
D. M. Motley	- RF Communications

This report is being published as Autonetics document number C6-1476.22/33.

CONTENTS

	<u>Page</u>
Foreword	iii
Summary	1-1
1. Introduction	1-1
2. Requirements	2-1
2.1 Missions	2-1
2.2 Failure Detection and Reconfiguration Time	2-21
2.3 Computational Requirements	2-22
3. Technology	3-1
3.1 LSI Today	3-1
3.2 LSI in Two Years	3-7
3.3 LSI in Ten Years	3-10
4. Parallelism	4-1
4.1 Introduction	4-1
4.2 Assignment and Sequencing Parallel Operations	4-3
4.3 Application of Parallelism Studies	4-6
4.4 Results of Parallelism Studies	4-19
5. Computer Structures	5-1
5.1 Development of Computer System Architecture	5-1
5.2 Operation of the Distributed Array Memory and Processor System	5-12
5.3 Communication Structures	5-16
6. Group Architecture	6-1
6.1 Introduction	6-1
6.2 Cell States	6-1
6.3 Cell Identification	6-3
6.4 Source of Instructions	6-4
6.5 Sources of Addresses	6-7
6.6 Sources of Data	6-9
6.7 Execution of Instructions	6-12
6.8 Neighbor to Neighbor Communications	6-33
6.9 Additional Topics	6-35

CONTENTS (Cont)

	<u>Page</u>
7. Input/Output	7-1
7.1 Input/Output Operation	7-1
7.2 I/O Mechanization	7-11
8. Failure Detection and Reconfiguration	8-1
8.1 Failure Detection Methods	8-1
8.2 Reconfiguration	8-5
8.3 Backup Equipment Assurance	8-15
8.4 External Status Reports	8-15
9. Cell and Group Switch Design	9-1
9.1 Processor Features of the Cell	9-1
9.2 Functional Description of the Cell	9-17
9.3 Group Switch	9-61
10. Communication Bus	10-1
10.1 Inter-Cell Bus	10-1
10.2 Inter-Cell/Inter-Group Communications	10-14
10.3 Mechanization of Inter-Cell Communication Bus Commands and Control Words	10-19
10.4 Mechanization of Inter-Cell Communication Bus Operations in a Cell	10-29
11. Executive Design	11-1
11.1 Introduction	11-1
11.2 Group Executive	11-1
11.3 System Executive	11-27
11.4 Cell Executive	11-29
12. Reliability Simulation	12-1
12.1 Monte Carlo Method	12-1
12.2 Simulation Models	12-4
12.3 Simulation Results	12-9
12.4 Summary and Conclusions	12-30
13. Summary and Recommendations	13-1

CONTENTS (Cont)

	<u>Page</u>
Appendix A. Example Program	A-1
A.1 Introduction	A-1
A.2 Example 1	A-4
A.3 Example 2	A-9
Appendix B. Error Analysis of Monte Carlo Program	B-1
Appendix C. Monte Carlo Simulation Results	C-1
Appendix D. Communications	D-1
D.1 Introduction	D-1
D.2 Communication Requirements	D-1
D.3 RF Circuit Technology	D-2
D.4 RF Communication Systems	D-5
D.5 Computer Organizational Considerations	D-22
D.6 Conclusion	D-26
Appendix E. Error Control in the Communications System	E-1
Appendix F. Alternative Distributed Organizations	F-1
References	R-1
Bibliography	B-1

ILLUSTRATIONS

Figure		Page
1-1	Distributed Processor Organization	1-3
2-1	Missions	2-2
2-2	Experiment Location Tree	2-11
2-3	Scientific Experiment Computational Functions	2-15
2-4	Interplanetary Lander Mission Storage Requirements	2-23
2-5	Interplanetary Lander Mission Speed Requirements	2-24
2-6	Interplanetary Flyby Mission Storage Requirements	2-25
2-7	Interplanetary Flyby Mission Speed Requirements	2-26
2-8	Computer Storage Requirements-Manned-Mars Lander Mission	2-27
2-9	Computer Speed Requirements - Manned-Mars Lander Mission	2-28
3-1	SOS Complementary MOS Memory Array	3-6
3-2	Packaging Concept	3-9
3-3	Layout of Distributed Processor	3-12
4-1	Sequential Steps in Computation	4-1
4-2	Applied Parallelism in the Computation	4-2
4-3	Applied and Natural Parallelism in the Computation	4-2
4-4	Sequential Solution of Computational Problem	4-7
4-5	Utilization of Applied Parallelism in the Computational Problem	4-8
4-6	Utilization of Applied and Natural Parallelism in the Computational Problem	4-9
4-7	Assignment and Sequencing of Tasks in the Applied and Natural Parallelism	4-10
4-8	Degree of Parallelism Utilized	4-11
4-9	Degree of Parallelism vs. Computation Reduction Ratio	4-12
4-10	Degree of Parallelism vs. Storage Required Per Cell	4-13
4-11	Computational Problem Assignment Among Groups	4-14
4-12	Degree of Parallelism vs. Storage Required Per Cell Using Groups of Cells	4-14
4-13	Degree of Parallelism vs. Number of Groups	4-15
4-14	Assignment of Tasks	4-17
4-15	Storage Graph/Cell	4-18
4-16	Applied Parallelism Speed Curve	4-21
4-17	Applied Parallelism Speed Curve	4-22
4-18	Natural Parallelism Speed Curve	4-23
4-19	Natural Parallelism Storage Curve	4-24
5-1	Structure of the Cob Web Array (From Ref. 24)	5-3
5-2	Holland Machine	5-5
5-3	Solomon Machine	5-5
5-4	General Cell Block Diagram	5-8
5-5	Distributed Processor Organization	5-9
5-6	Full Intercommunication Distributed Processor	5-17
5-7	Arbitor Communication Structure	5-19

ILLUSTRATIONS (Cont)

<u>Figure</u>	<u>Page</u>
6-1 Bus Operation Example	6-6
7-1 I/O Structure	7-2
7-2 Intergroup Bus I/O Scheme	7-3
7-3 Two Methods of Intergroup Bus I/O	7-4
7-4 Intercell Bus I/O	7-6
7-5 Selected I/O Approach	7-7
7-6 I/O Using Neighbor Lines	7-9
7-7 Communications Lines to a Cell	7-10
7-8 Request I/O System	7-15
8-1 Active Redundant Test Cells Within a Cell Group	8-3
8-2 Output Switching of Critical Conditioners	8-13
9-1 12-bit Instruction Word	9-4
9-2 Use of Two B/T Bits	9-4
9-3 14-bit Instruction Word	9-6
9-4 Use of Three B/T Bits	9-7
9-5 16-Bit Instruction Words	9-10
9-6 18-Bit Instruction Word	9-10
9-7 Byte Instruction Word	9-16
9-8 General Cell Block Diagram	9-18
9-9 Detailed Cell Block Diagram	9-19
9-10 Real Time Clock	9-50
9-11 Neighbor-to-Neighbor Communication	9-55
9-12 Basic Memory Cell Utilizing Complementary MOS Transistors Without Selection or Readout Provisions	9-57
9-13 Logical Operation of a Coincident Select Memory Cell	9-58
9-14 Organization of a Coincident Select Bit Plane Array	9-59
9-15 Connection of 6 Arrays to Form a 512 Word,16 Bit, Memory	9-60
9-16 Group Switch Block Diagram	9-62
11-1 Link List for RTC Requests	11-4
11-2 Periodic Control of System Resources by Real Time Clock	11-5
11-3 Processing a Cell's Request	11-7
11-4 Cell Poll Executive Routine	11-9
11-5 Executive Flow Chart	11-16
11-6 Preliminary Time Schedule, Inter-Cell Bus	11-18
11-7 Constructing the Bus Schedule	11-19
11-8 Shifting the Bus Schedule	11-19
11-10 Splitting the Periodic Use of the Bus	11-20
11-11 Cell Assignment Procedure	11-21
11-12 Software Test and Reconfiguration Routines	11-23
11-13 Machine Error Routine	11-31
11-14 Inter-Cell Bus I/O Routine	11-32
11-15 Inter-Cell Bus I/O Complete Routine	11-34

ILLUSTRATIONS (Cont)

<u>Figure</u>	<u>Page</u>
11-16 Cell Executive Real Time Clock Routine	11-35
12-1 Block Diagram of Monte Carlo Simulation	12-3
12-2 Mars Orbital Configuration	12-5
12-3 Non-Critical Phase Configuration	12-7
12-4 Critical Phase Configuration	12-8
12-5 Example of Computer Print Out	12-10
12-6 Average Availability vs Number of Spare Cells Per Group.	12-14
12-7 Average Availability vs Cell MTBF Varying Number of Spare Cells	12-16
12-8 Average Availability vs Cell MTBF Varying Number of Inter-Group Busses	12-17
12-9 Average Availability vs Cell MTBF Varying Number of Inter-Cell Busses	12-18
12-10 Average Availability vs MTBF-OFF/MTBF-ON	12-19
12-11 Average Availability vs % of Failures Affecting Busses	12-20
12-12 Percent Unavailability vs Mission Time for N=0, K=0, Y=0	12-21
12-13 Percent Unavailability vs Mission time for N=0, K=1, Y=0	12-22
12-14 Percent Unavailability vs Mission Time for N=1, K=0, Y=0	12-23
12-15 Percent Unavailability vs Mission Time for N=1, K=1, Y=0	12-24
12-16 Probability of Success vs MTBF-OFF/MTBF-ON	12-26
12-17 Cell and Group Switch Failures vs Cell MTBF	12-27
12-18 Cells with External Connections Failing vs Cell MTBF	12-28
12-19 Inter-Cell and Inter-Group Bus Failures vs Cell MTBF	12-29
A-1 Navigation Program Flow Diagram	A-3
A-2 Example 1, Inter-Cell Bus Requirements	A-7
A-3 Example 2, Program Allocation to Cells	A-10
A-4 Example 2, Inter-Cell Bus Usage	A-12
A-5 Example 2, Timing	A-27
B-1 Error vs P_S in Monte Carlo Results	B-4
D-1 Dielectric Effects on Size	D-3
D-2 Solid-State RF Power Oscillators	D-4
D-3 Signal/Crosstalk Ratio Plot	D-7
D-4 Intra-Computer Communication System Simplified Block Diagram (Single Channel)	D-8
D-5 Probability of Error as a Function of Data, Threshold Level and Signal-to-Noise Ratio	D-12
D-6 Theoretical Performance of ON-OFF Amplitude Modulation with Envelope Detection in White Gaussian Noise	D-13

ILLUSTRATIONS (Cont)

<u>Figure</u>	<u>Page</u>
D-7 Simple Time Multiplex	D-15
D-8 Simple Frequency Multiplex	D-16
D-9 Detailed Block Diagram of Simple Frequency Multiplex	D-18
D-10 Full Frequency-Simple Time Multiplex	D-20
D-11 Construction of New Cell	D-24
D-12 Time Frequency Spectrum	D-25
D-13 Multiple Dependent Program Sources	D-25
F-1 Decentralized DAMP System	F-2

TABLES

<u>Table</u>	<u>Page</u>
2-1 Mars Planetary Mission Computation Requirements	2-19
3-1 Integrated Equipment Components - 1966	3-3
3-2 Extrapolations to Distributed Processor	3-4
4-1 Applied Parallelism Results	4-19
4-2 Natural Parallelism Results	4-20
6-1 Cell States	6-2
6-2 Instruction Categories	6-13
6-3 Summary of Instruction Execution	6-14
6-4 CC Instructions	6-16
6-5 GC Instructions	6-17
6-6 GC-CC Instruction Execution	6-18
6-7 GC Formats	6-24
6-8 CC Transmitted Instructions	6-31
10-1 Communication Bus Operations	10-3
10-2 Communication Bus Commands	10-3
10-3 Inter-Cell Bus Group Switch Commands	10-15
10-4 Intergroup Bus Group Switch Commands	10-17
11-1 Control Program Data	11-3
11-2 Hardware Resources Table	11-11
11-3 Software Requirements	11-12
11-4 Program Requirements	11-13
12-1 Mission Phases	12-4
12-2 Simulation Results	12-12
A-1 Example 1 Storage Estimates	A-5
A-2 Relative Times Used in Programming Examples	A-6
A-3 Communication Sequence Timing	A-8
A-4 Subroutine Timing Requirements	A-9
A-5 Program for 1.2.4.1 (Ref 2)	A-13
A-6 Program for 1.2.4.2 Through 1.2.4.5 (Ref 2)	A-13
A-7 Controller Cell Program to Generate Sequence One	A-16
A-8 First Sequence Received by CX, CY, CZ	A-17
A-9 Neighbor to Neighbor Communication	A-19
A-10 Second Sequence Received by CX, CY, CZ	A-19
A-11 Third Sequence Received by CX, CY, CZ	A-20
A-12 Fourth Sequence Received by CX, CY, CZ	A-21
A-13 Example 2 Storage Estimates	A-25

SUMMARY

This final report presents the results of a research study of an advanced multi-processor computer organization for future space missions. The organization is entitled the "Distributed Array Memory and Processor" computer. A manned Mars lander mission was selected as a representative mission for application and the computer requirements were developed. The organization developed utilizes an array of cells capable of a high degree of computational parallelism. The feasibility of such cells is dependent upon future LSI technology. Research was carried out in the hardware design and software analysis of this organization.

1. INTRODUCTION

The purpose of this study was to investigate an advanced multiprocessor computer organizational concept. This organization is called the "Distributed Array Memory and Processor" and will be referred to as the "distributed processor." The distributed processor evolved from the Phase 1 portion of this study (Reference 1); the purpose of the Phase 1 portion was to evaluate various multiprocessor organizations for future space missions.

As a base for the study, computer requirements for a future manned space mission were used; in particular, a manned Mars lander mission in the 1980's was used. The basic requirements for this mission were developed during Phase 1. These requirements were updated for this phase of the study and are presented in Section 2 of this report. 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 period such as extended earth orbital space stations.

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 the organization. Although the time period of the mission considered is in the 1980's, it is necessary to have the technology available to hardware designers considerably ahead of when the missions are actually flown. Therefore, technology was projected out 10 years from the present. In particular, MOS⁽¹⁾ LSI technology was considered and it was projected that large wafers containing 100,000 or more active devices would be available.

The organizational concept exhibits the capability for computational parallelism. Section 4 contains a discussion of parallelism within computations and results of an evaluation of the space mission computations to determine the amount of parallelism contained therein are presented.

(1) MOS — Metal-Oxide-Semiconductor

Figure 1-1 contains a block diagram of the organizational concept. The organization is seen to consist of a number of identical cells interconnected in a particular manner. Each cell consists of a general purpose processor section and a small amount of memory (512 16-bit words) on a single MOS wafer. The cells are divided into groups (4 groups of 20 cells each are considered for the spaceborne application) and these groups are connected by an intergroup bus for communication. Within each group the cells communicate with each other by an intercell bus and by neighbor communication lines. Each group will have one cell designated as a controller cell; the remaining cells can be operated independently or dependently of the controller cell. This organization is thereby capable of simultaneously taking advantage of applied (global control) parallelism and natural (local control) parallelism within computations. The organization is seen to be somewhat technologically dependent in that it is desirable to have each cell manufactured on a single wafer, thereby resulting in having one identical wafer throughout the entire organization. However, this is not a rigid restriction since more than one wafer could be used to construct a cell.

The organization offers considerable advantages in application to future manned space missions: (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. The latter advantage is most prominent for this organization since it offers many levels of graceful degradation by reconfiguration around individual cell failures.

The organization was subject to a detailed analysis, in particular, the following topics were considered: (a) the development of this distributed organization, (b) the development of the architecture of the organization, (c) input/output methods, (d) failure detection and reconfiguration, and (e) software and executive programming methods.

As a result of this analysis, a preliminary design of the cell was conducted. In addition, a detailed design of the communication system (the inter-cell bus) was accomplished.

In addition, a reliability simulation using Monte Carlo techniques was conducted for the manned Mars lander mission. Statistics for probability of success and availability were thereby generated.

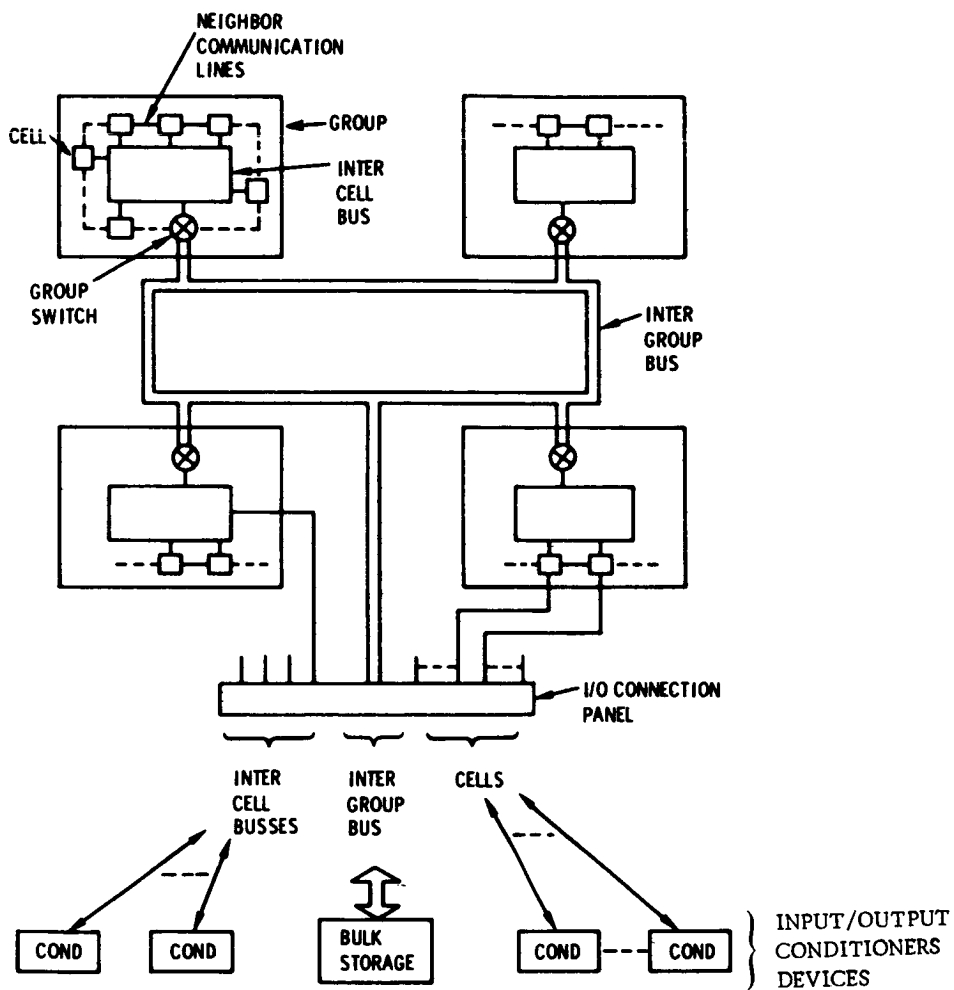


Figure 1-1. Distributed Processor Organization

2. REQUIREMENTS

The previous Phase 1 portion of the study included a detailed investigation of the computer requirements for a manned Mars lander mission in the late 1970's - early 1980's time frame. To be compatible with the greater flexibility and later time period of probable application of the distributed processor, the requirements were re-evaluated to include applicability to more missions and to move the time frame out to span the 1980's. In the first paragraph of this section, the requirements for various missions will be evaluated and updated for the manned Mars mission. In addition, further considerations on the requirements of failure detection and reconfiguration will be presented in this section.

2.1 MISSIONS

Manned and unmanned space missions shall be discussed in this paragraph; within these two classifications several types of missions shall be presented as shown in Figure 2-1.

2.1.1 Manned Missions

2.1.1.1 Orbital Space Stations

These applications may be expected to begin with post Apollo applications and continue indefinitely. It is expected that the first stations would be semipermanent with a lifetime of approximately 1 to 5 years; eventually the space stations would be permanent installations. They will require the servicing of a large crew (9 to 12). On board repair capability due to both on board maintenance facilities and resupply from earth may be expected. Space stations with an extended lifetime may be resupplied periodically or possibly on demand by shuttle vehicles. This is particularly evident when the fact that crews will be interchanged after some length of duty is noted.

This mission/application requires computer system availability as the primary consideration of reliability. Two portions of the mission may be considered to have somewhat of a critical nature associated with the computations: (a) rendezvous and (b) orbital plane changes. However, the criticality of these computations does not appear as severe as those associated with re-entry or midcourse corrections encountered on other missions since failures here may not have as much of a direct effect on crew survival.

Computational Functions: Some of the primary functions to be accomplished in this mission are inventories of earth resources, astronomical observations, weather observations, advanced communication services, and other scientific experiments; in addition the function of a planetary assembly and launch base may also be required.

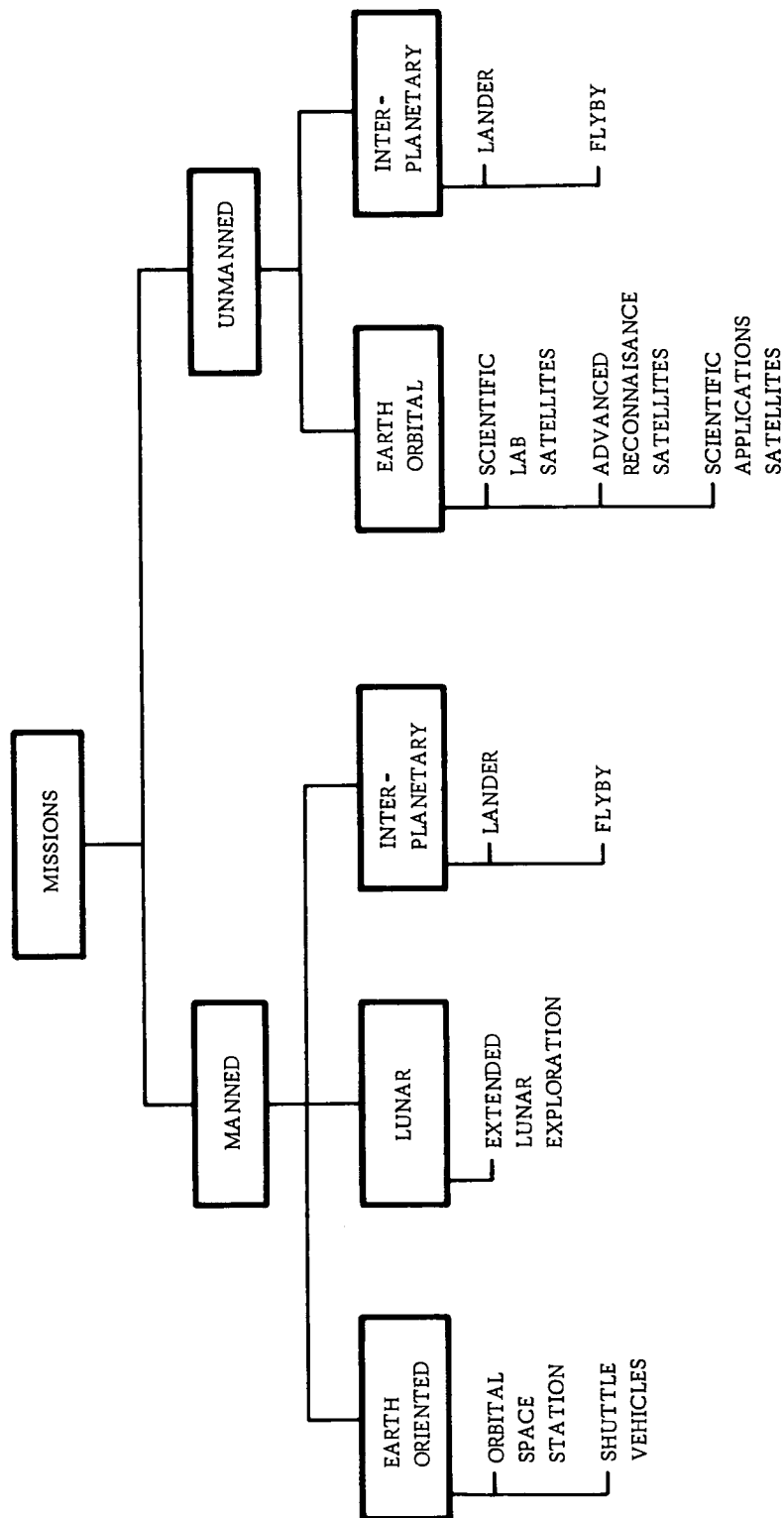


Figure 2-1. Missions

The computational functions that will be required in this mission are:
(1) Orbital Coast functions, (2) Orbital Maneuvering, and (3) Orbital Experiments.
A further listing of functions required in each of these three phases is given below.

1. Orbital coast

a. Orbital navigation

- (1) Rectify osculating orbit
- (2) Update osculating orbit
- (3) Compute perturbations and derivatives
- (4) Update for runge kutta integration
- (5) Update state vector estimate
- (6) Correct state vector

b. Orbit determination

- (1) Noise covariance matrix
- (2) Output matrix
- (3) System description matrix
- (4) A priori statistics
- (5) Filter computation
- (6) Compute correction

c. IMU aline and bias

- (1) Initialize matrices
- (2) Propagation and noise covariance
- (3) Measurement vector and output matrix
- (4) Optimum estimate and control vector

d. Landmark tracker operation

- (1) Selection executive routine
- (2) Pointing signals
- (3) Acquisition and tracking
- (4) Observational residuals
- (5) $\hat{a}_M$ computation

e. Attitude reference

- (1) Star selection executive routine
- (2) Tracker pointing
- (3) Acquisition and tracking
- (4) Kalman filter star data
- (5) Compute C_I^B from gimbal angles
- (6) Compute C_I^L matrix
- (7) Compute C_B^L matrix
- (8) Generate attitude control signal

f. Status monitoring

- (1) Self test operations
- (2) Performance monitoring

g. Telecommunications

- (1) Data compression and processing
- (2) Data formatting
- (3) Command processing
- (4) Transmission instrumentation pointing commands

h. Data display

Estimated requirements

Storage: 12,500 words
Speed: 60,000 operations/second

2. Orbital experiments

- a. Weather observations
- b. Advanced communication networks
- c. Experiment on demand (There will be many experiments onboard; however, generally, only one particular experiment will be conducted at any one time.)
- d. Continuous scientific experiments

There will probably be a considerable amount of data compression done here. Also there will be some data reduction for the experiments. A few major experiments such as in the MOL mission may be expected to require considerable computer mechanization.

Estimated requirements

Storage: 15,000 words
Speed: 150,000 operations/second

3. Orbital maneuvering

- a. IMU mechanization
- b. Navigation computation
- c. Steering
- d. Set up orbit injection

Some of f, g, and h (status monitoring, telecomm, display) under Orbital Coast will also be required here.

Estimated requirements

Storage: 6,500 words
Speed: 30,000 operations/second

It is expected that items (1) and (2), orbital coast and orbital experiments, will be conducted simultaneously; during (3), orbital maneuvering, a small portion of (2), the orbital experiments, may be required. Therefore the maximum requirement occurs during the orbital coast operation when full experimentation is being used.

Estimated total requirements

Storage: 27,500 words
Speed: 210,000 operations/second

It should be noted that it is also possible for the vehicle in this mission to be unmanned at times and possibly still be required to perform some functions such as scientific experiments.

2.1.1.2 Shuttle Vehicles

Logistics support is essential to long duration space stations both for resupply of expendables and transfer of crews. The duration of the mission depends on the length of stay at the space station. Typical missions may be expected to last on the order of 1 to 3 days. During the active portions of the mission no onboard repair can be made. However, the mid-portion of the mission, or the time after rendezvous, may be utilized for repair if necessary.

In general, all of the navigation and guidance computations of this mission may be considered as critical. Probability of success will be the primary reliability consideration in this mission.

Computational functions: the computational functions may be broken down into seven basic phases: Prelaunch, Boost, Orbital, Orbital Maneuver, Rendezvous, De-orbit, and Re-entry.

The subfunctions required during each of these phases are tabulated below.

1. Prelaunch

- a. Accelerometer Calibration
- b. IMU Aline and Bias
- c. Status Monitoring and Checkout
- d. Data Display

2. Boost

- a. IMU mechanization
- b. Navigation computation
- c. Status monitoring
- d. Data display
- e. Telecommunications

3. Orbital

- a. Orbital navigation
- b. Orbit determination
- c. IMU aline and bias
- d. Landmark tracker operation
- e. Attitude reference
- f. Status monitoring
- g. Data display
- h. Telecommunications

4. Orbital maneuver

- a. IMU mechanization
- b. Navigation computation
- c. Steering
- d. Set up rendezvous injection
- e. Set up midcourse maneuver
- f. Status monitoring
- g. Data display
- h. Telecommunications

5. Rendezvous

- a. Sensor pointing
- b. Navigation computation
- c. IMU mechanization
- d. Steering
- e. Status monitoring
- f. Data display
- g. Telecommunications

6. De-orbit

- a. Set up de-orbit
- b. Steering
- c. Navigation computation
- d. IMU mechanization
- e. Status monitoring
- f. Data display
- g. Telecommunications

7. Re-entry

- a. Navigation computation
- b. IMU mechanization
- c. Steering
- d. Status monitoring
- e. Data display
- f. Telecommunications

The first five phases will probably be loaded into the computer initially; then the last two phases, De-orbit and Re-entry, will be loaded into the computer when needed. This then results in the following computer requirements:

Phases 1 through 5

Storage: 19,500 words
Speed: 85,000 operations/second

Phases 6 and 7

Storage: 11,500 words
Speed: 75,000 operations/second

2.1.1.3 Extended Lunar Exploration

The computational functions required during this mission are very similar to those required in the interplanetary lander mission. The primary difference in this mission is the exclusion of certain functions such as spin up, de-spin, and aerobraking maneuvers required for the command module in the interplanetary mission. Computational functions associated with the lander module are expected to be closely identical for both missions. The lunar lander may be expected to have more equipment on board for experimentation; however, it will also have the capability for a much higher data transmission rate to earth, primarily due to the shorter communication distance. Therefore, the total computational requirement is expected to be similar since more experiments are involved; however, less data processing for each of the experiments (data compression, etc.) need be performed at the lunar base. There will be a significant amount of information collected from satellite probes, roving vehicles, etc. The lunar bases are expected to be semi-permanent or permanent in nature with periodic resupply. The capability for on site repair is expected to exist due to the extended nature of the mission.

2.1.1.4 Interplanetary Lander

As a prelude to most interplanetary lander missions, one or more flyby missions will be conducted to gain preliminary data on important parameters such as atmospheric parameters, radiation environment, etc. The early missions will be to the planets Mars and Venus. A thorough discussion and presentation of the computer requirements for this mission was given in the first quarterly report of the Phase 1 portion of this study. ⁽¹⁾ A brief summary of some of the requirements along with new information and changes in the requirements due to the extended time period of application (the 1980's) is given here.

The computer functions fall into four major categories:

1. Vehicle guidance and control
2. Telecommunications
3. Scientific experiments
4. System checkout and monitor

⁽¹⁾ Reference 2

This is not intended to imply that the vehicle contains equipment with four independent or separate computational functions. For example, the system checkout and monitor function services all the equipment interfacing with the computer system and the vehicle guidance and control function and the scientific experiments function interchange computed results primarily derived from navigation and guidance equipment.

A list of potential instrumentation for scientific experiments was given in the Phase 1 final report.⁽¹⁾ As more data became available (References 3 and 4) it was possible to update and add to the initial scientific experiments function. Also, since the time period of application is beyond that considered in Phase 1, it is likely that a considerably larger scientific experiment payload will be used. The following is a revised tabulation of scientific experiments for interplanetary and Mars orbital phases. This list contains only experiments that require or have the possibility of computer usage. Examples of experiments that do not require the computer are maintenance with special tools, collection and observation of soil samples, etc.

2.1.1.4.1 Interplanetary Experiments. These experiments are all conducted in the command module and will utilize computational facilities on board this vehicle. The experiments may be listed in four basic classes:

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, meteoroid, 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.

A tabulation of some of the specific items to be obtained from these classes of experiments is as follows:

Meteoroid (flux, mass, density, velocity, directional, and spatial distribution)

Magnetic field (mapping)

High-resolution photographs of the Sun

Coronal Photography

Sequential White Light Coronal Photography

Coronal spectra

X-ray scans of solar disk and corona

⁽¹⁾Reference 1

Large-scale coronal photos

Lyman-Alpha spectra of flares

Stellar photometry

Neublar photos in Lyman-Alpha

Multi-color mapping of the Milky Way

Map entire sky in the 1 to 10 μ IR

X-ray mapping of the entire sky

U.V. photometry of strongly reddened stars

U.V. polarimetry of strongly reddened stars

U.V. photometry of galaxies and radio stars

Galaxy photography

Zodiacal light photometry

Solar high-energy protons

Galactic protons

Solar high-energy electrons, Alphas

Solar wind protons

2.1.1.4.2 Mars Orbital Experiments. The scientific experiments to be conducted in the Mars orbital phase of the mission may be broken down into three basic classes:

1. 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 planet, the existence and distribution of plant life, the topography of the planet, geological and geophysical investigations, etc.
2. 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.
3. 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.

In addition some of the four basic classes of experiments listed under Interplanetary will be applicable during the Mars orbital phase also such as magnetic field measurements, charged particles measurements, etc.

To accomplish the desired experimentation objectives the performance of the scientific experiments may be divided into those centering around two separate areas:

1. Mars orbital
2. Mars surface

more specifically, this could be listed as those experiments centered around (1) the Command module which remains in Mars orbit and (2) the excursion module which descends to the surface of the planet and then returns to rendezvous with the command module. A further breakdown as to the location of experiments within these two centers is given in Figure 2-2.

The command module is initially in orbit with the excursion module. There are a number of experiments that will be conducted directly from the command module. In addition, there will be several probes launched from the command module; these may be divided into orbital and lander probes. Orbital probes will be used to attain widely different orbits than that used by the command module such as highly elliptical orbits. These probes will monitor environments and perform reconnaissance functions in areas out of reach of the command module. One or more lander probes will be launched from the command module; these may either be hard or soft-landing probes. Also it is possible for probes to be launched to the martian moons, Phobos and Deimos, in addition to the Martian surface.

In most likelihood these probes will not contain their own computational capability. The scientific and engineering data that is collected will be transmitted back to the command module for data processing, analysis, and storage. The specific scientific experiments to be conducted by these probes will not be listed here since the only interface the computer system has with them is via the transmission link and the data processing on the data received from the probe experiments is expected to be quite similar to that on the experiments conducted in the command module.

The excursion module is launched from the command module at some time after the command module probes have been launched. A base will be set up as a base for surface activities. The scientific experiment activities that will be conducted outside the base will include those on (1) the MOLAB (a roving manned vehicle), (2) automatic stations (unmanned fixed stations), (3) rockets and balloons (launched from the base primarily to study the atmosphere). Computation capability will exist at the base and more than likely on the MOLAB. The other scientific experiment facilities will transmit their data back to the base for data processing, analysis, and storage. The base, in turn, will transmit reduced and formatted experiment data back to the command module. A listing of surface experiments is given in this section.

Some of the specific items expected to be derived from the experiments are listed. The experiments are divided into two areas as noted before: (1) those centering around the Mars orbital activities and (2) those centering around the Mars surface activities.

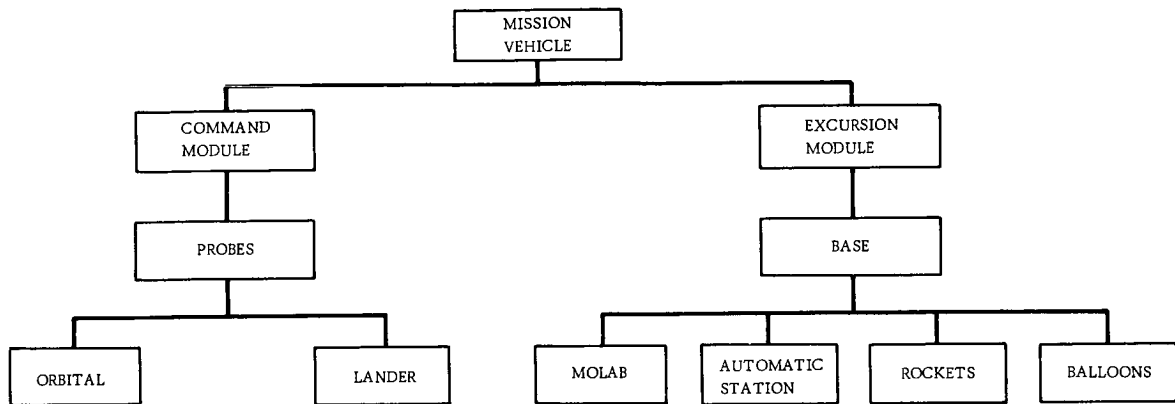


Figure 2-2. Experiment Location Tree

Mars orbital:

1. Orbital parameters such as: semimajor axis, period of revolution, eccentricity of orbit, orbital inclination to the ecliptic, mean daily motion, longitude of the node, longitude of the perihelion, mass, axis inclination, perihelion, aphelion, mean-orbital velocity, length of year, etc.
2. Radiation belts
3. Ionosphere and radio wave behavior
4. Photographic mapping
5. IR mapping
6. Radar mapping
7. Wind velocity
8. Cloud photography
9. Solar spectral absorption
10. Magnetic field

11. Solar plasma
12. Charged particles
13. Topside ionosonde
14. Electron density
15. Ion composition
16. Radio noise monitor
17. Gust distribution
18. Search for satellites
19. Determination of the martian moons' Phobos and Deimos, parameters such as size, shape, surface features, orbital parameters, etc.

Mars surface:

1. Topographic mapping
2. Seismic station short-period range
3. Seismic station long-period range
4. Short-range seismic reflection and refraction
5. Long-range seismic refraction
6. Gravimetry
7. Magnetic field
8. Magnetic permeance
9. Resistivity
10. Temperature
11. Heat flow
12. Beta Gamma mapping
13. Gamma mapping
14. Neutron flux
15. Alpha scattering
16. Gamma scattering

17. Neutron scattering
18. X-ray diffraction
19. IR spectrophotometry
20. Emission spectroscopy
21. X-ray fluorescence
22. Neutron activation
23. Mass spectrometry
24. Gas chromatography
25. Thermal analysis
26. UV absorption spectrophotometry
27. Atmospheric pressure
28. Atmospheric density
29. Wind velocity
30. Total incident sunlight
31. Cloud photography
32. Solar spectral absorption
33. Atmospheric composition
34. Water vapor
35. Oxygen
36. Ozone
37. Lander ionosonde
38. Radio noise monitor
39. Riometer

2.1.1.4.3 Computational Functions for the Scientific Experiments: There are basically four computational functions required to mechanize the scientific experiments. These are:

1. Data processing
2. Sequencing and scheduling
3. Pointing and control
4. Data storage and retrieval

Figure 2-3 shows these four basic functions and the next sublevel in these functions.

2.1.1.4.4 Data Processing. The data processing function consists primarily of data compression, data handling, and data reduction and analysis. The Phase I final report contains a detailed discussion of computational algorithms that may be applied in data compression. Two main types of data compression were cited in this report, namely: Compression by an encoding or curve fitting method whereby the data may be reconstructed after compression and compression by computing some statistical properties of the data (such as mean and variance). Examples of the first type of compression are:

1. Debiasing
2. Difference coding
3. Zero order polynomial predictor
4. Zero order polynomial interpolator
5. First order polynomial predictor
6. First order polynomial interpolator
7. Orthogonal polynomial series

Examples of the second type of compression are:

1. Quantiles representation
2. Representation by moments

As mentioned above detailed computational algorithms were given in the Phase I final report for the data compression methods. Data compression will be used wherever applicable on scientific experiment data to reduce the amount of data (1) to be transmitted back to earth, thereby reducing the communication band width requirements and (2) to reduce the total data storage requirements.



Figure 2-3. Scientific Experiment Computational Functions

The data handling function primarily consists of sampling experimental data and routing it to the proper area for processing, display, transmission, or storage. This is principally an executive control-type function with tables that are cycled through periodically. It should be noted that these tables can be both fixed and variable, since some may require changes on the basis of previous experimental results, changes in the mission, etc.

Data reduction and analysis functionally is the complete reduction of measured data into a useful and meaningful result. With man on board this function becomes important since there is now an adaptive element in the loop controlling the sensors. Based on the results from experiments the astronaut may decide to perform more extensive experiments on an item of particular interest or even to possibly reduce or eliminate certain experiments. This capability is very important when one considers that there is a considerable time delay in communicating data to a ground facility for data reduction and analysis. The communication time delay may preclude the adaption of an experiment to changes in conditions which may exist for a very short time.

Two examples of data reduction and analysis were given in the Phase 1 final report, namely (1) composition analysis of a soil sample and (2) human performance evaluation. The particular computational algorithms will of course depend on the particular experiment involved.

2.1.1.4.5 Sequencing and Scheduling. These computational functions are mainly executive control type of operations. Experiments may be sequenced or scheduled based on time and position information and other events that occur in the mission. The sequencing and scheduling could be fixed or adaptive. The computer is particularly useful when an adaptive role is needed in sequencing and scheduling. The procedures followed could vary considerably based on the outcome of previous experiments, changes in mission plans, etc. Computational functions required to mechanize this consist primarily of tables containing conditional logic sequences.

2.1.1.4.6 Pointing and Control. Some of the experiment sensors will require pointing and control to obtain the desired information in an optimum manner. The more obvious of these being the telescopic systems including the TV, infrared, etc. Computational functions here will require the computer to accept angular position and rate signals from the sensors and based on inertial navigation information compute appropriate pointing commands.

The computer can also be used to control the pointing of scanning sensors to prevent excessive overlap between frames. This involves the coordination of the pointing mechanization with navigational data. This is one example of an interface between two major computational functions namely navigation and guidance and scientific experiments. Another application of the computer in a real time control loop is in the performance of image motion compensation. The functions required here involve logic and arithmetic manipulations similar to those required in the navigation and guidance routines.

2.1.1.4.7 Data Storage and Retrieval. There will be voluminous quantities of data from a wide variety sources and in a broadly different spectrum of formats. The computer can assist in the data storage and retrieval function by assembling and formatting the data for storage.

This will involve placing appropriate headers with blocks of data noting important or relevant mission parameters such as time, position, calibration data, etc. Also the computer can assign storage locations depending on the rate and quantities of data coming in from the total scientific experiments package.

It is expected that the astronaut will require the capability to retrieve data at will from the central data storage facility. He will desire access to data taken from experiments. This data may be called up in different categories, some examples are, by experiment number, by time period, by sequence number (position in a file or experiment), or simply by storage location. The astronaut should have the capability of interrogating the central storage system through simple commands and have it respond by selecting and displaying in an optimally useful way the data relevant to the inquiry.

An example inquiry may be the request by the astronaut to review all temperature measurements made by a sensor over a certain period of time. The system should therefore scan the memory for the relevant data, compile the data with respect to chronological order, and display the results in a form for ease of reading and interpretation. It is conceivable that the requests may also be for data which will require computation on certain measured data. For example, instead of temperature alone the astronaut may desire the heat transfer rates at certain places in the vehicle over a given period of time. The computer system should be flexible to respond to a wide variety of interrogation requests, thereby relieving the astronaut of manual manipulation of data.

In addition to accessing data taken during the mission it is expected that significant quantities of prestored scientific and operational data will be desired in the computer system. The prestored data can be used for a direct comparison with the measured data and generally to assist in the initial interpretation of the measured data.

2.1.1.4.8 Requirements. One new function has been introduced in addition to that presented in the Phase 1 final report namely the data storage and retrieval function. As can be seen from the discussion above this function can essentially use associative features in searching the memory in retrieval of requested data. There are two primary methods of implementing the associative features required. One way is to have an associative memory, the operation of which is well known and need not be discussed here; secondly, it is possible to construct pseudo associative programs. The software approach is particularly attractive since it naturally eliminates the need for the associative memory. Basically the software approach requires the placing of data in areas which are located by one or more identifying headers or keys. When a request for retrieval of data is made, the keys are compared and once a match is achieved that particular block of memory identified by the matching key is searched until the desired data is found. What this method requires is the ability to locate the data according to some fixed number of keys. The data storage and retrieval function for the scientific experiments is expected to be readily implemented by the pseudo associative programming approach.

The requirements in the Phase 1 final report were based on a more near term time period and consequently the resultant data transmission bit rate was considerably lower (20,000 bits/sec) than that which will be realized in the time period of interest

here. The capabilities presently predicted for the 1980's appear to be in the range of 10^6 bits per second (Reference 5).

The fact that a higher information transmission rate will be available coupled with the inclusion of a larger quantity of more sophisticated scientific experiment instrumentation will result in a significantly larger amount of computational requirements for the scientific experiment function. An example of more sophisticated instrumentation is higher resolution video data from better reconnaissance equipment (10^8 to 10^9 bits per frame).

In addition to the increased requirements of the data processing, sequencing and scheduling, and pointing and control functions there are the requirements to be added by the new function, data storage and retrieval. An estimation of the increased requirements for the scientific experiment function is given below. It should be noted that the significant factor in affecting the speed requirements is the percentage of data that will be subject to data processing (data compression or data reduction) prior to data transmission. With the assumption of the higher transmission rate (10^6 bits/sec), it was assumed that 10 percent of the experiment data was subject to data processing such as compression or reduction prior to transmission.

Mars Orbital Phase

Speed: 1,325,000 short ops/sec.

Storage: 9,200 words

Non Mars Orbital Phases

Speed: 235,000 short ops/sec

Storage: 6,500 words

The remaining three major computational functions, navigation and guidance telecommunications, and status monitoring are also expected to increase in terms of computational requirements. However, the increase is expected to be considerably less than that of the scientific experiment function. Therefore, since the effects of these increases will be quite a small percentage, the computer requirements will simply be updated on the basis of the new scientific experiment requirements. A tabulation of the requirements as a function of Mars mission phase is given in Table 2-1.

It should be noted that the same discussion as given in Paragraph 2.9 of the Phase 1 final report on computer requirements applies here. Basically the requirements assume a conventionally organized 18-bit-word-length computer, with multiple accumulators, indexing features, and a basic instruction repertoire.

2.1.1.5 Interplanetary Flyby

The interplanetary flybys will be conducted prior to the lander missions and may contain similar scientific instrumentation in a number of areas. Computational functions similar to those required for the interplanetary lander mission will also be required here. The primary difference between the missions being the fact that there

Table 2-1. Mars Planetary Mission Computation Requirements

Mission Phase	Storage (words)	Speed (short ops/sec)
1. Atm ascent	2, 224	4, 812
2. Earth orbital	10, 303	110, 802
3. Trans Mars Inj	6, 161	57, 554
4. Trans Mars coast	17, 886	314, 860
5. Trajectory corr	12, 799	379, 454
6. Spin up	12, 180	308, 550
7. Spin cruise	18, 476	318, 660
8. De-spin	12, 180	308, 550
9. Mars appr corr	13, 919	379, 454
10. Aerobraking	14, 520	287, 550
11. Mars orbit Inj	12, 449	285, 454
12. Mars orbital	24, 633	1, 449, 402
13. Trans earth Inj	12, 869	301, 254
14. Trans earth coast	17, 886	314, 860
15. Trajectory corr	12, 799	379, 454
16. Spin-up	12, 180	308, 550
17. Spin cruise	18, 476	318, 660
18. De-spin	12, 180	308, 550
19. Earth appr corr	5, 964	199, 554
20. Earth re-entry	8, 620	69, 010

will not be a planetary orbital phase. During the orbital phase for the lander missions, the scientific experiments data is operated on in real time and transmitted back to earth; for the flyby mission there will only be a very short time during which data close to the planet will be obtained. The collected data will be stored on board and then transmitted back to earth during the cruise mode.

There are two approaches to data processing for this data: one is to process the data in real time before it is stored and the other is to store it in real time and simply process it later as it is being transmitted. The first approach will reduce the total storage required at the expense of requiring very high processing speed capability. In most likelihood a combination of the two approaches will be used, that is, for data which is expected to have a very high compression ratio it is likely that real time processing will be applied. The computer requirements for this mission will be similar to those for the lander mission. Of course the requirements for certain phases such as planetary orbit injection and aerobraking are eliminated here. The requirements associated with the data processing function for the scientific experiments are expected to differ somewhat; there will be a lower processing speed requirement during the planetary phase due to the fact that much of the data are simply being stored and the processing speed requirement will be greater during the trans-earth cruise phase when the stored data are being processed for data transmission.

2.1.2 Unmanned Missions

2.1.2.1 Earth Orbital

Unmanned earth orbital missions may be broken down into three basic classes: extensive scientific lab satellites, advanced reconnaissance satellites, and scientific applications satellites. In general, the computer requirements may be characterized by several features: long lifetime satellites will require operation over several years, the computational requirements are relatively constant and not varying as a function of time, and as characterized by all unmanned missions the capability for on board repair due to human action is not available.

The computational requirements are expected to be relatively low in these missions. Some of the basic functions that the computer may be required to perform are:

1. Attitude control
2. Status monitoring and checkout
3. Position determination
4. Data processing and control for experiments
5. Command decoding

2.1.2.2 Interplanetary

Two types of unmanned interplanetary missions will be involved: flyby and lander. Both missions have the characteristic that the capability of human action for onboard repair is not available. The computer requirements for the lander mission are very similar to those for the manned lander up to and including the planetary orbital phase (spin up and de-spin phases are not required here); the unmanned lander mission terminates in the orbital portion, however. The unmanned lander mission will, however, have slightly lower computational requirements due to the elimination of certain functions, such as: display, biomedical and life support, status monitoring, and information storage and retrieval.

The computer requirements for the unmanned flyby are similar to those for the manned mission. However, the unmanned flyby does not require the earth re-entry phase and, therefore, terminates in the trans-earth cruise phase. Spin-up and de-spin also are not required here. In addition the same comments hold true for lower computational requirements due to the elimination of certain functions required in the manned mission.

2.2 FAILURE DETECTION AND RECONFIGURATION TIME

The most critical periods during the manned Mars lander mission occur during the atmospheric entry phases (Mars entry and Earth re-entry). The critical nature of the computations during these phases may be appreciated when one considers that the computer, as part of its navigation and guidance routine, is computing attitude commands to guide the vehicle along some narrow corridor at very high speeds. A malfunction in the computer could cause the issuance of erroneous commands which could force the vehicle towards a temperature or acceleration limit and result in disaster to the space mission.

The entry corridor defines the limits of the incoming velocity vector such that an atmospheric braking maneuver can be accomplished. Entry corridors are generally determined by establishing overshoot and undershoot limits. The overshoot limit is defined as the skip out condition and the undershoot limit is defined as a load factor limit for either structural or biological requirements. Several study reports were investigated to determine what is currently being used as entry corridors and velocities (References 6 and 7). The worst case conditions (generally considered for an 0.5 L/D vehicle) are on the order of 50,000 to 60,000 feet/second entry velocities and 70,000 to 100,000 feet entry corridors. Currently even more stringent requirements are foreseen for further out time period missions (Reference 8); entry velocities on the order of 75,000 feet/second and corridors of 25,000 feet are predicted. These predictions will be used here in this consideration of failure detection and reconfiguration time.

The determination of the maximum allowable time a failure can exist in the entry guidance system is required in order to determine the necessary failure detection and computer reconfiguration time. The determination of this time requires the assumption of some nominal position of the vehicle in the entry limits and an analysis of the dynamics of the vehicle if an error occurs (such as a hard over attitude command). Such an analysis is beyond the scope of this study. However, based on some simple assumptions a worst case time may be determined. If it is assumed that the vehicle is in the center of the corridor (12,500 feet from a boundary limit) and an erroneous command forcing it towards the corridor limit occurs, a worst case time to the limit may be found by assuming the vehicle velocity vector is pointed instantaneously toward the corridor limit. This time then is 166 milliseconds to reach the limit; following through with the same assumption a correction may be applied up to this time to correct the erroneous command. For the purposes of this study a time limit of 100 milliseconds will be set as the failure detection and reconfiguration time. Reconfiguration is defined here as having the critical computational program with all the necessary values of the variables mechanized and being performed correctly in a computational facility after a detected failure. The time defined includes both failure detection and reconfiguration.

2.3 COMPUTATIONAL REQUIREMENTS

The requirements (speed and storage) for the interplanetary missions are shown in Figures 2-4 through 2-7. For purposes of assigning a specific time scale on each figure, Mars planetary missions were considered. These figures basically contain three mission phases: trans Mars, Mars (Mars Orbital or Flyby), and trans Earth; this has been done to clarify the requirements and differences between missions. It should also be noted that portions of the requirements are not continuous but required periodically such as 1/2 hour every 5 hours etc., this also has not been shown so as to point out the basic differences between the missions.

The requirements for the lander mission (manned and unmanned) are shown in Figures 2-4 and 2-5. The unmanned mission is expected to have slightly lower requirements as discussed previously, primarily due to elimination of certain functions associated with manned missions (display, biomedical, etc.) and also because that it will probably precede the manned mission by several years there by utilizing less sophisticated technology (lower transmission rates, less experiments, etc.) It should also be noted that the manned mission has a fixed Mars orbital time whereas the unmanned does not.

Flyby mission requirements are shown in Figure 2-6 and 2-7 for manned and unmanned missions. The requirements show a rise for a short period of time (approx. 1 hour) at the period of flyby and then subside to a level somewhat higher than that of the trans Mars phase. This is due to the fact that the data collected during flyby will be processed during its transmission to earth. For the same reasons as given above the requirements for the unmanned mission are slightly lower than the manned mission. The manned mission has a fixed duration, terminating at Earth re-entry whereas the unmanned mission continues on indefinitely.

Comparing the lander and flyby missions, the requirements during the trans Mars phase are the same for each mission. During the Mars vicinity phase, however, the flyby mission has a much lower requirement in terms of speed and storage increase over the trans Mars phase. This is due to the fact that there will not be real time data processing to any significant extent on data being collected. Only a small portion of the data may be processed and the remainder simply stored. The exact amount of processing done here is not firm and therefore the requirements during this phase are only postulated. The stored data collected during flyby is processed and transmitted to Earth during the initial portion of the trans Earth phase. This accounts for the increase in requirements during this phase over that for the lander mission (data will be processed and transmitted in real time during the Mars orbital phase for the lander mission).

Figure 2-8 and 2-9 represent updated versions of the speed and storage requirements given in Paragraph 2.9 of the Phase 1 final report. This is primarily based on the increased scientific experiment requirements as discussed in Paragraph 2.2 of this report.

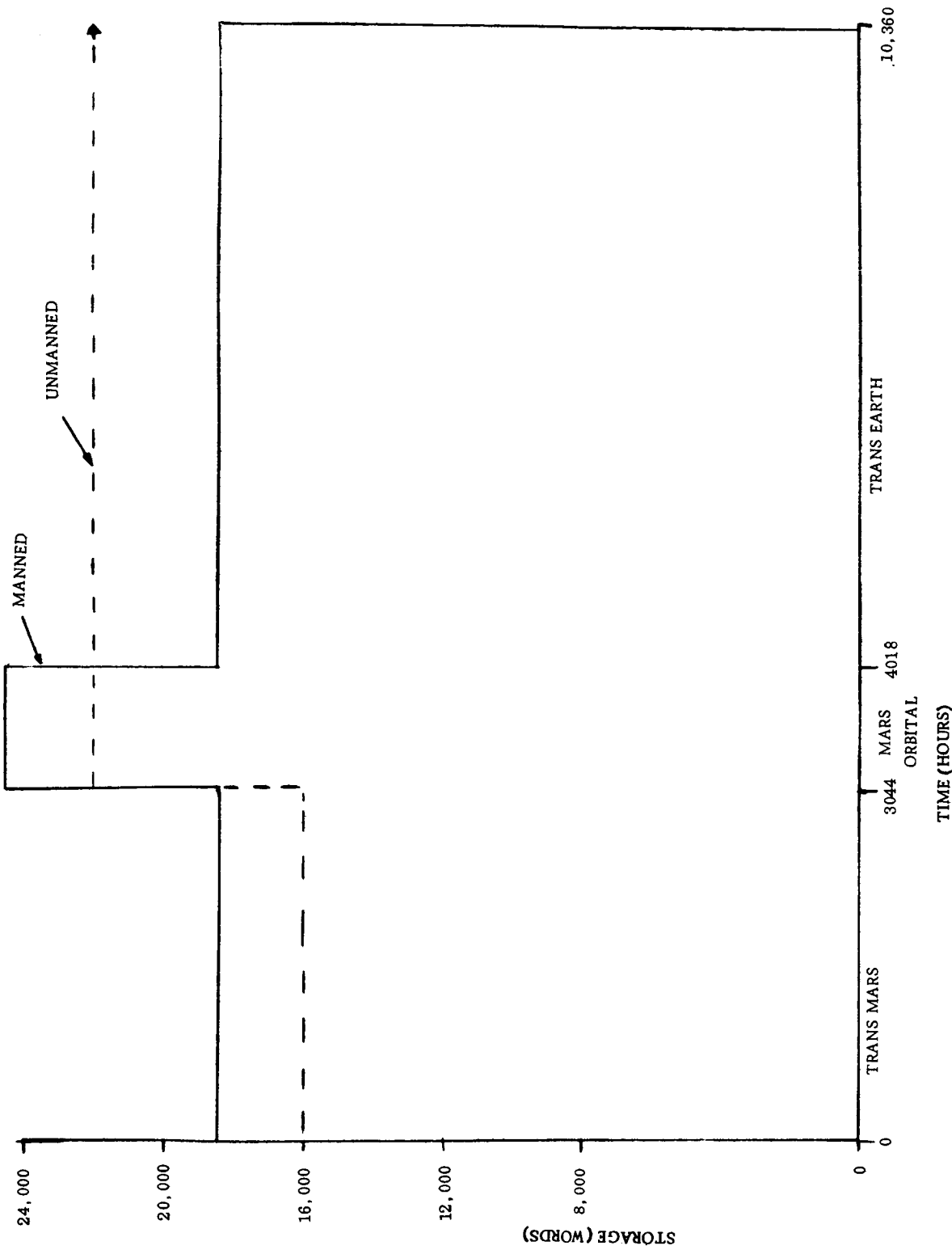


Figure 2-4. Interplanetary Lander Mission Storage Requirements

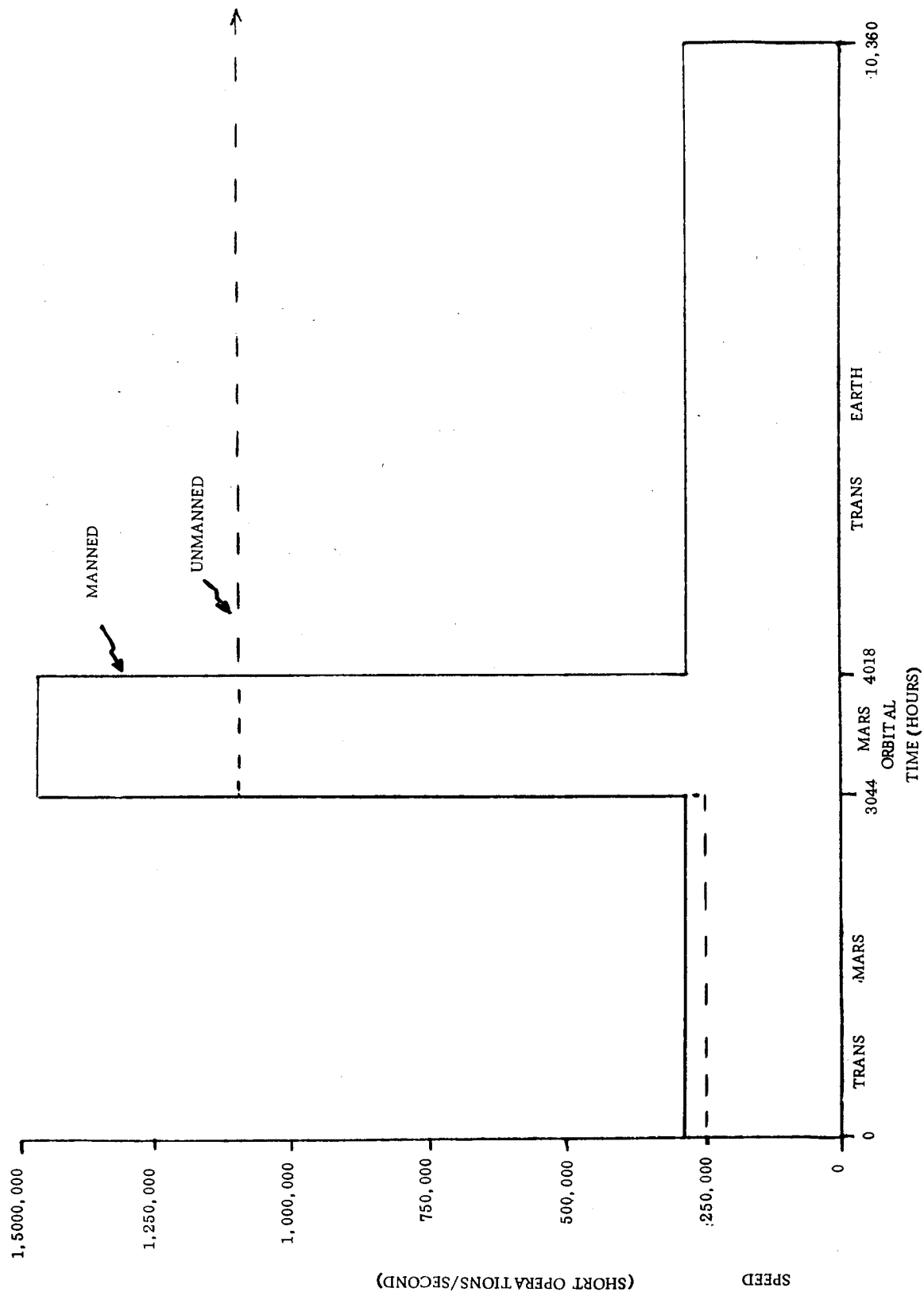


Figure 2-5. Interplanetary Lander Mission Speed Requirements

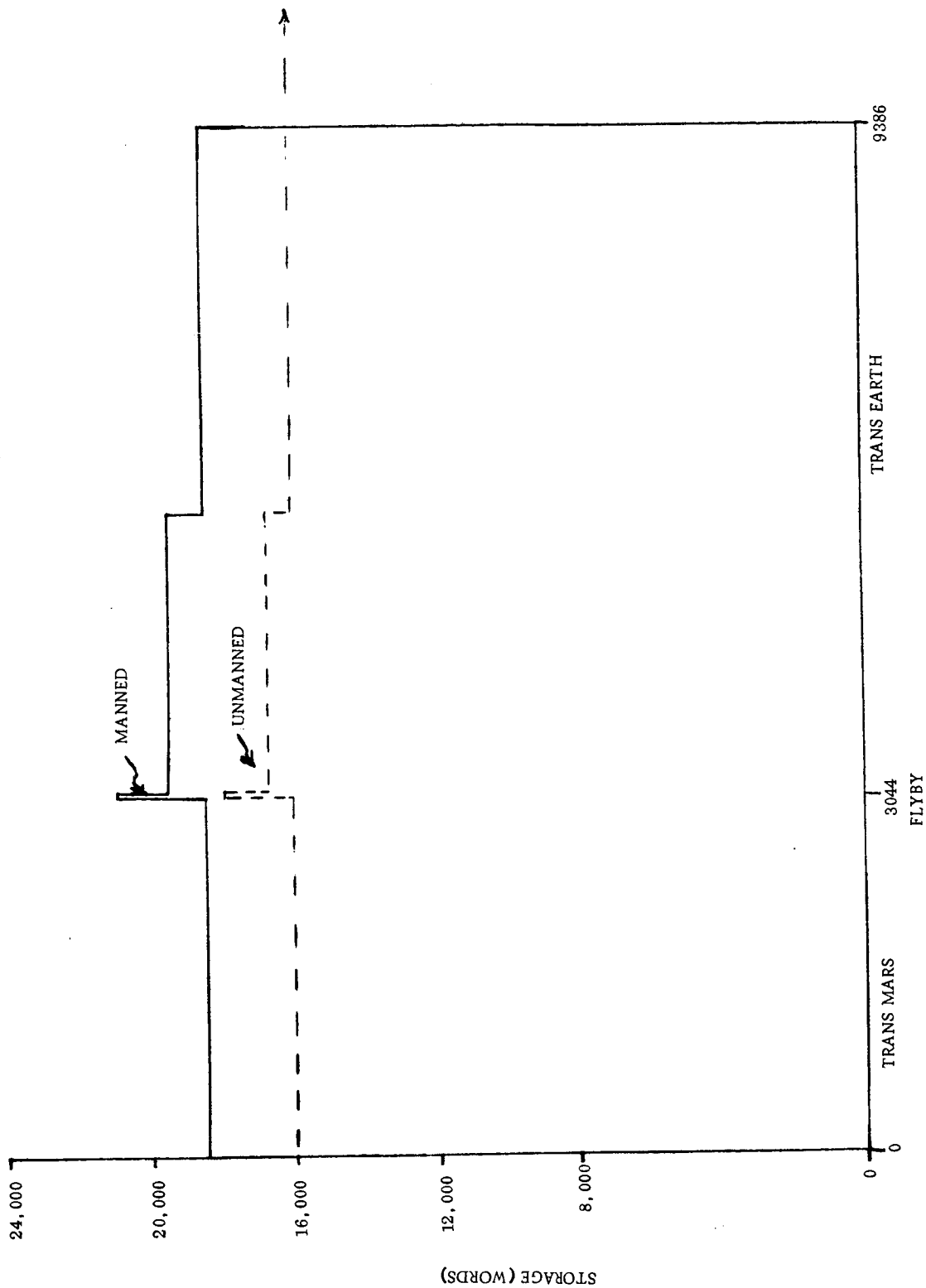


Figure 2-6. Interplanetary Flyby Mission Storage Requirements

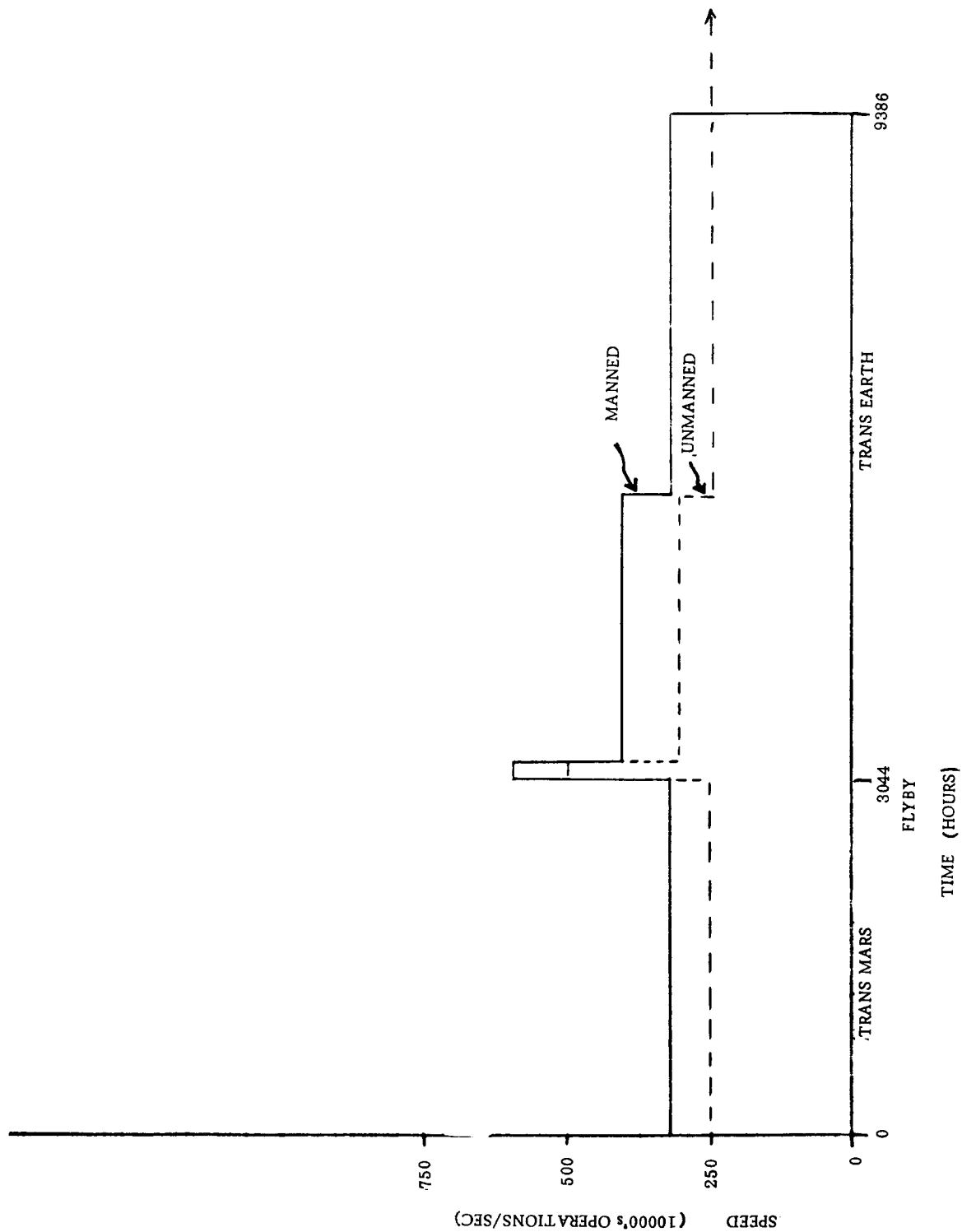


Figure 2-7. Interplanetary Flyby Mission Speed Requirements

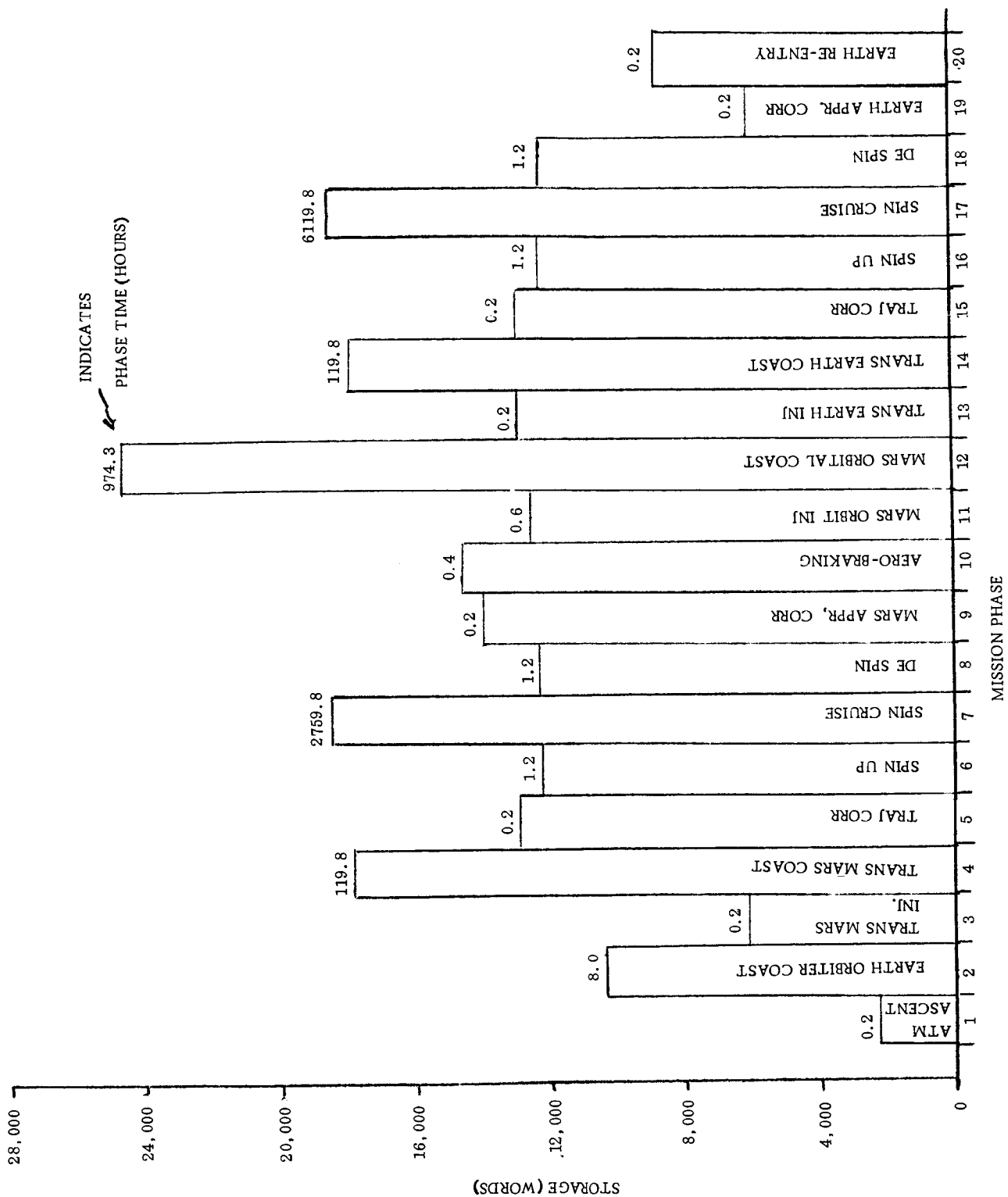


Figure 2.8 Computer Storage Requirements-Manned-Mars Lander Mission

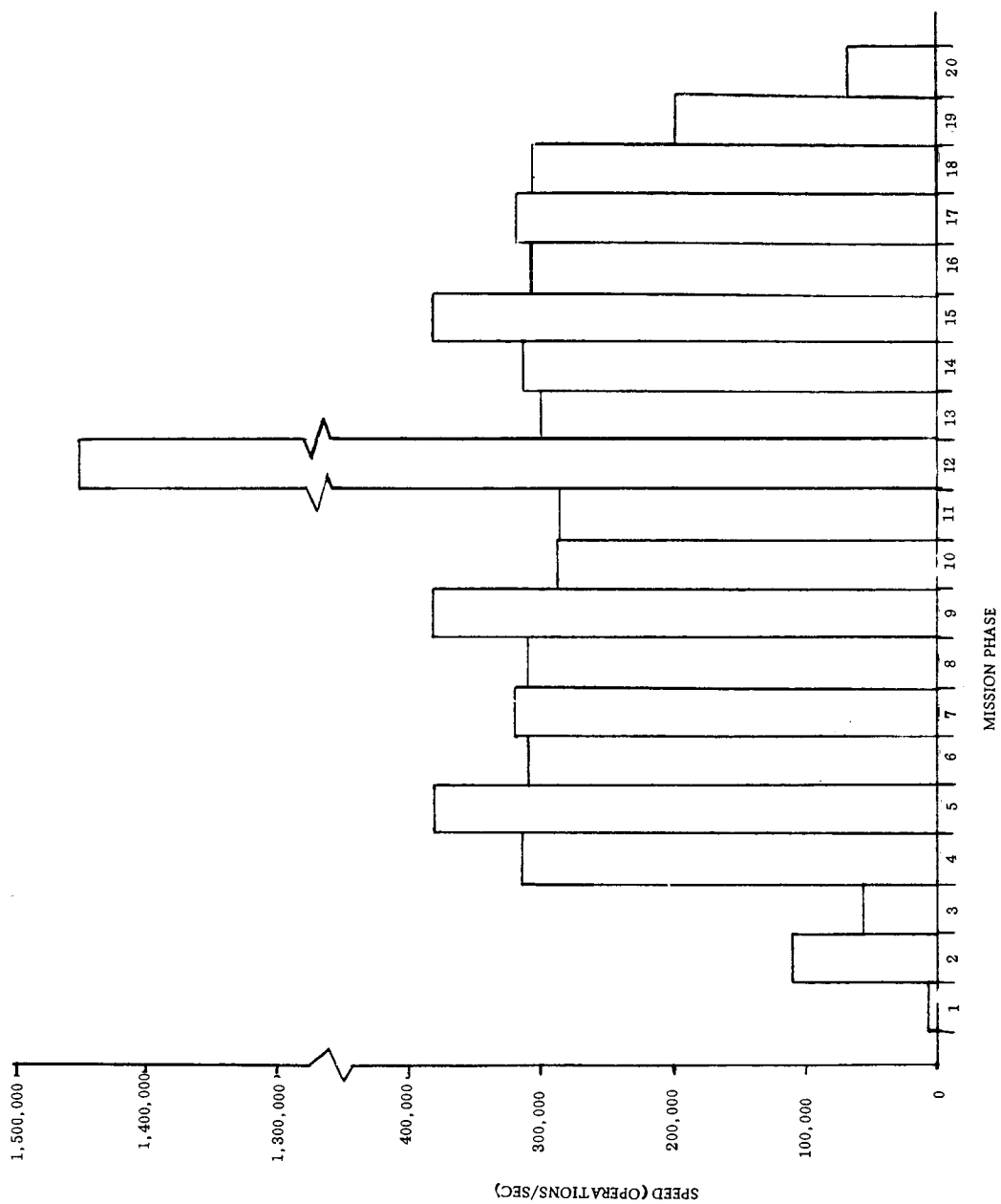


Figure 2-9. Computer Speed Requirements — Manned-Mars Lander Mission

3. TECHNOLOGY

The objective of this section is to indicate the means for mechanizing a novel machine-organization, the distributed processor, within the constraints that are part of the microelectronics revolution. The distributed processor will be introduced and developed in later sections of this report in addition to detailed design descriptions. A summary of this description is presented in this section to place the technology discussion in perspective.

The distributed processor is a computer organization made up of a number of cells, each cell fabricated by large-scale-integration (LSI) on a single substrate. A cell has both processing logic and memory and operates as a small computer. For example, a cell may contain a memory of 512 words of 18 bits each and a small parallel-by-bit processor having approximately 5,000 active devices. (See Figure 5-4). The cells are organized into groups, each group may contain on the order of 20 cells. For the requirements defined in Section 2, the computing system may contain about four groups as shown in Figure 5-5. These numbers represent preliminary estimates to describe the nature of the distributed processor and the numbers are subject to revision as further detailed design of the distributed processor is undertaken. However, this does establish design goals for the technology.

This section also describes current work on LSI, and some of the immediate research-and-development goals. From this base, trends are identified which can be extrapolated to predict the capabilities of technology for mechanizing aerospace computing hardware within the next ten years. This speculation is a necessary part of the dialogue between system-organization designers and hardware-fabricators in the pursuit of advanced systems. The conceptual approach must exploit the expected progress in the fabrication-technology without indulging in wishful-thinking. This section points out the anticipated developments that will make it possible to construct a distributed-processor-computer in the 1980's. Given the undeniable trend toward more complexity, as exemplified by LSI, the concept of the distributed processor was conceived and is being advanced as a candidate organization which effectively trades increased hardware complexity per wafer for an advantage in reliability, expandability, and power dissipation.

The following discussion of technology is presented in three parts: (1) A review is given of the contemporary technologies for LSI describing the state-of-the-art today; (2) The problems and goals of present R&D programs are outlined to anticipate the progress toward LSI within 2 years; and (3) A projection is made to predict the achievements in LSI within ten years. Special attention is given to those recognized problems most difficult to solve and needing particular attention to realize an effective solution.

3.1 LSI TODAY

There are two device-oriented technologies that lead the development in LSI today. Complete functions are being mechanized by integrated networks using; (1) bipolar transistors or (2) metal-oxide-semiconductor-field effect transistors (MOS-FET). Both of these competing approaches to LSI take advantage of the planar-fabrication process in bulk silicon wafers to produce monolithic structures. Each technology offers unique characteristics which can be exploited to achieve digital signal processing

These characteristics are summarized in Reference 9. For any given size of the active devices, bipolar transistors offer a basic speed advantage over MOS-FETs for two reasons: (1) Bipolar transistors can deliver charge faster to change the voltages across capacities that are associated with the interconnections in a large-scale-integrated-circuit; (2) The effective Signal-swing, ΔV , is smaller for the operation of a bipolar transistor. The speed advantage for bipolar over MOS transistors is about 10X (Reference 9). However, the process for making bipolar transistors is harder to control and the transistors are larger in area compared to fabrication of MOS-FETs. The MOS-FET approach has advantages of 2X in process simplicity and 5X less area compared to the fabrication of the same function using bipolar transistors (Reference 9). These two technologies do compete because system/circuit designers can scheme to trade increased complexity for speed to achieve an overall desired function. There are both advantages and disadvantages associated with complexity, but with the use of batch-fabrication, the cost of complexity is no longer the principle factor to be considered. In particular, aerospace computing hardware is not evaluated in terms of maximizing through (speed) or in terms of the number of computations per dollar. Instead, the characteristics of low power, high reliability, and small physical size are important factors. As attainable complexity increases, the advantages of MOS-FETs over bipolar transistors become increasingly important to the system/circuit designers.

Table 3-1 is a comparison of some results achieved in LSI by each of the two technologies as summarized in Reference 10. Also included, are entries describing integrated circuits manufactured by Autonetics which use p-channel, enhancement mode MOS-FETs in four-phase dynamic logic networks.

The basic feature of the distributed processor organization is the integration of memory with processing logic on a single substrate. Assuming that the storage and control for a single bit of memory requires about 10 devices, each cell of the processor would then require about 100,000 devices to mechanize ($N = 512 \times 18 \times 10 + 5,000 = 97,160$ devices). Using the data from Table 1, the required area and power dissipation can be estimated for a cell mechanized today by the two leading technologies. These estimates are presented in Table 3-2. The arrays chosen from Table 3-1 are somewhat representative of complex logic although the Honeywell Memory probably has a more regular and simple structure than the Autonetics DDA.

It is clear from the estimates in Figure 3-2 that stand-by power dissipation is the dominant limiting factor which makes the fabrication of 10^5 devices on a single substrate unfeasible today. It is also clear that at this level of complexity the advantages of MOS-FETs over bipolar transistors are impressive.

There are other batch-fabrication technologies being developed that might provide low-standby power and, therefore, should be considered as a means for mechanizing the proposed distributed-processor. Three of these are: (1) Cryoelectric thin film devices; (2) Optically coupled logic and memory devices and; (3) Heteroepitaxy of thin semiconductor films in which active devices are fabricated.

Cryoelectric technology has been promoted as a means for realizing very large memory subsystems in which the overhead-like expense and power consumption of the refrigeration equipment can be spread over a very large number of bits e.g., 10^9 bits. (See Reference 11). However, one of the principle motivations for developing the concept of the distributed processor is to improve computer reliability. The use of cryoelectric technology would require that the reliability of the refrigeration

Table 3-1. Integrated Equipment Components - 1966

Mfg.	Component	Device	Area in ²	No. of Devices	Devices Dens Device /in ²	Speed nsec	Power mW	Pads	Power Per Device
Autonetics	DDA	MOS	0.013	770	59,000	250 kHz	50	40	0.065 mw
TI	Series 53 Array	Bipolar	0.7	1,200	1,720	35	1200	60	1 mw
TI	8 Bit Shift Reg	Bipolar	0.006	160	26,500	15 mHz	190	6	1.2 mw
TI	Honeywell Memory	Bipolar	0.0071	100	14,000	25	250	14	2.5 mw
TI	Parallel Load Serial Shift	Bipolar	0.01	150	15,000	25	270	22	1.8 mw
GME	A-C 100-Bit Shift Reg	MOS	0.0065	613	94,500	1 mHz	200	12	0.32 mw
GI	D-C 21-Bit Shift Reg	MOS	0.0042	158	37,600	500 kHz	150	11	0.95 mw
TI	22-Bit Shift Reg	Bipolar	0.0126	350	27,800	3 mHz	35	8	0.1 mw
TI	Decoder	MOS	0.0057	152	26,700	200 kHz	25	26	0.16 mw

Table 3-2. Extrapolations to Distributed Processor

	Bipolar*	MOS**
Required Area for 10^5 devices	7.0 in ² @ 14,000/in ²	1.6 in ² @ 60,000/in ²
Power dissipation for 10^5 devices	250 watts -25 ns speed	6.5 watts -250 kHz clock
	Figure of merit	

*Honeywell Memory used as representative (See Table 1)

**Autonetics DDA used as representative (See Table 1)

equipment was much superior to that of the distributed-processor-cells, a claim which has never been suggested. The power consumption and size of such a system would also be very large.

Optical computing elements have been described in the literature; (Reference 12) and their use for inter-cell data-transmission has been considered. The use of optical coupling from radiating storage and logic devices to sensing devices should be faster than achievable by wired interconnection but the efficiency of the coupling, in terms of the energy required to transfer the data, probably will be less. The most complicating feature in the use of LSI with optical devices is the additional constraints of physical registration and alignment of devices to achieve the desired optical paths. The LSI technique of discretionary wiring, to enhance yield, is more difficult to use because the physical location of electrically tested circuits is no longer arbitrary. The development of materials, devices and LSI techniques for optoelectronic computing systems may lead to dramatic advances in computing performance. While the potential for such breakthroughs is acknowledged, the use of optoelectronic technology does not appear to offer predictable advantages for the proposed distributed-processor.

The growth of thin-semiconductor-films, in which active devices can be fabricated, is receiving considerable attention because of the potential for LSI. A particular example is the growth (Heteroepitaxy) of single-crystal silicon-on-sapphire (SOS) by chemical-vapor deposition. Since this technique was developed by Autonetics research chemists (Reference 13), Autonetics engineers have taken a particular interest in this new candidate for LSI. The basis of this interest is the possibility of creating large networks of devices having unusually small stray capacity, i.e., high speed potential. There are two reasons for the small stray capacity:

1. Interconnecting lines are made on top of the insulating sapphire surface. Conductor-to-substrate capacity is eliminated.
2. The P-N junctions for devices are formed perpendicular to the substrate. Therefore, the junction area is directly proportional to the thickness of the film which is typically one micron. Junction capacity is extremely small, a few femto-farads (femto = 10^{-15}).

Using SOS, three kinds of useful computing devices have been fabricated: diodes, p-channel MOS-FETs, and n-channel MOS-FETs. Combinations of all three have been fabricated successfully on the same substrate so that complementary devices can be used to realize memory and logic functions. Based on these developments, two LSI projects have been attempted at Autonetics:

1. Project 1 involves the design and fabrication of large diode-arrays. Perfect 96 x 70 arrays have been delivered to the Massachusetts Institute of Technology for a read-only memory application.
2. Project 2 involves the design and fabrication of large complementary flip-flop arrays for high-speed, very low-power read/write memory applications. Yield of the R-S flip-flops in a 22 x 24 flip-flop array is low at present, but the characteristics of working flip-flops are encouraging. The description of this memory array and some measured characteristics are summarized as follows:

Size: 17 devices per flip-flop (6 p-channel MOS-FETs, 6 n-channel MOS-FETs, and 5 diodes)
25 x 25 mils per flip-flop.
22 x 24 flip-flops per array
The array includes power distribution and read-out electronics.

Packing Density: 22,000 devices/in² on 1 in. dia sapphire substrate

Standby Power: 60 nanowatts per flip-flop at room temperature.
(512 x 18 flip flops would dissipate less than 1 milliwatt)

Speed: Read operation - 200 nanoseconds
Write operation - 300 nanoseconds

The development of the complementary SOS technology is important to the realization of the proposed distributed processor. The combination of high speed and very low standby power is unique to the complementary circuit and, with SOS, the mechanization is very attractive. The major cause for the low-yield at present is the critical nature of the doping-process step which converts the silicon to form the channels in half of the MOS-FETs. Experimental work is in progress on new process steps and device structures to simplify or eliminate this critical step in the fabrication of complementary integrated circuits. It should be noted that the dimensions of the individual devices on SOS are about three times larger than comparable dimensions of devices built in monolithic form. This is an indication of the current state of the development and does not constitute a fundamental limit due to photolithography or imperfections in the silicon film. Work on devices with smaller topologies is planned. Figure 3-1 is a photograph of the complementary flip-flop memory array.

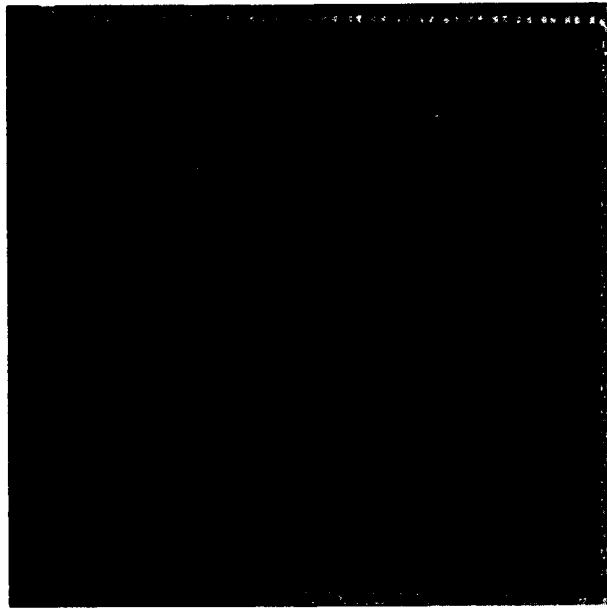


Figure 3-1 SOS Complementary MOS Memory Array

3.2 LSI IN TWO YEARS

Research and development programs are in progress to solve the recognized problems in LSI. This work can be divided into three categories:

1. System/circuit design through mask-set fabrication
2. Materials, devices, process improvements and packaging
3. Testing and customizing

This section of Section 3 will describe the nature of the problems under study and predict the state of the development in two years.

3.2.1 Research Category 1 - Computer Aided Design

When circuits are assembled from discrete components, it is possible to confirm the validity of a circuit-design or adjust parameters by building a bread-board model. A valid comparison could be made between actual performance and predicted performance. Two basic complications frustrate this approach in LSI. First, LSI makes it possible to build systems which were impractical to build with discrete components. The word, "large", in LSI really adds a distinctive new feature to the design problem. It must be recognized that LSI will produce complex circuits of non-linear elements which are many times more difficult to analyze, in terms of device models and equivalent circuits, than any previously analyzed circuits. This fact could be ignored if it were not for the second complication: the system/circuit designer cannot avoid the detailed selection of the topology of the integrated circuits. The relationships of device sizes and layouts to design-performance-goals could be used by systems/circuit designers for performance-optimization if the design process were augmented with machine-aids. However, the optimization of dynamic-performance through a man-machine interchange (e.g., conversational-mode computing) is a huge undertaking. LSI leads to smaller packages but to simulate the complex inner working will require the solution of many simultaneous non-linear differential equations which describe the network-dynamics. As a result, it can be seen that smallness in size does not make the system problems any easier to solve. In fact the problems are more difficult if only for the reason that integration excludes the flexibility afforded by the manipulation of separable subsystems.

Current R&D work on computer-aided design is directed toward the creation of new design tools for the automatic handling of design details as described above. The objectives are to permit the designer to exercise simulated alternative designs, to select optimum device characteristics and to create the tooling (optical masks) for the fabrication of systems using LSI. Further, computer-aided design development promises to relieve the designer of the greatly increased, tedious, book-keeping chores in which the human is most prone to error. In two years, these design tools should develop to permit the dynamic simulation of MOS-FET networks with about 100 nodes. The quality of the device-models will still be fairly crude and the man-machine interaction will still be painfully awkward, but estimation of dynamic performance and optimization of topology should be possible in 100 node networks. However, the automatic layout and generation of art work (masks) for LSI will not be possible within the next two years. Human pattern-recognition-skill will outperform machines for these tasks in the foreseeable future.

3.2.2 Research Category 2 - Materials, Devices, and Processes

Significant advantages have been attributed to complementary MOS-FET circuits on sapphire for the fabrication of the proposed distributed processor. The anticipated development of this technology, CMOS by SOS, is judged to be of special interest for the purposes of this report.

The Autonetics project on the CMOS by SOS memory array should produce a complete working memory module with about 10,000 devices integrated on a single substrate. This accomplishment will result from three developments:

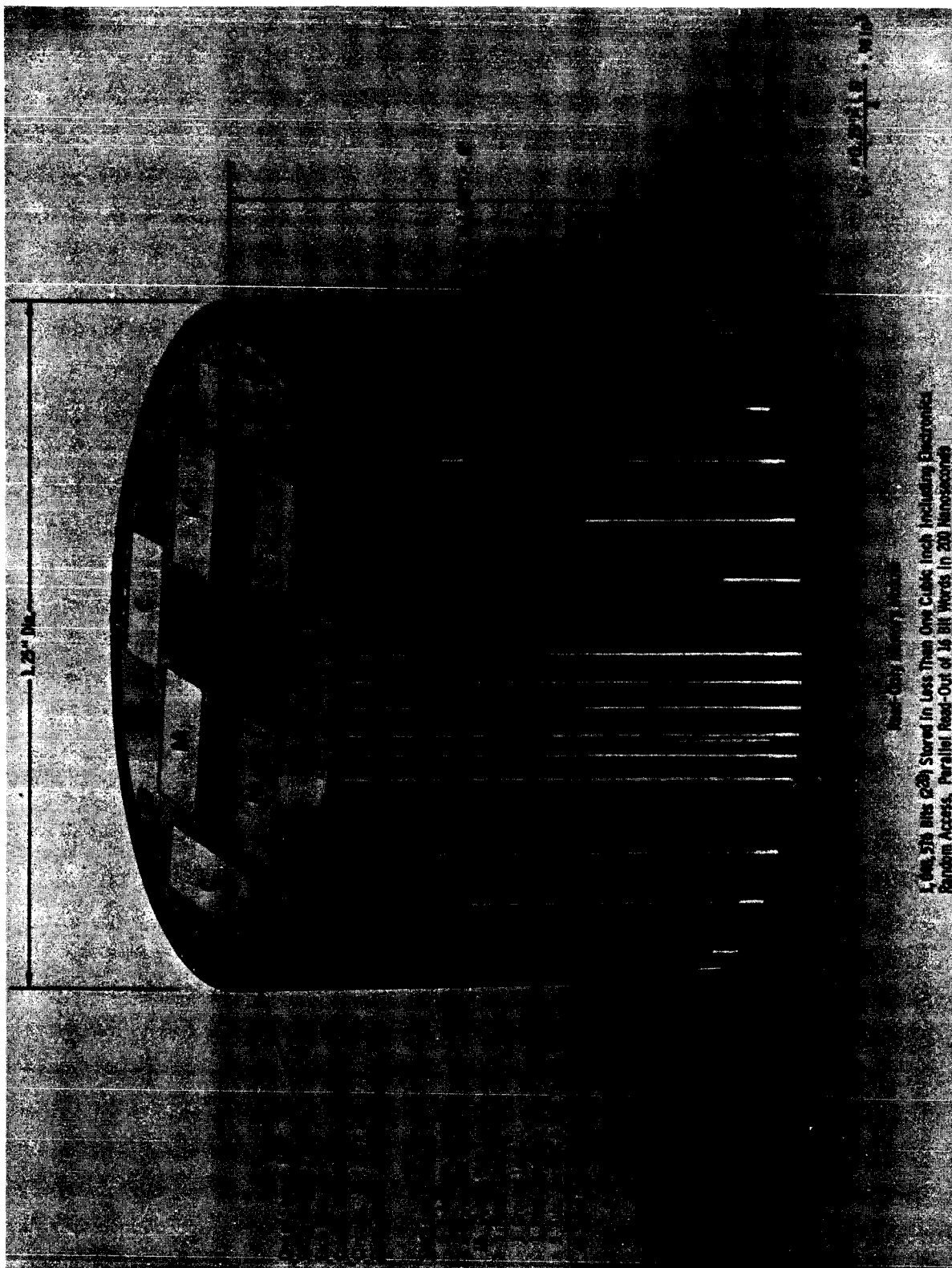
1. Improvements in the quality of the single-crystal silicon film which are grown on the sapphire will permit predictable and repeatable diffusion operations.
2. An alternative method will be found to replace the critical process step which is used now to convert (dope) the silicon when forming the channels in one-half of the complementary devices
3. Discretionary wiring will be used to enhance the yield by combining functionally-tested flip-flops into custom-wired sub-arrays, e.g., 16 x 16 working flip-flops from a total of 22 x 24 flip-flops.

It should be noted that CMOS by SOS is not a one-company development any longer. In particular, RCA is reporting significant progress under a USAF sponsored program in this same technology (Reference 14 and 17).

There are other process developments in LSI which will contribute to the eventual mechanization of the proposed distributed processor. For example, the experimental work at Texas Instruments on multilayer-interconnects and their pioneering work on automated discretionary wiring are of particular interest (Reference 10). Both of these developments are keys to the successful fabrication of very large, high-density arrays of devices such as those in the proposed distributed processor. Texas Instruments has made substantial investments in capital equipment, e.g., automatic CRT generation of masks and automatic laser-position-sensing plotting machines, to support their judgement on the potential of discretionary wiring and automation in LSI.

Packaging of CMOS by SOS wafers will be developed to increase the volume-density of devices to form more complex computing subsystems. For example, it would be possible to realize one cell of the proposed distributed processor in a cubic inch within two or three years by stacking 18 of the previously described Autonetics CMOS memory wafers with a processing wafer to form a physical structure resembling that shown in Figure 3-2. This packaging concept is under development at Autonetics for the Diode-Array Memory program. Interconnection from wafer to printed-circuits are shown as conventional ball-bonds but a better method should be developed. Two candidates are:

1. An adaptation of the beam-lead concept, developed by Bell-Telephone Laboratories
2. The combination of ultrasonic techniques with flip-wafer to execute simultaneous bonding of the metalization pattern on the wafer to external wiring patterns.



1,000,000 Bits (200) Stored in Less Than One Cubic Inch Including Encrypted
Random Access, Parallel Read-Out of 16 Bit Words in 200 Nanoseconds

Figure 3-2. Packaging Concept

3.2.3 Research Category 3 - Testing and Reliability

One of the most sobering aspects of LSI involves the matter of establishing the testing program and the criteria for acceptance of a processed wafer. It should be noted that a computer cannot be tested completely because it can exist in too many internal states to permit the complete exercise of the machine. The fact that a machine might exist on a single substrate does not change the astronomical number of tests that would be required for complete testing. In addition, integration tends to add a number of unplanned for internal states, characterized by unique worst-case combinations of regular internal states and signals, whose existence would be difficult to predict. Finally, the option to exercise separable subsystems is lost by the act of integration. Under these conditions, the question of what constitutes a failure-in-the-field is difficult to answer. It is apparent that a designer must prepare carefully to be able to define the test-for-acceptance at the end of the fabrication process. This preparation will involve three areas: 1. Simulation will assist in the search for worst-case conditions before array-designs are cast. 2. Discretionary wiring will help to restore the advantage of being able to perform functional tests on smaller sub systems on the wafer using external-to-the-wafer instrumentation. 3. Built-in-test-equipment will be included to provide integrated instrumentation to assist in the testing of large integrated systems.

In two years, a combination of special probing/testing machines and built-in-test-equipment will be used to inject test-signals at operating speed, to monitor the responses, and to perform the data-management required during acceptance-testing. In this same period the development trends in LSI technology will become clearer as experience with largeness is gained and new problems are recognized. These trends will further influence the machine-organization designer as he attempts to exploit the technology. For example, the ease of building shift-registers in integrated form might lead to conceptual approaches which trade memory capacity for access speed. Another strong possibility is to incorporate error-detection and error-correction coding into the hardware so that reliability may be improved using data-redundancy, as described in Reference 15. Reliability improvement through hardware redundancy has also been described in very great detail in Reference 16. The distributed processor is another candidate organization which suggests an approach to affect a trade of hardware for reliability. However, on the question of predicting the reliability of complex integrated arrays, very little is known. The claim that batch-fabricated interconnections on a wafer must improve matters is generally accepted; as already discussed, the distinction between production faults and in-the-field failures is blurring due to the difficulties in defining acceptance tests for the complex products of LSI. Nonetheless, it is possible to predict that integrating complete complex functions on a wafer, such as a cell from the distributed processor, will probably provide a good increase in system reliability due to a decreased number of interconnections from each wafer and a decreased number of packages.

3.3 LSI IN TEN YEARS

The feasibility of building the distributed processor, described in this report by 1978 requires the development of a number of disciplines in parallel; e.g., computer-aided design, materials, devices, processes, automatic testing and data-management.

The states of development for these LSI fundamentals have been described in previous parts of this section. In the following paragraphs of Section 3, a closer

examination is made of the distributed processor so that the progress required to achieve this advanced computing hardware can be appreciated. The question of feasibility in ten years can then be resolved into specific judgments concerning the reality of specific predictions.

Figure 3-3 is an illustration suggesting a gross layout using SOS for a cell from the proposed distributed processor. The memory section of this diagram includes provision for both data redundancy (double-error-detection, single-error-correction using 8 extra bits for the coding) and for discretionary wiring (within each memory subarray, 768 flip-flops exist from which 512 will be selected by discretionary wiring).

It is not the intention to specify a firm commitment to using data redundancy and in particular 8 extra bits for purposes of achieving higher yields. However, these numbers are used to illustrate a possible method to approach higher yields and will be used in the sample calculations for a cell.

The processor section includes sufficient area so that more than one copy of the processor can be fabricated. Discretionary wiring can then be used to choose a good processor and thus increase the wafer yield. The first obvious question concerns the density of devices required to integrate such a cell on a single substrate. The highest densities will be within the memory arrays. The number of devices that must fit into such a subarray is computed as follows:

$$N = \frac{10 \text{ devices}}{\text{flip-flop}} \times \frac{768 \text{ flip-flops}}{\text{subarray}} = \frac{7680 \text{ devices}}{\text{subarray}}$$

The allotted area on Figure 3-3 for each subarray is computed as:

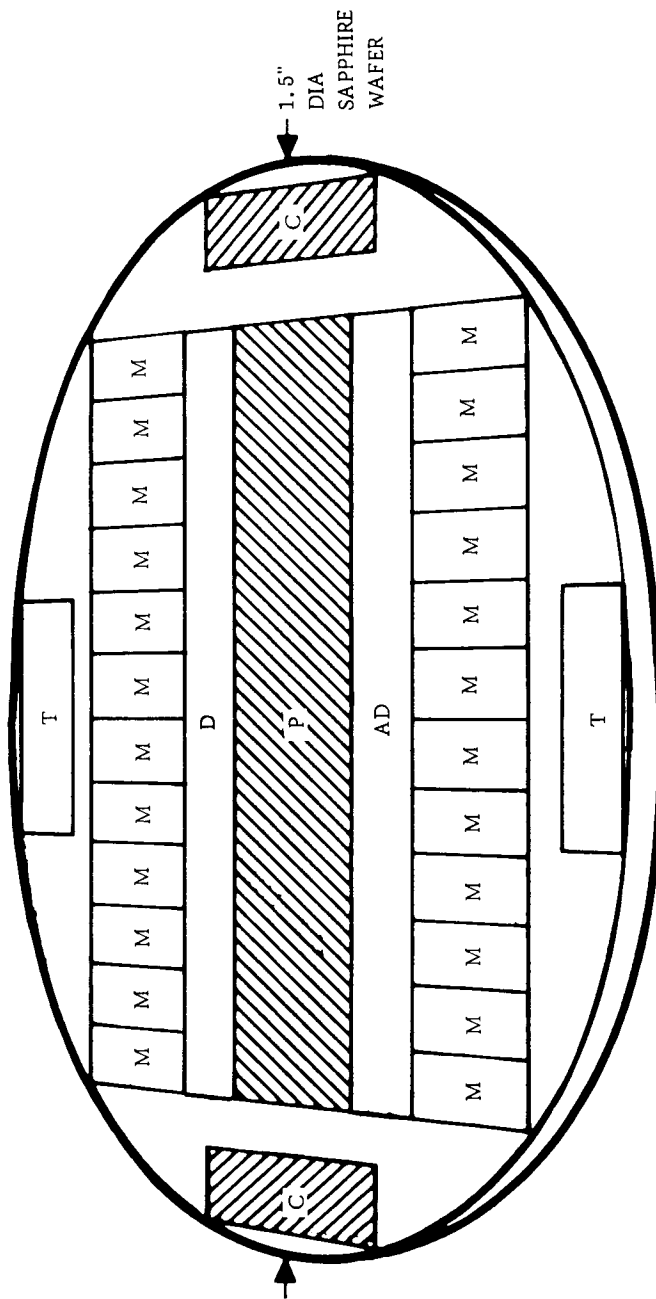
$$A \approx \left[\frac{1}{12} \cdot \frac{D}{\sqrt{2}} \right] \times \left[\frac{1}{3} \cdot \frac{D}{\sqrt{2}} \right] = \frac{D^2}{72} \text{ inches}^2$$

where D is the diameter of the sapphire wafer.

The growth of larger sapphire crystals can be anticipated to provide wafers which are 1.5 inches in diameter (compared to 1 inch maximum diameter now). Thus the required density within a memory subarray will be:

$$\text{Density} = \frac{7680 \text{ devices}}{2.25/72 \text{ inches}^2} \doteq 245,000 \text{ devices/in}^2$$

Compared to the present Autonetics CMOS memory project, this is an increase of about 10X in density. It was noted earlier that a factor of 3X decrease in linear dimensions could be expected within the CMOS by SOS array of static flip-flop without pushing the present state-of-the-art in photolithography. Therefore, it does appear feasible to achieve an increase of $(3X)^2$ which approximates the required factor of 10X to realize the densest part of the cell. If more memory capacity were desirable, it might be possible to trade access time and power to gain the higher density found in dynamic shift registers. This option is available to the machine-organization designers to increase the cell memory as the concept of the distributed processor is developed. (An explicit tradeoff of static versus dynamic shift register memories is not in order here; however, the shift register memory will be practical to build if its power dissipation can be sizably decreased from that of today).



- M - MEMORY SUBARRAYS 768 BITS EACH
- C - COMMUNICATIONS/CONTROL DRIVERS
- AD - MEMORY ADDRESS DECODERS
- P - PROCESSORS
- T - TESTING

Figure 3-3. Layout of Distributed Processor

The second basic question concerns the power dissipation of the static memory. Total power dissipation is composed of two parts: (1) standby power, and (2) dynamic power. The basic advantage of CMOS by SOS is that low standby power can be achieved simultaneously with high speed performance within the memory subarrays. This standby power is computed using measured data from Autonetics CMOS static Flip-flops as follows:

$$P_s = 24 \text{ subarrays} \times \frac{512 \text{ Flip-Flops}}{\text{Subarray}} \times \frac{60 \text{ nanowatts}}{\text{Flip-Flop}}$$

$$= 740 \text{ microwatts per cell at room temperature}$$

The standby power for an entire distributed processor should be less than 0.1 watts. (1mw/cell x 20 cells/group x 4 groups < 0.1 w). Recalling that the duty cycle on memory cells is very low, the basic advantage of CMOS by SOS is impressive.

Dynamic power, consumed by charging and discharging the nodes of the network during operation, is more difficult to estimate. The dynamic power associated with any single node of the cell would be computed as:

$$P_D (1 \text{ node}) = C_N V^2 f \text{ watts}$$

Assuming an average of 10,000 active nodes in the whole distributed processor, each having an average of 5 pf capacity, a supply voltage of 10 volts, and an operating frequency of 2 MHz (easily obtainable using CMOS), the dynamic power is:

$$P_D = (10^4) \times (5 \times 10^{-12}) \times (10^2) \times (2 \times 10^6) = 10 \text{ watt}$$

The dynamic power completely dominates the standby power at room temperature. The total operating power, 10 watts, is divided among some 80 cell wafers so that there does not appear to be a power dissipation (heating) problem using CMOS by SOS.

The capacity to produce, in 1978, CMOS by SOS integrated cells of about 200,000 devices for the processor including redundant processor arrays, is more a question of management and company goals than of technical limitations. Progress in materials and processing techniques, e.g., ion-implantations, multiple heteroepitaxy, multilayer interconnects, must be matched by progress in the following contributing fields within the next ten years:

1. Computer aided design including advanced I/O equipments should permit the man and machine to function in the conversational mode. Dynamic simulation of an entire memory subarray (7680 devices) or the processor (5,000 devices) must be possible. The simulation programs should permit the detailed selection of topology of the networks including device dimensions, physical location, and interconnection patterns to optimize performance.
2. The use of computers will be extended to the automatic production of the tooling i.e., the layout and fabrication of the photo-mask-sets. At the level of complexity represented by a cell from the proposed distributed processor, the quantity of detail in such a mask set is too great for human generation.

3. The use of computers will be extended to the control of the acceptance testing including dynamic functional testing and data management followed by the automatic generation of masks for discretionary wiring. Retest of the custom-wired packaged cells must also be under computer control. The access to various subsystems on the wafer will make particular use of multilayered wiring and built-in test signal generators.

The importance of all these parallel developments in LSI is appreciated by the management throughout the industry as evidenced by their direction of R&D funds. Therefore, it is predicted that the technology will exist in 10 years to build the distributed processor

4. PARALLELISM

4.1 INTRODUCTION

This section of the report covers parallelism within the computations. Definitions of parallelism will be given and a discussion of the methods and procedures used to evaluate parallelism within a set of computations will be presented. Parallelism will be investigated in order to provide a basis for evaluating various parallel or distributed computer organizations. In particular, the amount of parallelism that may be utilized will be evaluated and the methods of making use of parallelism or assigning parallel operations shall be investigated. Finally, results derived from the application of parallelism studies to the spaceborne computational problem shall be presented.

Two types of parallelism, applied and natural, may be defined as follows: Applied parallelism: The property of a set of computations that enables a number of groups of identical operations within the set to be processed simultaneously on distinct or the same data bases. Natural parallelism: The property of a set of computations that enables a number of groups of operations within the set to be processed simultaneously and independently on distinct or the same data bases.

It may be noted from these definitions that applied parallelism is a special case of natural parallelism, since the naturally parallel operations could be groups of identical operations. The distinction between the two types is introduced since it has important implications with regards to computer organization. The two types of parallelism will be illustrated with the simple example in Figure 4-1.

The example is the computation of the expression $a/x + b/x + cy = Z$. Figure 4-1 illustrates how the expression would be computed in a sequential manner. The numbers above each circle indicate the time that might be required to compute each term on a sequential computer (S). If applied parallelism were capable of being taken advantage of, the computation on such a machine would take place as illustrated in Figure 4-2. The term A/S in this figure is the ratio of the time required on the machine with the capability for executing applied parallelism to the time required on the sequential

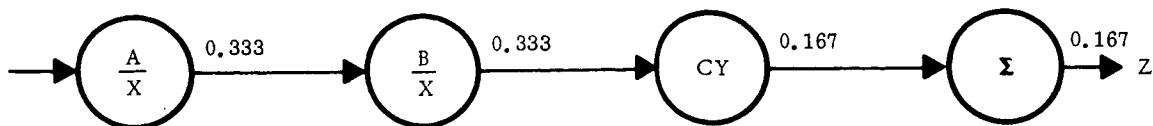


Figure 4-1. Sequential Steps in Computation

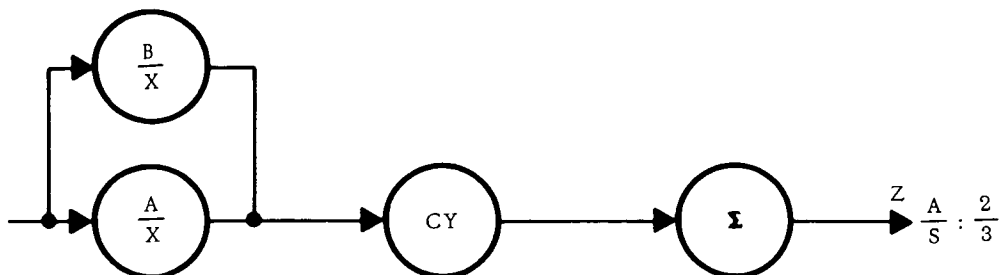


Figure 4-2. Applied Parallelism in the Computation

machine. It should also be noted that a degree of applied parallelism of 2 is utilized in Figure 4-2 (this occurred during the parallel computation of a/x and b/x).

If the capability for taking advantage of natural parallelism is now introduced, then the computation may take the form illustrated in Figure 4-3. It should be noted that the term 'cy' may now be computed in parallel with a/x and b/x . The total computation illustrated by Figure 4-3 may be classified as utilizing natural parallelism; however, it should be noted that a subset of this may actually be classified as applied parallelism as indicated by the dashed lines. Therefore, the computation may be said to consist of applied and natural parallelism; this combination may be referred to as total parallelism. Utilization of total parallelism results in the reduction ratio T/S in Figure 4-3. With this introduction to the notion of parallelism, some of the problems in attempting to analyze a set of computations to determine the utilization of parallelism will be given next.

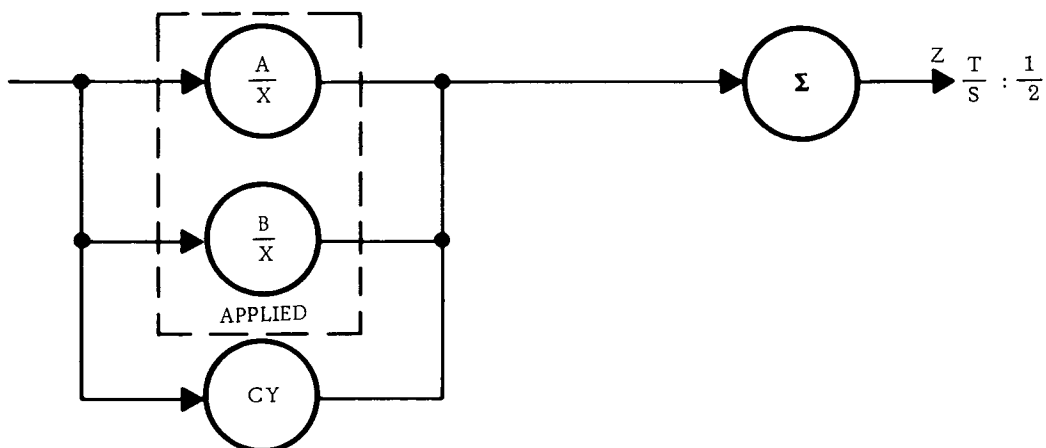


Figure 4-3. Applied and Natural Parallelism in the Computation

#2

4.2 ASSIGNMENT AND SEQUENCING PARALLEL OPERATIONS

It can be seen from the above simple example that an important part of any parallelism analysis is the formation of computation graphs to study the degree of parallelism utilized and the inherent reductions in computation time. Unfortunately the graphs are quite complex to construct for problems of practical interest. It will be shown below that the problems encountered in forming the graphs are analogous to those studied in assembly line and job shop scheduling theory. This particular topic will be treated first before discussing the overall problem of analyzing parallelism in the spaceborne computational problem.

An important consideration in the analysis of parallelism is the degree of parallelism utilized and the computation time reduction or "effectiveness of the parallelism". In determining this effectiveness of parallelism one approach that may be used is to assume a certain degree of parallelism and then determine the minimum computation time by re-ordering the computation graph to make use of the parallelism. Following this procedure a curve of the degree of parallelism vs. computation reduction ratio is obtained. It should be noted that this problem can be approached another way to arrive at the same curve; this alternate approach is to assume a certain computation reduction ratio and solve for the minimum degree of parallelism required to achieve this reduction ratio.

These two approaches of obtaining the "parallelism" curve are analogous to problems proposed and studied in the area of operations research. The first approach, determination of minimum computation time given a degree of parallelism, is analogous to the problem encountered in job shop scheduling or assembly line balancing, namely: given m jobs or tasks to be performed by n men or assembly stations, determine the sequence or assignment of the jobs or tasks that completes them in the shortest time.

The second approach is analogous to another job shop scheduling or assembly line balancing problem, namely: given m jobs to be performed, the total time spent by any one man or at any one assembly station is not to exceed T , what is the sequence or ordering of the jobs to minimize n the number of men or assembly stations required?

Considerable research has been carried out in this field of operations research and several references were reviewed to determine if any of it could be extended to the parallelism investigation carried out in this study; some significant points from the references are discussed below. An extensive list of references was found in a fairly recent report: "The Automatic Assignment and Sequencing of Computations on Parallel Processor Systems" by D. F. Martin at UCLA (Reference 18).

The following problem is considered in Reference 19: n jobs with partial or total ordering restrictions are to be performed, if all jobs must be completed by time T arrange a schedule that completes them with the minimum number of men, or if n men are available, arrange a schedule that completes all jobs at the earliest time. It is assumed that all jobs require equal time to complete by any man and that any man can do any job with the capability of immediately starting another after finishing one.

Under these assumptions a lower bound for the number of men is found. A brief description of how this lower bound can be found will be given here.

If

$$y < 1 / (y^* + c) \sum_{j=1}^{j=y^*} p(\alpha + 1 - j) \quad (1)$$

is true then it is impossible to complete all jobs with y men in $\alpha + c$ units of time.

Where

y = number of men

$p(\alpha + 1 - j)$ = number of nodes on the graph with $\alpha_i = \alpha + 1 - j$

α_i = $X_i + 1$, where X_i is the length of the longest path from the node N_i to the final node in the graph (it is assumed that each branch takes an equal unit of time, 1, to complete)

α = maximum X_i over the entire graph

c = a non-negative integer

y^* = a positive integer such that

$$\max_y \left[1 / (y + c) \sum_{j=1}^{j=y} p(\alpha + 1 - j) \right] = 1 / (y^* + c) \sum_{j=1}^{j=y^*} p(\alpha + 1 - j) \quad (2)$$

Basically, the lower bound (y) is found by: labelling all the nodes of the graph with their appropriate $p(\alpha_i)$, determining c from the allotted time, T , to complete the tasks (c corresponds to a time interval that may be added on to the shortest possible time to complete the tasks, thereby giving the allotted time, T , to complete the tasks, c units of time greater than the minimum), and then determining y^* for Equation 1 by solving Equation 2.

Unfortunately in using this algorithm to find the upper bound on the minimum number of men, the ordering restriction that the graph be a tree* must be imposed. With this restriction the minimum is given by m , where m is the integer satisfying:

$$m - 1 < 1 / (y^* + c) \sum_{j=1}^{j=y^*} p(\alpha + 1 - j) \leq m \quad (3)$$

and where the same definitions apply.

It is also proven that for these assumptions a very simple algorithm may be applied to determine the sequencing procedure for the jobs. This algorithm states that

*A tree may be defined as the ordering restriction that the directed graph has no nodes which contain more than one successor node.

if the number of jobs that may be started at any point in the graph is greater than the number of men available, then the jobs assigned are those with the longest path through the graph, in case of a tie the choice is arbitrary. This method is simply that of starting the longest job first or as soon as possible.

The restrictions that each job requires an equal amount of time and that the ordering of the graph be a tree are severe in terms of attempting to extend this work to parallelism studies of computations. It may be possible to reduce computational graphs to the micro-operation level where each job takes an equal amount of time; however, for problems of reasonable complexity, this may become a monumental task. The restriction of a tree limits the type of graphs that may be analyzed; unfortunately most of the computations involved here do not form trees (This is because the intermediate results are often used at many points within a computation).

Nevertheless this work has been pointed out here since it represents the only pure analytical approach found in the literature to solving the assignment and sequencing problem. If the first restriction namely equal lengths of time can be reckoned with, it does offer a lower bound for any computational graph.

The following problem is investigated in Reference 20: given n jobs, the sum of the execution time at any one station is not to exceed some T , what is the ordering of the jobs to minimize the number of stations? The approach taken for solution here is by means of dynamic programming. A recursion type approach is proposed. Basically what is proposed is to vary the assignment of the last job that is executed and determine the assignment over all feasible cases that results in the minimum cost (cost here is the number of execution stations). Then recursively the remaining assignments are determined to minimize the total cost. This method will yield an optimum solution. However, for all but modest size problems the program will become extremely lengthy. For large problems approximate solutions are proposed by testing subsets of all possible sequences thereby yielding a suboptimal solution.

An apriori assignment and sequencing approach to assigning jobs on a computer in order to minimize overall computation time is given in Reference 21. The problem is stated for a set of autonomous processing units where the jobs take different times depending on which unit it is assigned to; the situation where a job may be assigned to only one particular unit is considered here however. The graph of the computation flow is used to derive a "precedence" matrix which is then used to derive an "assignment" matrix. This matrix contains information as to what must precede each task, its time to complete, and also the number of tasks that must follow it (called the precedence number).

Several rules are defined for assigning the tasks; the objective being to minimize the overall computation time. One rule given for assigning tasks in the case where more than one task can be assigned to a given processor is to assign the task with the highest precedence number. Another case is considered for this situation where the precedence numbers are the same; a very simple situation is examined and some special rules are stated. It is noted that when more tasks and processors are involved, the solution becomes very complex and in fact it is stated that no general solution exists for the general case of n tasks and m processors.

Evaluation of these references and others show that there are no general solutions to the optimum sequencing problem that are applicable to the study of

parallelism here. The only solutions to the problem are generally very lengthy computational programs considering all permutations of the solution space to find a minimum; analytical procedures are non-existent for general solutions. It should also be mentioned here that there is another field in the assignment of computations, this is the scheduling problem where jobs are scheduled on one or more processors to achieve a minimum cost. What is generally assumed here is that there is a cost function, $c(t)$, associated with each job and a schedule is to be determined which will minimize the total cost. The methods used here are generally applicable to and used for time sharing or multiprogramming applications.

Some work has been done on several specific mathematical algorithms in Reference 22 to determine parallelism and computation reduction ratio. The type of algorithms investigated were matrix multiply, solving ordinary differential equations, etc. This work may be useful in analyzing the problems here when these types of specific mathematical operations are encountered in the computations to be analyzed.

4.3 APPLICATION OF PARALLELISM STUDIES

The foregoing has presented some general concepts of parallelism and their relationships to the problems involved in job shop scheduling and assembly line balancing. In this section the application of parallelism studies to the spaceborne computational problem shall be investigated. The theoretical approach to the application of parallelism studies shall first be presented; this approach is too complex to apply to the problem at hand. Therefore, at the end of this section an approach that will be used with simplifying assumptions shall be presented.

Two types of parallelism shall be considered: applied and natural. The computations as executed on a sequential computer for the spaceborne computational problem are shown in Figure 4-4. These computations are shown as a sequential solution of tasks 1 through n . The computations are cyclic, in that when task n is completed, task 1 is once again initiated. These tasks are subtasks of the four major computational tasks, namely:

1. Navigation and Guidance
2. Telecommunications
3. Scientific Experiments
4. Status Monitoring and Checkout

The subtasks may be items such as "compute body to inertial transformation matrix," "compress video experiment data," etc. These subtasks are either periodic in that they require a certain repetition rate or they are asynchronous in that they are required upon request, as background, etc. As might be expected the four major tasks consist of a variety of subtasks in terms of size and repetition rates of solution. Some of the tasks shown in Figure 4-4 are identical due to the required periodicity of certain tasks for example task 4 may be required to be computed once per second and assuming task 1 thru n are cycled thru once per second then task 4 is unique and represented by only one task in this figure. Now assume task 5 is required to be computed twice per second, then this task would appear twice in Figure 4-4; for example as task 5 and say task n .

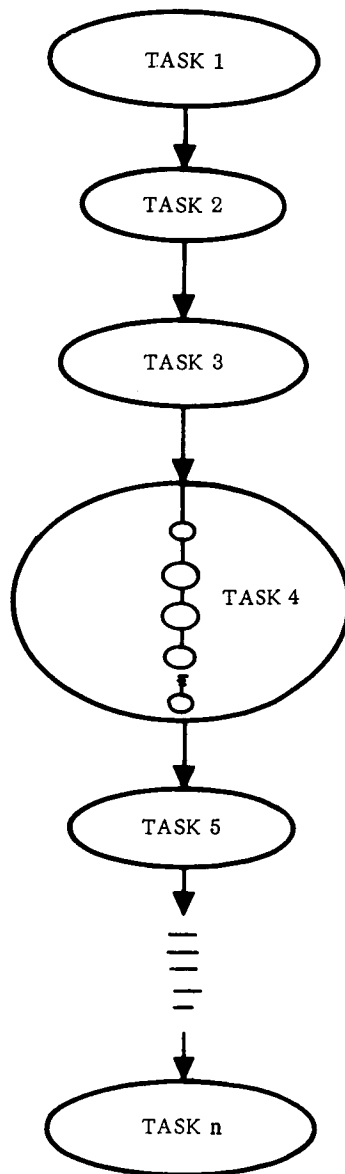


Figure 4-4. Sequential Solution of Computational Problem

4.3.1 Applied Parallelism

If the computational facility has the capability for execution of applied parallelism, the execution of the set of tasks shown in Figure 4-4 may now be depicted as shown in Figure 4-5. Referring back to Figure 4-4, the sequential solution of a task is depicted by the sequential set of steps shown in task 4. The application of applied parallelism is shown in Figure 4-5 by the formation of an applied parallel graph shown in the solution of tasks 1 and 2. As inferred by the definition of applied parallelism, the vertices of these graphs that are executed simultaneously (indicated by vertices lying in the same horizontal plane), consist of an identical computational step (e.g. add, transfer, etc. is the level considered here). The degree of parallelism utilized varies as a function of time or vertical position in each graph. The degree of parallelism is given by the number of vertices in a particular horizontal plane.

The considerations of parallelism in Figure 4-5 are within tasks only. It is possible to consider applied parallelism between tasks also. For example, the graphs for tasks 1 and 2 in Figure 4-5 may have several horizontal planes that execute the same applied parallel operations and as a result may be combined horizontally; this combining cannot be carried out if precedence relationships exist, however. Likewise many of the graphs for the remaining tasks may have horizontal planes that could also be combined. The application of applied parallelism to be considered in this study will be within a task only or between tasks only when the tasks have identical graphs so that they may be combined directly. This limitation is introduced since the consideration of combining applied parallelism between tasks and the timing of the remaining non-

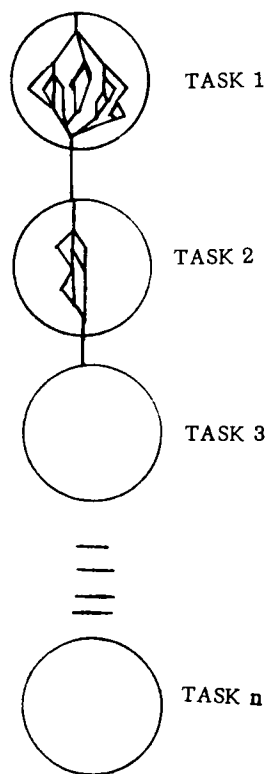


Figure 4-5. Utilization of Applied Parallelism in the Computational Problem

applied computations between these tasks becomes a very complex problem. Indeed it may not be practical to program the tasks in such a manner since executive costs in attempting to schedule tasks to take advantage of all the applied parallelism between tasks may prove extremely high. Some consideration will be given to this problem when software studies are conducted on the distributed organization later in the report.

The computational tasks will be studied as described above and a curve will be derived indicating the degree of applied parallelism vs. the computation reduction ratio. The computational reduction ratio is basically the speed advantage gained by using applied parallelism or how much faster the problem will run. It is determined by dividing the time it takes to execute the problem on a sequential machine (Figure 4-4) by the time it takes on a machine with applied parallelism (Figure 4-5). The curve is obtained by varying the degree of applied parallelism available when constructing the solution as shown in Figure 4-5. This curve gives a measure of the effectiveness or efficiency of applied parallelism, e.g., if a reduction ratio of α were obtained with a degree of parallelism of α , then one could say the effectiveness was 100 percent or the applied parallelism were utilized all throughout the solution of the problem.

4.3.2 Natural Parallelism

The next step in investigating parallelism within the spaceborne computational problem is to consider natural parallelism. If natural parallelism is assumed available on the computational facility, the solution of the computations may take the form shown in Figure 4-6. In this figure the tasks have utilized applied parallelism as in

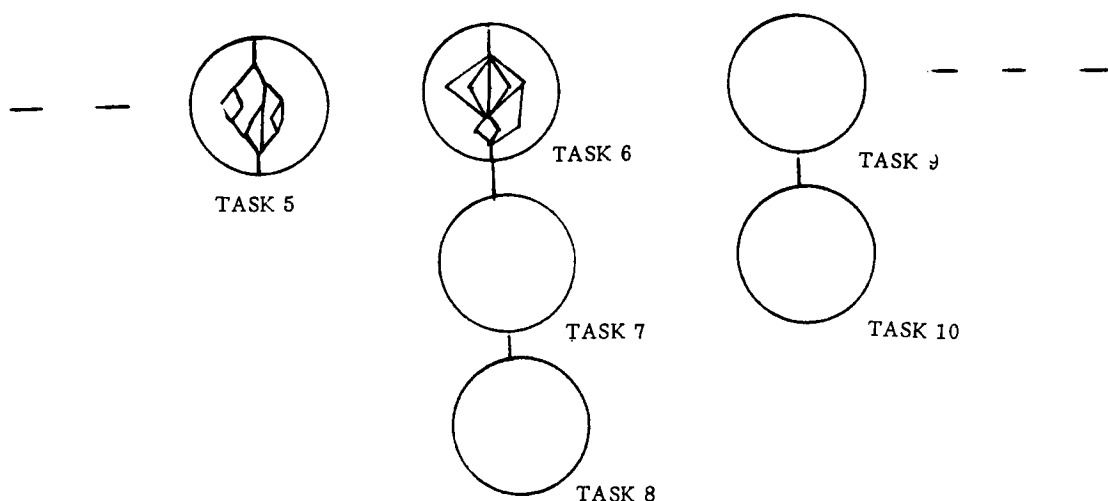


Figure 4-6. Utilization of Applied and Natural Parallelism in the Computational Problem

Figure 4-5 and then natural parallelism when possible. This utilization of natural parallelism occurs both within a task and between tasks. Consider task 6 for example some of the graph of this task may exhibit applied parallelism and some of it natural parallelism. It is seen in Figure 4-6 that task 5 may be solved in parallel with tasks 6, 7, and 8 and also tasks 9 and 10 and all the other tasks indicated by the dashed lines. However, note that tasks 6, 7, and 8 contain certain precedence relationships and must be solved in the sequence as indicated. Solving the tasks in this manner, one can obtain the minimum computation time or maximum computation reduction ratio by finding the tasks or set of tasks which take the longest to compute in Figure 4-6, for example say tasks 9 and 10. Of course this assumes that whatever degree of parallelism that is required to achieve this is available. The problem that one is interested in however is to find this minimum computation time using the minimum degree of parallelism to achieve it. This then requires assigning and sequencing the computations in an optimum manner, it should be noted that this must be done within a task and also between a task, as might be expected this is not an easy problem to solve due to the large amount of permutations involved.

As an example consider the situation depicted in Figure 4-7. In task n is shown the degree of applied and natural parallelism required as a function of time through this task; the remainder of the tasks also have similar graphs associated with them. It should be remembered that these graphs of the degree of parallelism are obtained by an optimum assignment and sequencing analysis within each of the tasks. The problem of optimum assignment and sequencing between the tasks involves arranging the tasks both vertically and horizontally while observing precedence relationships

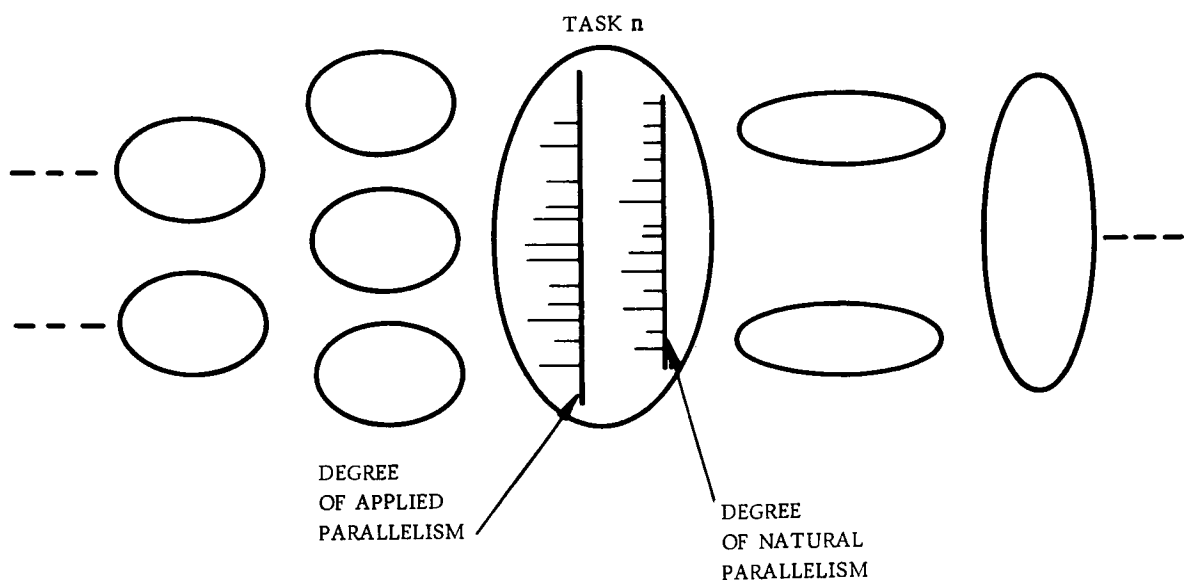


Figure 4-7. Assignment and Sequencing of Tasks in the Applied and Natural Parallelism

between certain tasks. As may be expected this involves many permutations. To further complicate the problem it is also desired to utilize as much applied parallelism as possible. This requires trying to match up identical applied operations between tasks where possible and also trying to sequence the tasks so that as much applied is being utilized at any one time as is possible. The total problem has two curves similar to that given for task n and it is desired to have as much applied parallelism as is possible utilized and the minimum amount of natural parallelism utilized to achieve the maximum computation reduction ratio. (It should be noted that the natural parallelism curve also includes the applied parallel curve by definition.)

In general the final result desired is to consider the degree of parallelism required throughout the total problem, as shown in Figure 4-8, and minimize the maximum point on this graph. Each permutation in assigning and sequencing the tasks will result in a new graph in Figure 4-8 and the objective is to determine the optimum permutation.

4.3.3 Limiting the Amount of Parallelism

The above discussion was centered about determining the minimum computation time or maximum reduction ratio. As noted above some task or set of tasks will require the longest to compute, for example 9 and 10, in Figure 4-6. Then with the above objective as a goal the problem involved a horizontal and vertical permutation around these longest tasks. The next consideration involved in investigating parallelism is to limit the degree of parallelism (lower than that degree obtained above) available

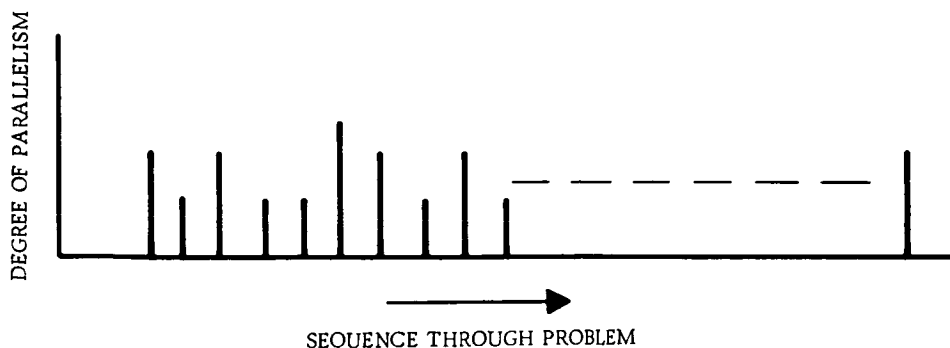


Figure 4-8. Degree of Parallelism Utilized

and determine the maximum computation reduction ratio with this degree. The procedure now is the same as above, however, since there is now a lower degree of parallelism considered, additional tasks must be vertically combined with tasks 9 and 10 (the longest tasks). Essentially as the degree of parallelism is reduced the graph of the set of computations as shown in Figure 4-6 will be stretched out vertically and reduced horizontally. It should also be noted now as the degree of parallelism is reduced the permutations involved in determining the optimum computation reduction ratio become more complex. This is due to the fact that not only must the permutations between the tasks horizontally and vertically be considered but also the permutations within each task must be considered at the same time.

The result of the above analysis would be a curve as shown in Figure 4-9. This curve shows the degree of parallelism required vs. the computation reduction ratio. The curve shows the computation reduction ratio from 1 with a degree of parallelism of 1 (equivalent to solution with a single sequential computer as in Figure 4-1) to the maximum reduction ratio and the minimum degree of parallelism required to achieve it.

4.3.4 Computer Organization Relations

Relating the above effort to computer organization, it is possible to think of computational cells executing the above parallel operations where one cell is required for each degree of parallelism. The computation reduction ratio essentially dictates at what speed the cells must be capable of operating. (Details of organizations such as this is given in Section 5.) The above, however, assumes as much storage available

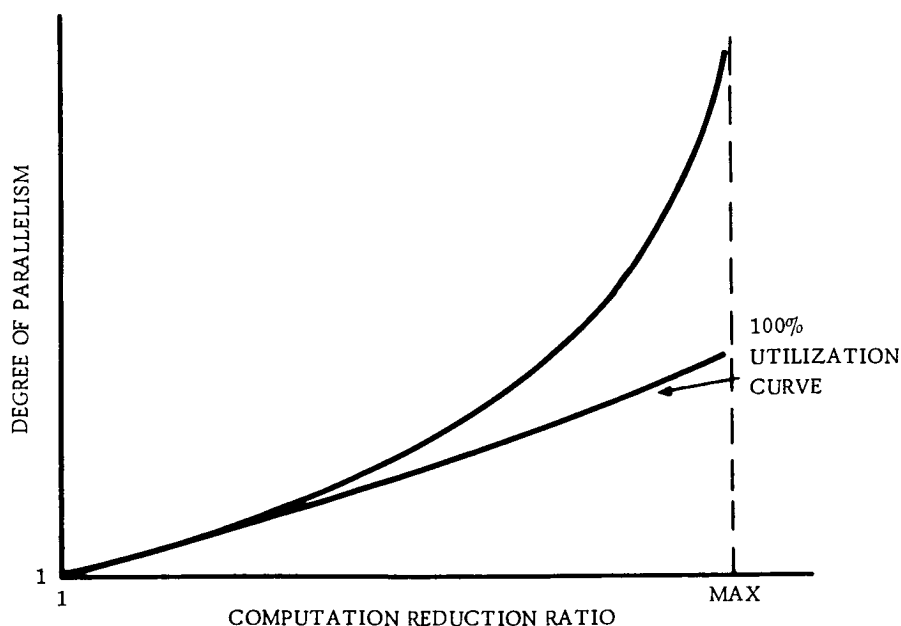


Figure 4-9. Degree of Parallelism vs. Computation Reduction Ratio

per cell as required. After having gone through the above analysis it is possible to consider the programming of the tasks on the cells and a curve of storage required per cell as a function of the degree of parallelism (or number of cells) would be obtained as shown in Figure 4-10. Given the curves in Figures 9 and 10, it is possible to select a degree of parallelism (or number of cells) and determine the required speed and storage capability of each cell.

It is possible to group together tasks or portions of the total computational problem as shown in Figure 4-11; this figure is identical to Figure 4-4 except that tasks are now assigned as parts of groups as shown. The purpose of introducing this concept of groups is that it offers the possibility of making use of more applied parallelism; for example, if two groups were used two different types of applied parallel computation could be executed simultaneously if one considers the two groups as assigned to two independent computation facilities. In considering groups, one will obtain a set of curves for degree of parallelism vs. storage required per cell as shown in Figure 4-12. This figure is identical to Figure 4-10 except a new parameter, the number of groups, has been added. The degree of parallelism in Figure 4-12 refers to the total parallelism (or total number of cells) required within the groups (e. g. 2 groups with a degree of 10 each result in a total parallelism of 20). Only an equal amount of parallelism for each group is considered here since this is perhaps the most practical from a computer organizational viewpoint as discussed in Section 5. Figure 4-12 is not intended to imply the exact shape of the curves or which curves lie on top of each other. Actually the curve for 2 or more groups could lie below that for 1 group etc. The determination of this would require the plotting of curves as shown in Figure 4-13 which shows the total number of cells vs. the number of groups

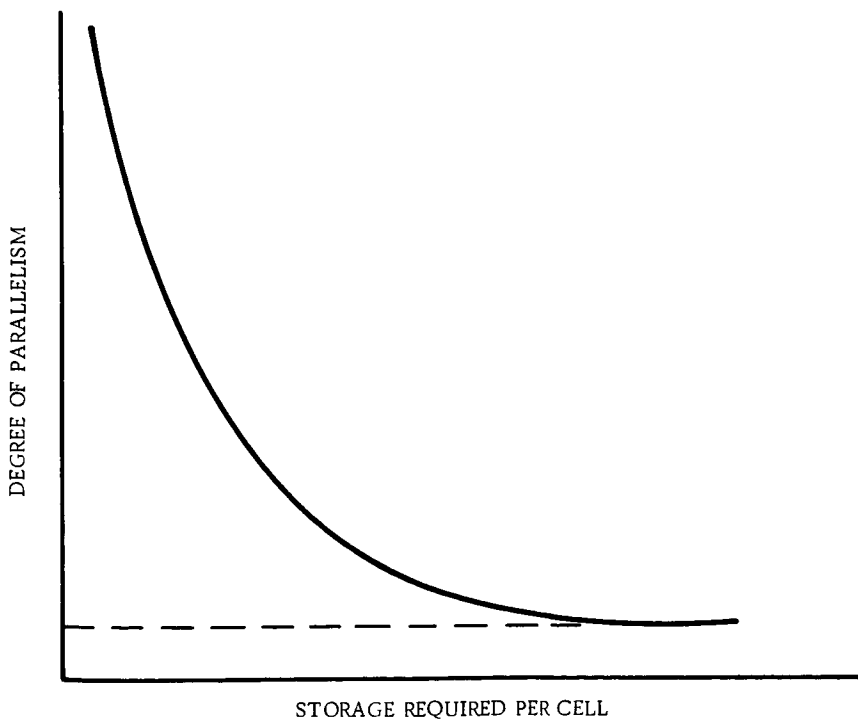


Figure 4-10. Degree of Parallelism vs. Storage Required Per Cell

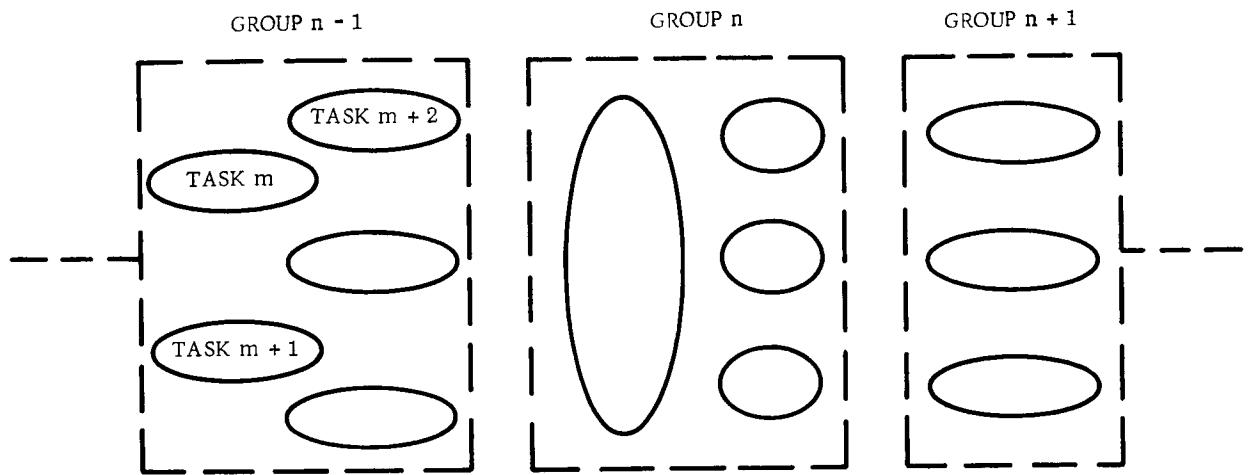


Figure 4-11. Computational Problem Assignment Among Groups

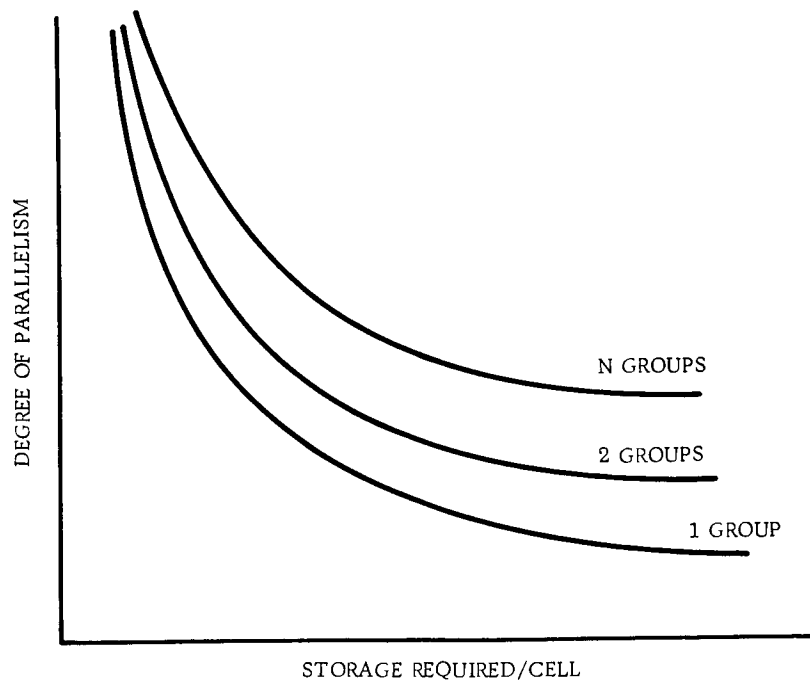


Figure 4-12. Degree of Parallelism vs. Storage Required Per Cell Using Groups of Cells

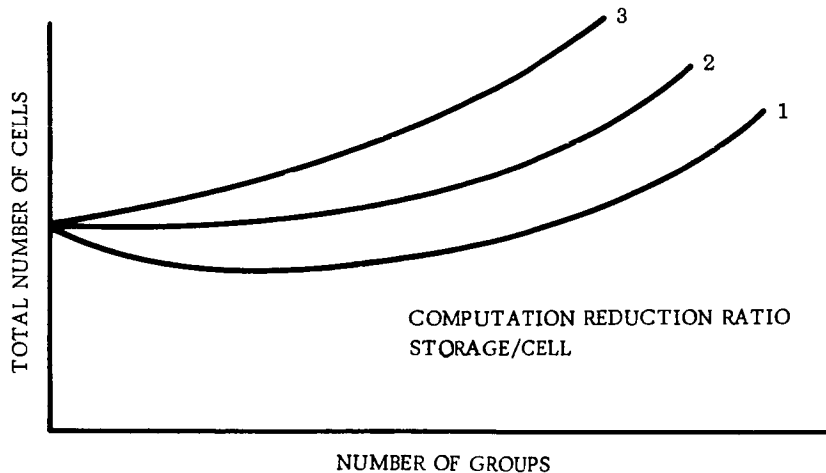


Figure 4-13. Degree of Parallelism vs. Number of Groups

(assuming a given computation reduction ratio and storage/cell). Curve 1 in Figure 4-13 shows the situation where a minimum is achieved with more than 1 group; this can occur if there is a significant amount of applied parallelism in the computations. This results since it is assumed that in distinct groups applied parallel operations can be executed independently and simultaneously of each other whereas in one group only one type of applied parallel operation can be executed at any one time by definition.

Curve 2 shows the situation where for a low number of groups the number of cells are approximately the same. In curve 3 the number of cells increases rapidly as more groups are used, this would occur if the computations contained little applied parallelism and there was considerable inefficiency in splitting the tasks among groups (e. g. one group used 27 cells and the other required 29 etc.). Given a set of curves as in Figure 4-13 one would attempt to pick a point which made efficient use of the number of cells.

4.3.5 An Alternative Approach

The entire discussion above has been oriented from a computation reduction ratio or speed viewpoint. That is with a certain degree of parallelism the maximum computation reduction ratio has been found and associated with each of these points is a storage requirement per cell if one considers a cell required for each degree of parallelism. It is possible to approach this problem from the other viewpoint: assume a certain degree of parallelism or number of cells available, arrange the computations so as to minimize the storage per cell, and associated with each point will be a speed requirement per cell.

It is obvious that the two approaches can yield completely different results. The latter approach will not be gone into detail here since the general problems associated with it have been brought out above in the first approach. It is possible to take both approaches simultaneously and set up criteria which yields a sub optimum approach in terms of speed and storage. This would be similar to finding the optimums by each approach and backing off each somewhat to determine the effect on the other parameter and selecting a suitable point.

4.3.6 A Practical Approach to Parallelism Studies

The theoretical approach to studying parallelism within the computations has been presented above. Unfortunately the approach is too complicated, as pointed out above to be used with a reasonable degree of success. To apply it however certain simplifying assumptions will be employed. The applied parallelism investigation as outlined above will be followed and a curve of degree of applied parallelism vs. computation reduction ratio will be obtained. A number of tasks will be analyzed to determine curves such as Figure 4-9 for each. Suitable points will be selected for each task to represent the degree of parallelism. For tasks not analyzed or for which no information is available, curves will be hypothesized. As mentioned above one task or set of tasks will take the longest to compute and therefore determine the minimum overall computation time. The tasks will then be laid out as in Figure 4-14. In this figure task m takes the longest to compute, T, and defines the maximum computation reduction ratio. The remaining tasks are assigned as shown. The assignment can be based on following an algorithm as shown in the lower portion of Figure 4-14. Essentially this assigns the tasks in order of increasing or decreasing degree of parallelism as indicated by the arrows. The time allotted is now varied and the assignment procedure repeated to obtain the degree of parallelism vs. computation reduction ratio curve for the total problem. It should be noted that the above assumes a certain constant degree of parallelism is required for each task throughout its execution. This is of course a gross simplifying assumption since as pointed out previously this degree of parallelism varies as a function of time through the task. The storage required per cell will be determined by combining storage graphs of the type indicated in Figure 4-15 for each task. The graphs will be combined for all the tasks assigned to any one vertical column (e.g. task i, j and k in Figure 4-14). This procedure may require the shifting of tasks somewhat if the storage requirements per cell are very unbalanced between vertical columns (e.g. task k may require being exchanged with two or more tasks which total up to the same degree of parallelism to smooth out the storage requirements). Following this very over simplified procedure curves as shown in Figures 4-9 and 4-10 will be derived.

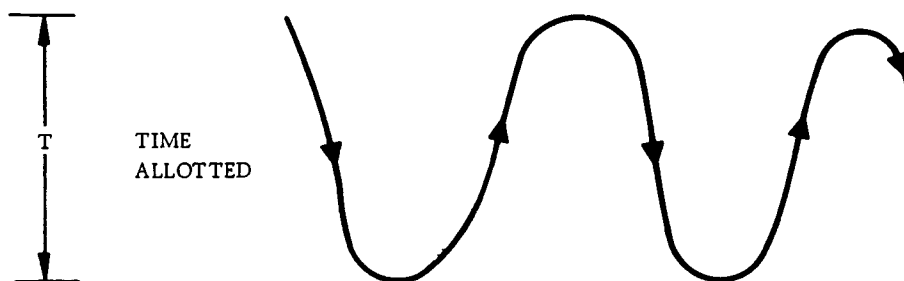
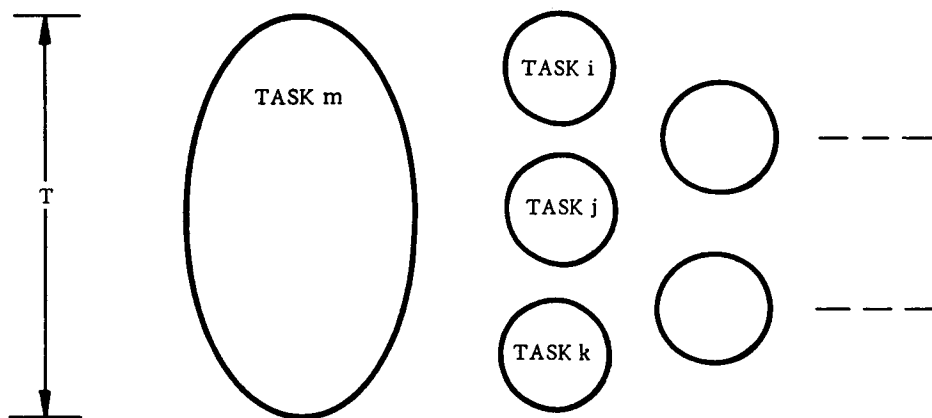


Figure 4-14. Assignment of Tasks

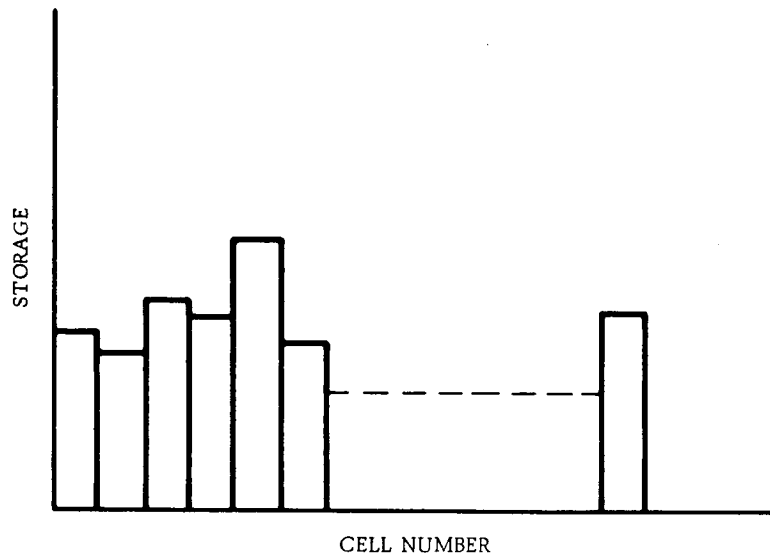


Figure 4-15. Storage Graph/Cell

4.4 RESULTS OF PARALLELISM STUDIES

The prior sections of this chapter contained a discussion of parallelism in general and methods for analyzing computations to determine the amount of parallelism within them. The computations for the manned Mars lander mission were analyzed for parallelism. The results of this analysis are presented in this section. Each of the computational tasks as defined in the requirements in the Phase I study (Reference 2) were investigated and the results are summarized in Tables 4-1 and 4-2.

Figures 4-16 and 4-17 contain the applied parallelism speed curve; they show the computation reduction ratio vs the degree of applied parallelism available in the computation system. The 100 percent utilization curve in the figures is the 1-to-1 curve, i. e. , for a degree of parallelism of 2 the computation reduction ratio would be 2, for 5 it would be 5, etc. The actual curve is seen to deviate slowly from the 1-to-1 curve at first and then reaches an asymptotic reduction ratio value of 13.66 for higher degrees of parallelism. The knee of the curve occurs at approximately a degree of 15; beyond this degree the curve deviates sharply from the 1-to-1 curve.

Figures 4-18 and 4-19 contain the computation reduction ratio and storage required per cell (assuming one degree of parallelism results in one cell), respectively, vs the degree of natural parallelism available in the computation system. It should be recalled that natural parallelism includes applied parallelism by definition. The vertical scale in Figure 4-18 is the same as in Figure 4-16; however, note that the horizontal scale is considerably larger. The computation reduction ratio curve does not

Table 4-1. Applied Parallelism Results

Degree of Applied Parallelism	Computation Reduction Ratio
1331	13.66
800	13.66
300	13.65
100	13.47
50	12.58
15	8.75
5	3.98
2	1.895
1	1

Table 4-2. Natural Parallelism Results

Degree of Parallelism	Computation Reduction Ratio	Storage (Words/Cell)
1342	42.4	
300	42.4	200
100	42.4	350
50	28.2	575
15	12.15	1,800
5	4.6	5,250
2	1.96	12,700
1	1	24,633

have as sharp a knee as in the applied parallelism case. However, it appears that somewhere in the range of 40 to 60 in degree of parallelism the curve starts to deviate rapidly from the 1-to-1 curve. The storage curve is drawn on log-log paper to bring out the deviations from the 1-to-1 curve. It can be seen that the curve begins to deviate rapidly from the 1-to-1 curve in the vicinity of a degree of parallelism of 80 to 150.

The above curves give an indication of the efficiency or utilization of parallelism. They may also be used in determining the speed-storage characteristics of the cells. It should be recalled from the technology section of this report that a storage capability of approximately 512 words per cell was considered to be achievable. If this storage is available per cell, then, referring to Figure 4-19, one can see that approximately 58 cells are needed (assuming a degree of parallelism equals one cell). Translating this into speed requirements one can see from Figure 4-18 that a computation reduction ratio of approximately 31 results with 58 cells. Since the speed requirement for a single computer is approximately 1,450,000 short operations/sec (the parallelism investigations were carried out for the Mars orbital phase which has the maximum speed and storage requirement), the speed requirement per cell is therefore approximately 25,000 short operations (ADD, SUB, etc.) per second. Since these requirements did not include overhead functions (such as the executives) one may estimate the number of cells required at approximately 80 with a storage capability of 512 words each and requiring a speed capability of 25,000 operations per second. One can now see that the cells will most likely be storage restricted rather than speed restricted since the cells should be capable of more than 25,000 short operations per second.

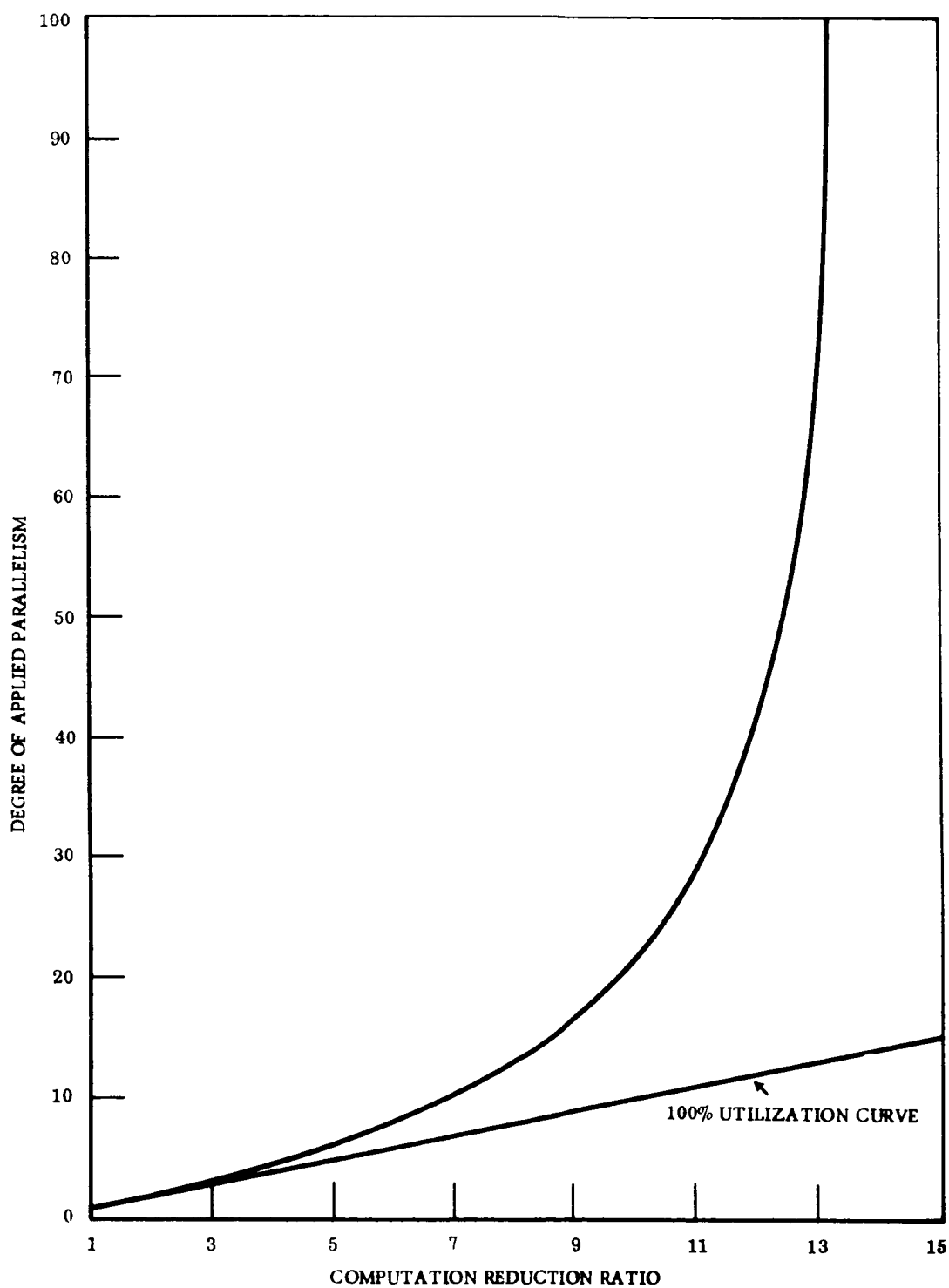


Figure 4-16. Applied Parallelism Speed Curve

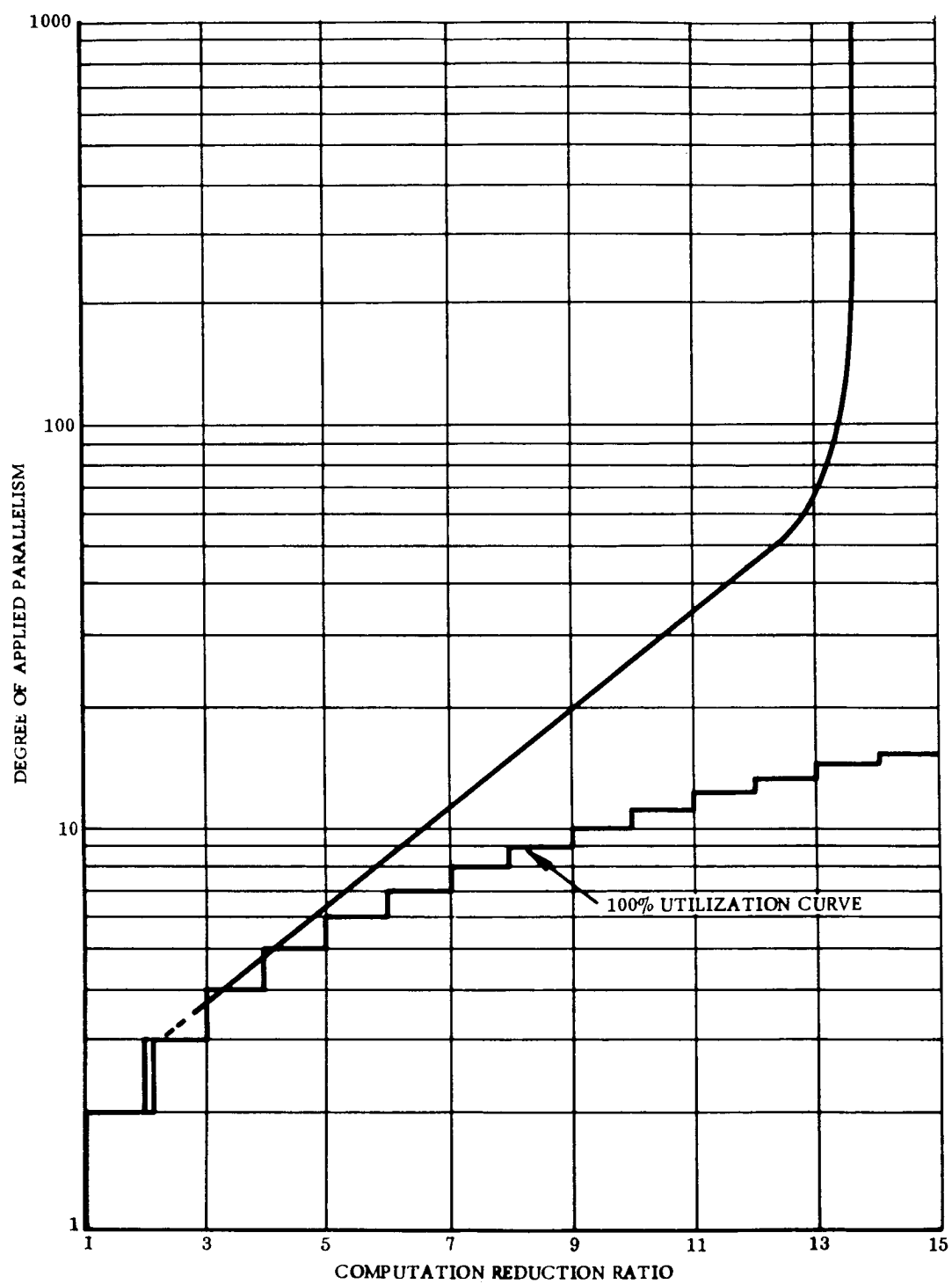


Figure 4-17. Applied Parallelism Speed Curve

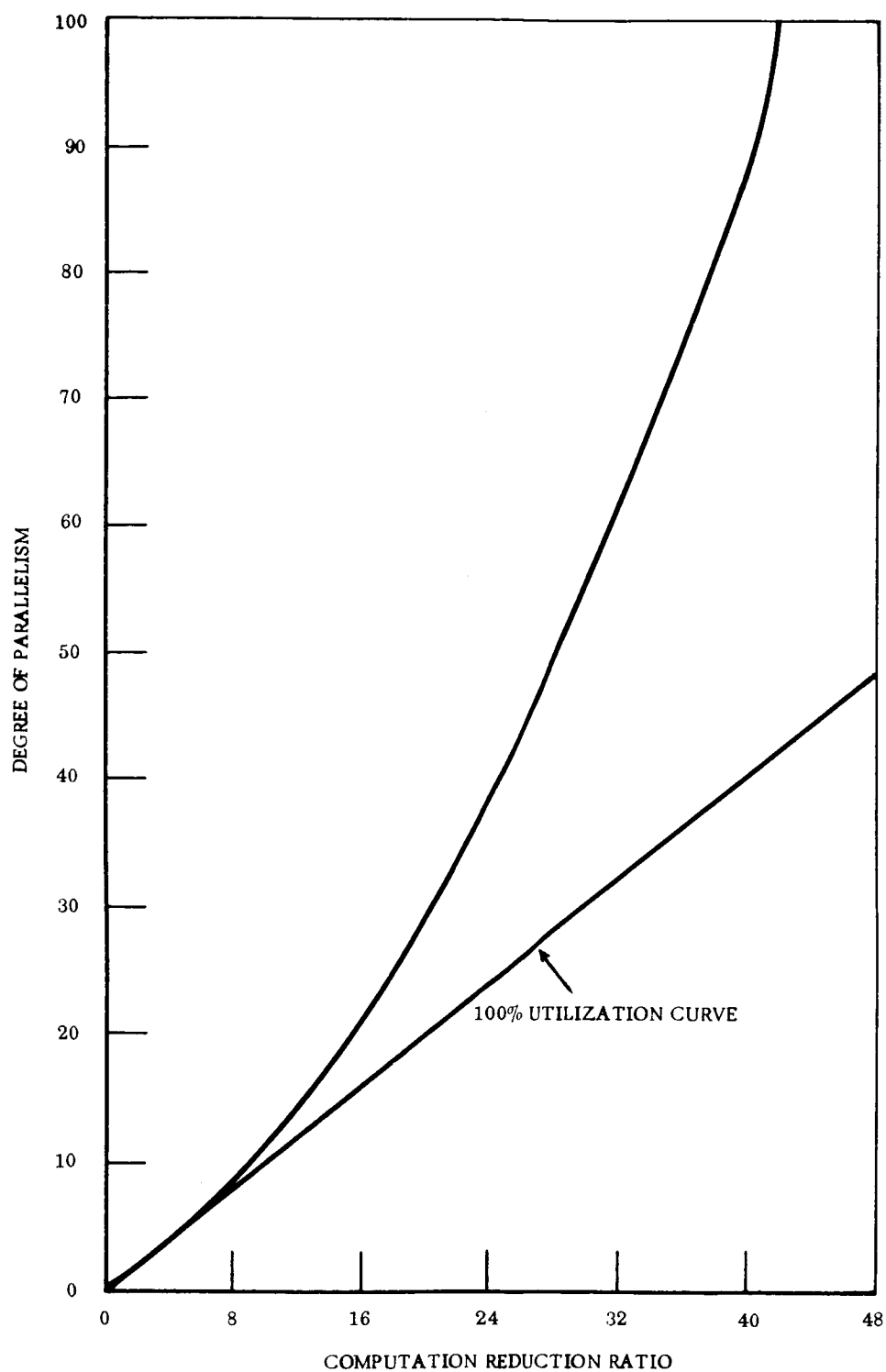


Figure 4-18. Natural Parallelism Speed Curve

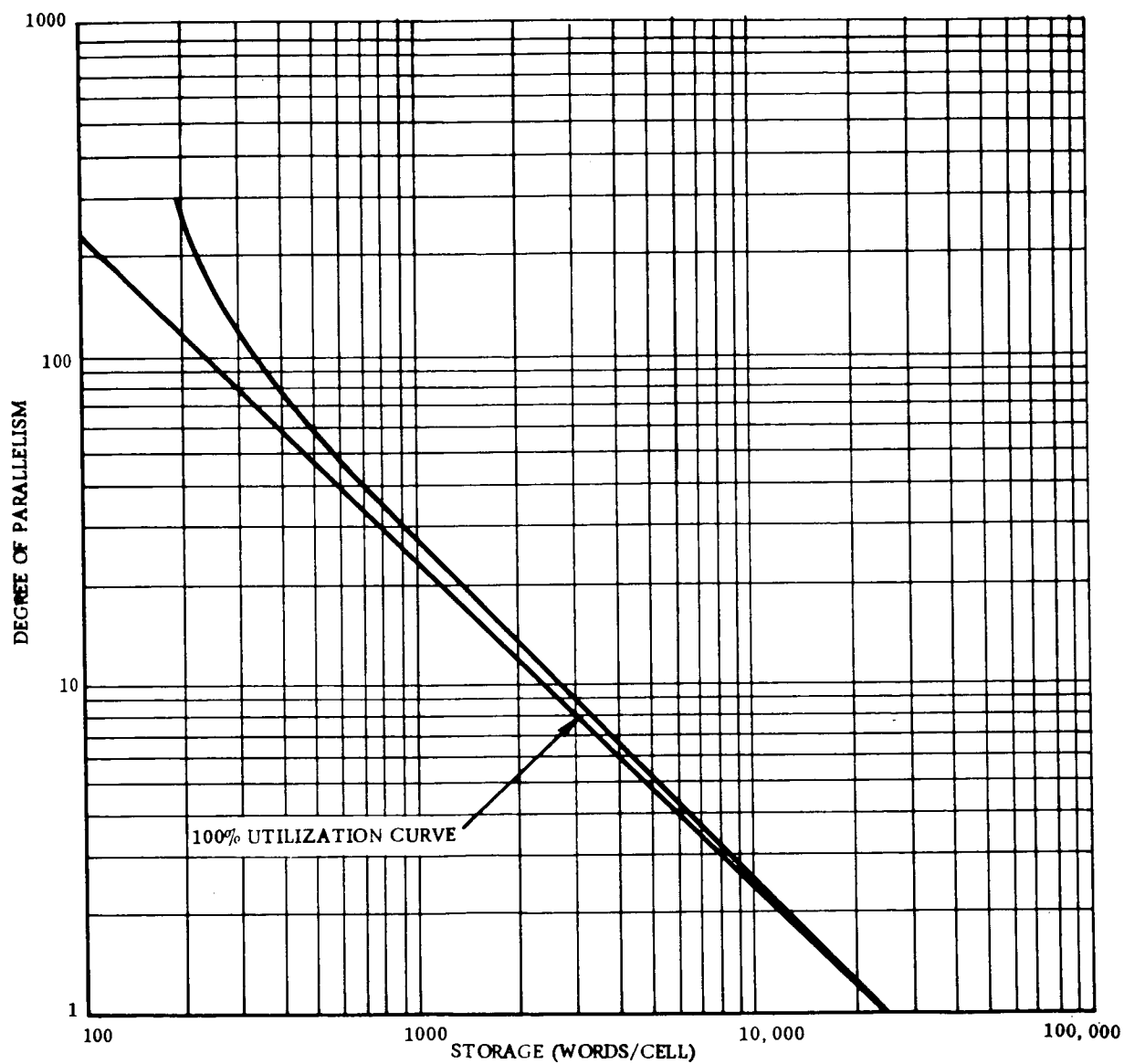


Figure 4-19. Natural Parallelism Storage Curve

5. COMPUTER STRUCTURES

5.1 DEVELOPMENT OF COMPUTER SYSTEM ARCHITECTURE

5.1.1 Introduction

This section discusses a number of approaches to construction of general purpose computer systems. It is included in order to put into perspective the Distributed Processor System under study in this contract. (Hereafter the Distributed Processor will be referred to as DAMP for Distributed Array Memory and Processor.)

The various system approaches are presented in an order that is consistent with the development of the technology necessary to implement each structure. This order will be seen to be almost chronological; however, there will be some deviation since a number of interesting and practical computer organizations have been conceived both a number of years after and before the technology was available to practically implement them. It should be noted that each organization was not developed to explicitly use the technology but rather was developed to attempt to meet a certain set of requirements (cost, high speed, reliability etc.) in a more efficient manner by taking advantage of new hardware technology. In fact, in many cases, initial organizational considerations of a system may have not even considered technology, but instead were totally concerned with a theoretical structure suitable for meeting certain requirements.

The discussion of computer organizations is pointed toward general purpose structures since the requirements for the missions discussed in Section 2 dictate a machine capable of carrying out a wide variety of computational tasks. As a result, associative memory structures will not be discussed here since it is generally felt that associative machines are not practical for general purpose computers. (See Reference 23.) There are also such a large number of these structures suggested in the literature for solution of certain types of problems that it would not be practical to present a synopsis here. However, this lack of discussion does not necessarily eliminate the possibility of using associative memories for carrying out certain specific functions within a more generalized G. P. computer. On the other hand the Solomon and Illiac machines are discussed even though they are also recognized to only be practical for execution of a relatively limited set of computational tasks. However, these structures are of interest since they are powerful distributed processing structures that can be practically implemented with near-term LSI technology and because they provide good examples of the use of global control (a single instruction stream with multiple data streams).

5.1.2 Single Computer

In the early stages of computer development machines were generally built from a variety of digital circuits with varying fan-ins and fan-outs, such as AND gates, OR gates, inverters, etc. The development of the transistor enabled machines to be efficiently built with a very few basic circuits, for example, simply NAND gates. This eased the design and fabrication process considerably; and as a result, much

less expensive machines could be constructed. With the more recent development of integrated circuits the construction of computers from a very small number of types of circuits has become wide spread. This has not only vastly decreased the price of computers, but has also simplified reliability programs, sparing, and maintenance. The above developments have, of course, also reflected into more economical and reliable memory circuitry.

The economy possible with transistor and IC computers enabled multiple computer and multiprocessor structures to be proposed and constructed for applications with requirements that were difficult to meet with a single computer. For example, large scientific computing centers with requirements for greater processing power than available from a single computer are able to use multiple computers or multiprocessors (time-sharing systems, etc.). These structures have also found use in aerospace applications where greater on-line reliability than that offered by a single computer was required.

5.1.3 Cellular Arrays

The development over the past number of years of batch fabricated complex integrated circuits has helped foster a good deal of research. Of particular interest here is the increase in research activity pointed toward development of a single logic element that could be used with a fixed connection pattern to implement a wide variety of combinational (and sequential) circuitry. This type of work can be seen to be the logical extension of the early use of IC's to minimize circuit types since structures built from these IC circuits required complicated interconnections. Combinational circuitry of the above type has been called cellular arrays. The cells or the single logic elements in the arrays can either be fixed or variable (using cutpoints as in Reference 24); and the interconnection structure can also be fixed or variable. (It can be variable in the sense that cells can be preset so that they do not use certain available connections.) A survey of this microcellular research is given in Reference 25.

Reference 24 by Minnick et. al. at SRI provides a good example of research being carried out on cellular arrays. It is concerned with the organization, use, and design of arrays constructed from "cutpoint" cells. These cells are constructed on part of an IC chip and are provided with cutpoints (switches or flip-flops) that can be used to specialize the operation of the cell to one of six simple logic functions or to an R-S flip-flop. The usefulness of these cells within a variety of fixed and variable connection structures is then discussed and evaluated. The structure of one representative array organization is shown in Figure 5-1. This organization can vary its connection structure by disconnecting various input leads from any cells. These structures used to implement the logic in a computer could offer a number of advantages by the good utilization of IC technology. For example, a number of cells including an interconnection structure could be fabricated on one IC chip and then mounted in a package. If this was practical it would clearly provide very inexpensive and reliable logic for the processing section of a computer. (It would be reliable since a small number of external connections would be required.) However, there are a number of problems with the approach that may make it impractical: (1), In order to use the fabrication and packaging structure described above, an efficient algorithm to avoid faulty cells must be developed. The present algorithms require extensive duplication of hardware; (2) the number of cells required to implement a given switching function is much larger than the number required with non-cellular logic; (3) at the present,

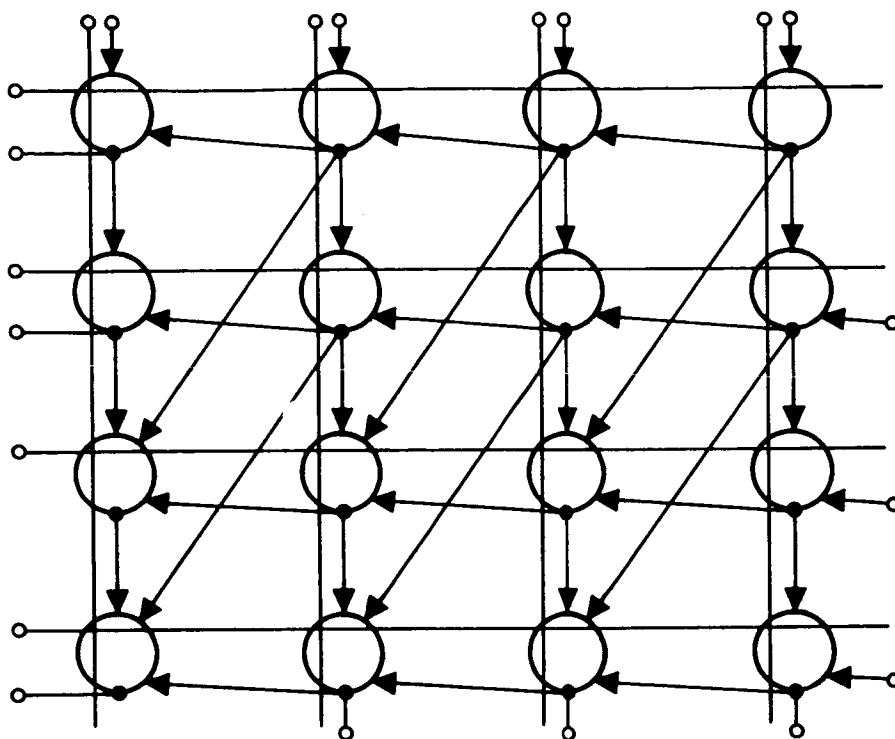


Figure 5-1. Structure of the Cob Web Array (From Ref. 24)

effective logical design techniques do not exist; and (4) much more effort is required to be able to simply design a computer with the circuitry. Solutions to the above problems are necessary to make the Minnick's arrays and in fact cellular arrays in general at all practical for computer design and fabrication. In fact, an even greater draw back to these structures is the fact that the onset of LSI technology in the future will make it practical to construct a complete processor on a chip. (Just a few chips in the near future) As a result, a single computer, a multiprocessing system, or a distributed processing system using a single chip for a processor may well preclude the use of cellular arrays to construct processors. However, these techniques, if efficiently developed, may find some use in increasing yields of processor chips by using discretionary wiring or some other faulty cell avoidance algorithm.

5.1.4 Distributed Processing Arrays

In the past and at present a considerable amount of thought has been devoted to developing arrays of processors capable of carrying out a number of computations in parallel. For the most part these structures have been suggested to solve various types of problems, such as associative processors for information storage and retrieval, Holland machines for direct control of highly parallel systems (e. g., pattern recognition etc.), and Solomon machines for solution of linear systems, ordinary and partial differential equations, matrix operations, etc. Some work has also been pointed toward the use of these machines on a set of general purpose problems; however, although they would be capable of solution of these problems, the array structures are quite inefficient in this case when compared to conventional computing systems. An additional reason for interest in these distributed processors

is that they could take good advantage of the present and developing LSI technology. Each processor could be constructed from a single chip and the chips could be packaged on a standard board with a fixed connection structure. Two distributed processing structures will be briefly discussed below.

5.1.4.1 Holland Machine

The Holland machine has been discussed in an earlier report Reference 1 and also in the literature, Reference 22 and 26. It consists of an array of cells, see Figure 5-2, each of which has a small amount of storage and is capable of executing a small number of arithmetic and control instructions. This structure, like the Minnicks logic elements, would take good advantage of IC technology by implementing each cell as a chip and by packaging the chips on a board containing the fixed standard interconnection structure. The real advantage here over the Minnicks structure is that both the processing and memory are included in this regular structure. The cells operate under local control and as a result can execute a number of different instruction streams in parallel. Any cell can act as a controller cell, 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 to carry out the operation. This basic iterative machine structure, investigated in some depth at the University of Michigan (see Reference 22), was found to be impractical because the path building necessity extremely complicates the programming and also makes it difficult to reconfigure after failures. The above problems were also compounded by the fact that very little memory was available to each cell and that only a small instruction set was available.

Although this machine was not considered practical, the discussion of parallelism and the usefulness of local control for executing parallel computations is of interest. In particular the ability to locally control computations enables a large number of distinct problems as well as parallel parts of a single problem to be executed simultaneously. Although the Holland machine was investigated for problems in the fields of pattern recognition, games, simulation and adaption etc., this local control feature can be seen to be of value for executing a set of general purpose problems on a distributed processor machine without severe programming problems. Each separate problem can be executed simultaneously if the computing power is available; however, the efficiency of hardware utilization would not be as high as for a fast single computer since some problems or groups of problems will be found to be sequential in nature. Nonetheless, it will be shown below that the use of a distributed processing machine capable of local control is worthwhile under certain sets of requirements (e.g. high reliability, high speed on certain problems).

5.1.4.2 Solomon and Illiac Machines

The Solomon machine was developed in order to obtain a high solution rate on certain classes of problems by applying parallel processing. The class of operations involve "sets of variables which permit simultaneous independent operation on each individual variable within the set". The high solution rate is achieved by utilizing a central control unit to control the operation of a large number of processing elements operating on separate variables (global control). This machine, shown in Figure 5-3, is also discussed in Reference 1 and in the literature (References 27 and 28). Each processing element has two thousand bits of memory available to it along with some

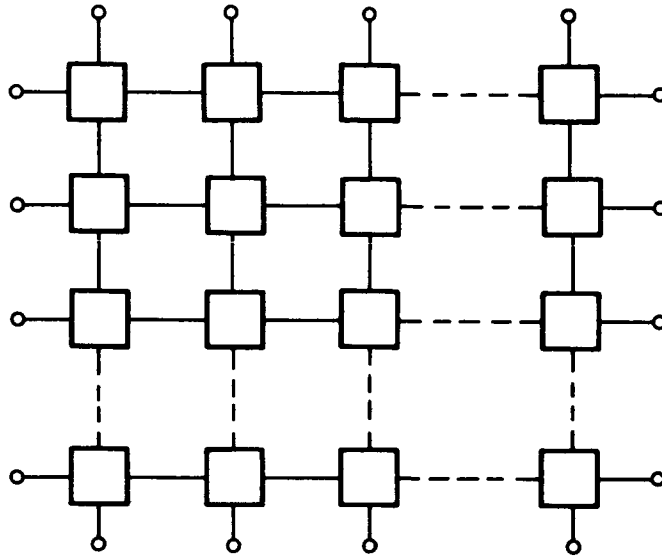


Figure 5-2. Holland Machine

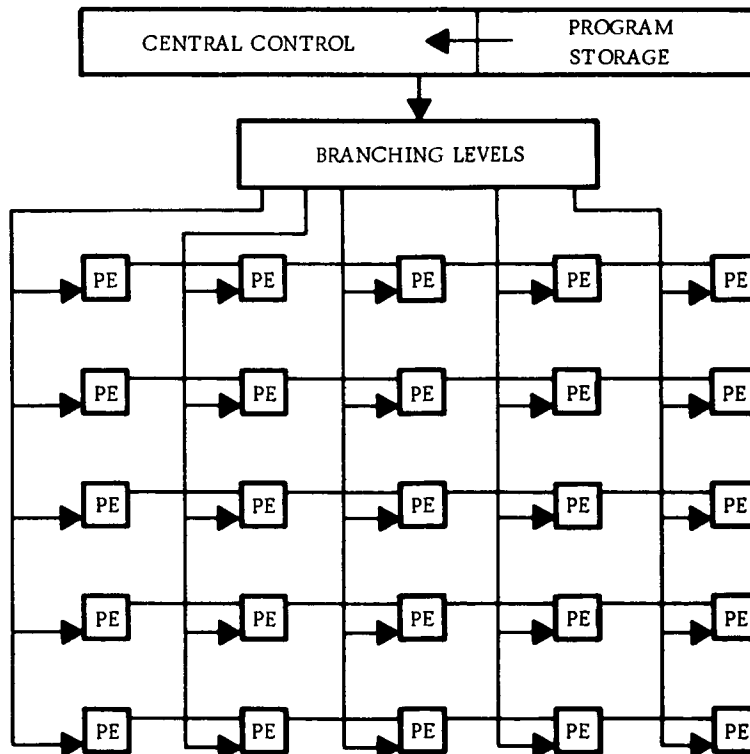


Figure 5-3. Solomon Machine

processing ability. The local control is limited to simple decisions as to whether or not to receive the present control word from the central control unit. This structure simplifies the processing element which of course is important since there may be a few hundred of these in the system. It is also organized to take good advantage of present LSI technology. (The connection structure is regular and the processing elements and memory can be constructed from a small number of common LSI chips; however, separate types of chips must be used in the central control unit, the processing elements, and the memories. Note that the Holland machine, if practical, could have used just one chip type for everything.)

The Solomon structure, as mentioned earlier, was developed to efficiently solve the class of problems characterized by a single instruction stream simultaneously operating on multiple data streams (global control). This means that the structure would be relatively inefficient (and speed limited) on a set of general purpose computations since global type operations would only occur a portion of the time. For example, whenever sets of sequential calculations must be performed all but one of the cells are idle. (Even with parallel computations many cells are idle since each calculation may not involve enough data streams to use all the available processing elements simultaneously.) This limitation of processing capability for general purpose computations is lessened to some extent in the Illiac machine by dividing the processing elements up into separate groups each with its own control unit. The Illiac is an advanced version of the Solomon machine and as a result its operation and purpose for development (application area) is basically the same as for the Solomon machine. The main difference is that the grouping of processing elements enables the Illiac structure to obtain higher hardware utilization since all the groups can operate simultaneously on separate tasks or on the same task. It should again be noted that the limitations of the Solomon and Illiac machines on general purpose computations, shows the problems of using only global control for execution of these computations; however, this is not intended as a criticism of these machines since they were not developed to efficiently execute a wide variety of computations.

The real interest here in these array structures is to introduce and investigate the value of global control for execution of certain parallel computations. It was stated above that the global control of a large number of processing elements enabled high solution rates on appropriate problems. These high rates are obtainable since a single central control minimizes the overhead associated with obtaining instructions and sequencing computations on the processing elements. (As mentioned earlier this structure also saves a significant amount of control hardware in the Solomon processing elements) In addition to saving overhead the use of global control can save a good quantity of memory since only one instruction stream must be stored to operate many processors. With a local control structure the same instruction stream would have to be stored in a number of places in order to be accessible to a large number of processing elements. (This would be necessary since a central subroutine memory would have low accessibility because it would need to be used by many processors.) Because of these advantages of global control, it would be useful to add global control capability to a local control distributed processor if the hardware did not become unduly complex. Such a structure is discussed in the next section.

5.1.5 Distributed Array Memory and Processor (DAMP) System

The DAMP System was developed to provide general purpose computing with very high reliability for long duration space missions. In addition it was desirable for the system to make good use of future circuit and memory technology, to dissipate a small amount of power, and to be easily expandable or contractable so that it could be flexibly applied to a variety of missions. The high reliability requirement can actually be interpreted as a need to continue executing a critical subset of the computations in spite of hardware failures (see Paragraph 2.2). In other words the computations can be "gracefully degraded." The DAMP system meets this goal by dividing its memory up into a number of modules (approximately 512-16 bit words used per module) and then integrating each memory module with a processor to make an independently operating "cell." These cells can then individually fail without bringing down the whole computing system. In addition they can be turned on and off to match the requirements from phase to phase. This will substantially decrease the power dissipation of the system and also increase the reliability (assuming off line or dormant reliability is higher than on line or operating reliability). The same DAMP structure consisting of many cells can meet the expandability requirement by adding a small or large number of cells to match a new set of requirements. The system also makes good use of future circuit and memory technology since each cell can be fabricated on a single LSI (MOS) wafer in the 1980 time frame (see Section 3 on technology). In particular, the regular pattern of the memory in a cell will enable discretionary wiring or encoding techniques to be used to obtain reasonable yields on large complex wafers (or cells). In addition the cells are used in a regular fixed connection pattern so that they can be easily packaged on a board. This discussion of the general features of the DAMP System is expanded below.

The organization of a cell is shown in Figure 5-4. Other possible cell organizations could use additional chips in order to give each cell more memory. For example separate wafers containing 512 to 1024 words of memory and no processing could be added to the cell shown in Figure 5-4. Further investigation and programming effort is necessary to determine if the presently chosen single wafer cell is memory limited and thus in need of additional memory wafers. Clearly the single wafer cell is preferable if the system is not memory limited since it requires somewhat less connections, offers higher processing rates on certain problems, and because it offers a higher degree of graceful degradation (more memory per cell would mean less cells in the system).

The cells, as previously mentioned, are organized into groups that have a regular connection structure that results in simplified packaging. The system grouping and connection structure is shown in Figure 5-5. Appendix F presents a discussion of the alternative organizations that were also considered, thereby leading to the structure shown in this figure.

In light of the above discussion of the cell and group structure the ability of the system to gracefully degrade can be better understood. A few spares will be placed in each group so that a few cells can fail and simply be replaced by the spares without affecting computation. After these first initial failures, the computation power of a group would simply degrade in small increments with each additional cell failure. Note that this enables the critical subset of computations to be continued in spite of a large number of failures. It should also be noted here that a failure of an inter-cell bus would cause a whole group to fail, but the system is designed such that this

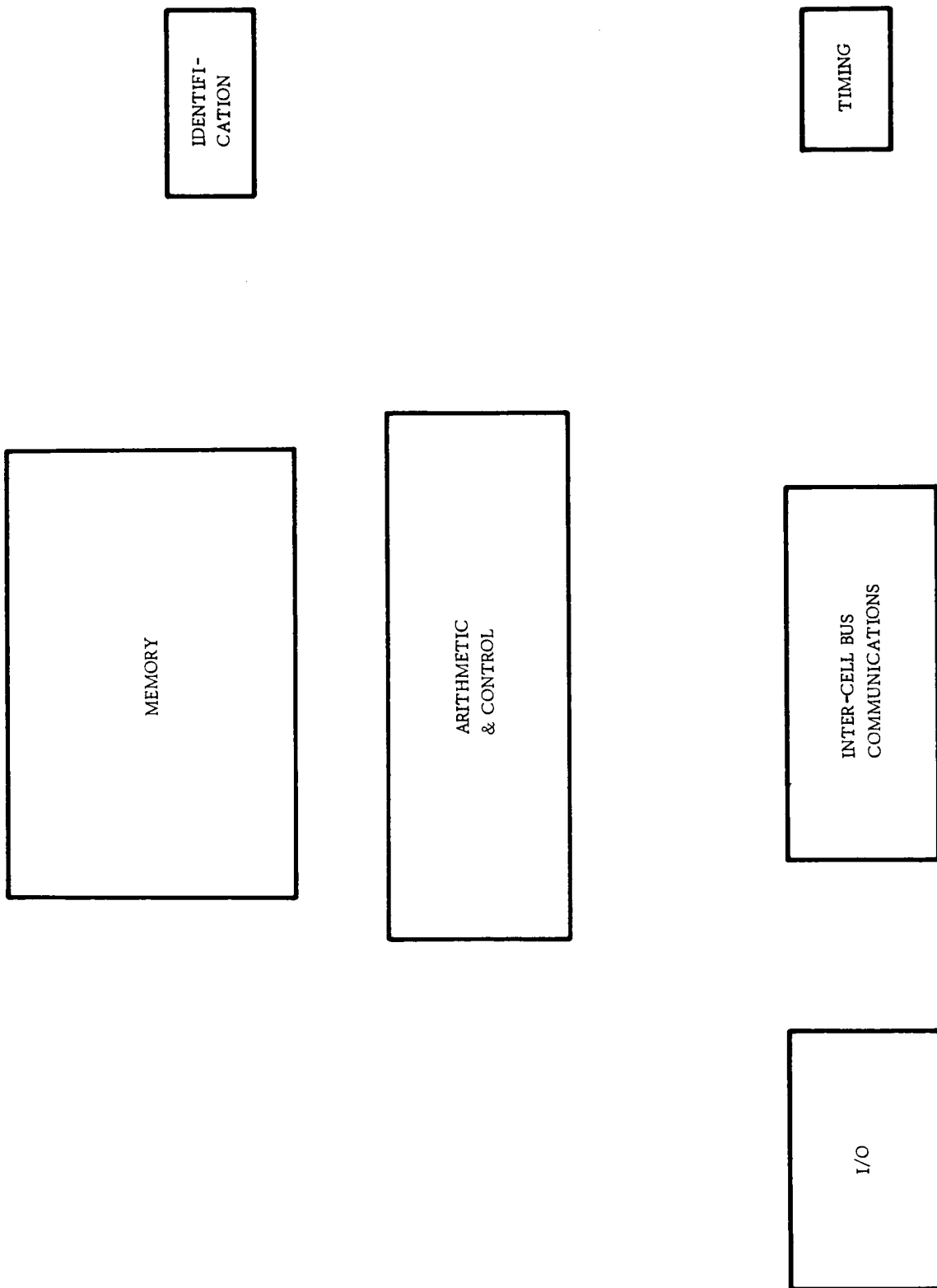


Figure 5-4. General Cell Block Diagram

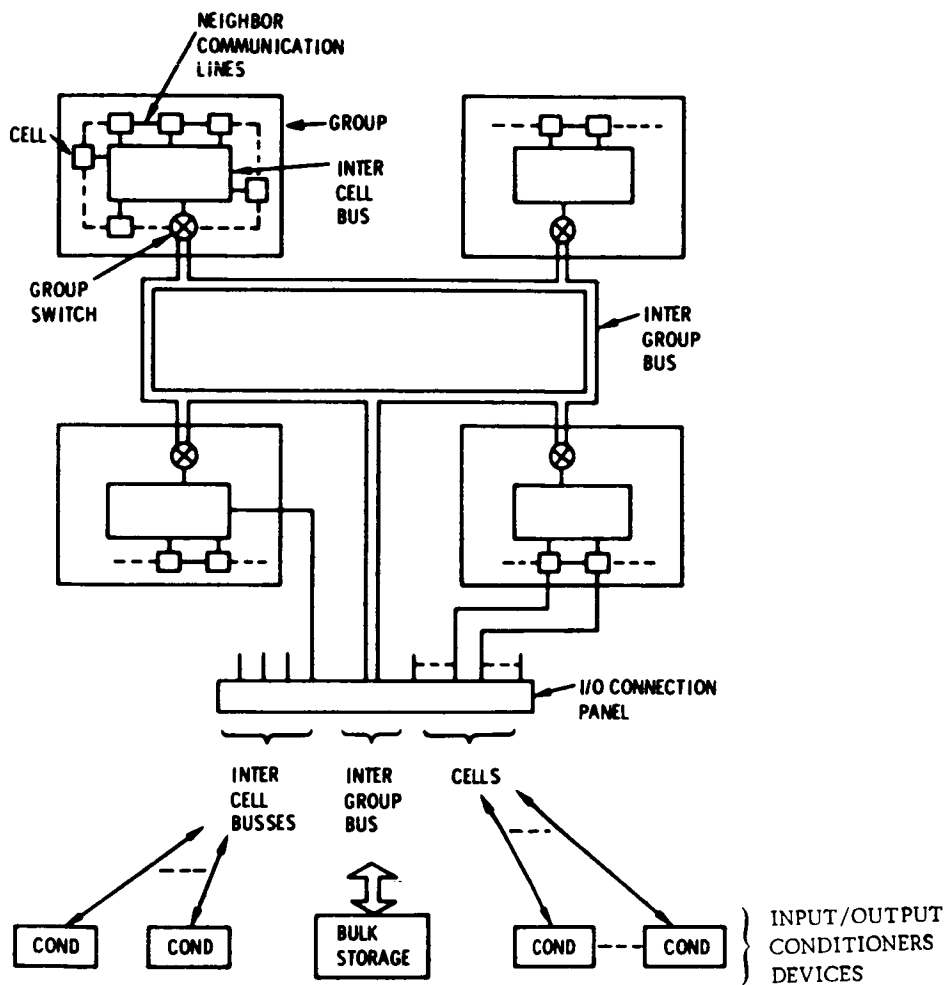


Figure 5-5. Distributed Processor Organization

failure mode has a very low probability of occurrence. (The bus contains no hardware, besides lines and hardwired or deposited connections.)

The interesting point to note about the DAMP System is that it is able to meet the reliability, expandability etc., goals as described above with only a small addition of hardware to the memory of a conventional computation system. (In fact, depending on the particular set of general purpose computations, discussed below, this may run from slightly more to even less total hardware than would be needed in a multiple computer system to do the same tasks.) To understand this statement, consider the point brought out in Reference 1 that in the time frame of interest the main memory of most computation system could be equally well constructed from semiconductor wafers or from magnetic arrays; as a result, the DAMP System can be considered to be simply a semiconductor memory that has a small amount of each wafer devoted to processing (the processor will take up less than 1/10 of each wafer containing processing and 512-16 bit memory words as discussed in the technology section). This means that the high reliability, flexibility, etc., gained with this system has cost a very small hardware increase. In fact, for some computation mixes the DAMP system may use less hardware than the equivalent multiple computer. For example the DAMP System would use less hardware if the computations were somewhat specialized so that they contained a good amount of applied parallelism and/or small high rate programs that are naturally parallel. This point will be made clearer by the discussion of local and global control in the next paragraphs.

The DAMP System provides reasonable efficiency on general purpose computations by taking advantage of both local and global control.¹ In addition, each cell has a reasonably large instruction set and memory; as a result, relatively large subtasks can be executed by a cell. This large capacity significantly eases the programming and communication problems typically encountered with distributed processing organizations. The use of local control (global control is described in the next paragraph) enables each cell to execute any computation simultaneously with other computations as long as there is no sequential dependence of this computation on others being processed or about to be processed. (In other words no precedence relations amongst the parallel computations.) This latter restriction provides almost no conflicts for execution of separate tasks simultaneously, such as the navigation and guidance task and the status monitoring task. However, for those subtasks or tasks that are sequential in nature, the above restriction means that only one cell can be used at any one time to execute these computations. (Other independent tasks or subtasks can still be in execution, however). For the latter reason during certain periods, the DAMP system can be operating in an inefficient manner when compared to a uniprocessor capable of executing the same set of computations.² The processing

¹ A uniprocessor would provide the most efficiency on a set of general purpose computations, if it could be made fast enough, since less overhead is required; however, this type of system would clearly offer much lower reliability. In addition, if the computations had some specialization as noted in the last paragraph, the DAMP System would offer both higher efficiency and reliability.

² This local operation can be thought of as a simple extension of a multiprocessor (or multiple computer) system. The same restrictions clearly apply here too, but they are not as pronounced since there are a smaller number of processors and memories.

section of a cell will be described in detail in section 9, however it will be stated here that each cell contains a processor capable of executing a reasonably large instruction set in a parallel by bit fashion with clock speeds as high as a few MHz. This means as mentioned earlier, that the operation of a cell may be limited by the amount of memory available within itself. This will not be the case only for computations that require small programs that are executed at high repetition rates. One method used to alleviate the possible memory restriction involves letting one cell use another cell's memory via the communication buses. This enables the cells with small high-rate programs to share their memory with other cells that are memory restricted. A second method of alleviating the memory limitations is by providing the system with the ability to use global control where it is applicable. (As mentioned earlier, global control is characterized by the use of a single instruction stream on multiple data streams and is therefore useful in problems that require the same set of operations on different pieces of data, e.g., matrix manipulations etc.) The DAMP System takes advantage of global control by using one cell to hold the instruction stream and to send this stream over the communication bus to the other cells in a group executing the same problem. In this manner, redundant storage or sequential redundant access of the same program is avoided. The use of global control as discussed here thus enables a number of processors to be used with only a minimal amount of memory. A third method of decreasing the amount of storage required per cell is to provide less buffering in the DAMP's memories for I/O data. This would increase the required I/O rates but that is acceptable since extra processing is available. In addition to the above, one of the groups that operates as the central executive will provide a storage area for common variables and constants if appropriate.

One of the primary advantages of this organization, as previously discussed, is its ability to withstand many failures and still continue executing at least some computations. Another way to look at the system is to say that it is designed to be tolerant of cell failures. It is tolerant in the sense that not only can a number of cells fail and simply be replaced by spares connected into a group but also cell failure beyond the number of spares available only causes small degradations of the system. A little reflection will show that this type of operation is significant since actual on-line failures (caused by overstresses etc.) can be tolerated along with a reasonable number of on-line failures caused by use of originally defective parts. This failure distinction, also made in the technology section, is a fine point, but it is important, since it has been observed that a great many failures in complex IC's are actually caused by manufacturing defects that were not detected prior to placing the system on-line. In fact, many of these failures only occur under a very specific set of circumstances and so may not cause on-line failures for thousands of hours. These failure modes can generally be removed by feeding back information to the production line, but it is a long and expensive process to obtain very reliable components. Of even more importance is the fact that as circuitry reaches greater and greater complexity on a single chip, the practicality of giving a chip a 100 percent test on the ground before inclusion in a system decreases immensely. As a result, the ability of the computation system to tolerate a number of latent manufacturing defects will gain more significance in the future. It should also be noted that this ability to tolerate cell failures does not exist to the same extent in a global structure such as the Solomon machine. A single failure in the main control unit or memory of such a structure could bring the whole system down. (Clearly making these central units redundant is costly in terms of hardware when compared to a structure like the DAMP System.)

The main anticipated difficulty with the DAMP System is making it relatively easy to program. The tasks must be split up into subtasks such that when programmed they will fit into a single cell. This splitting should also give as much freedom as possible in scheduling the tasks on cells. In other words constraints on executing certain computations in a sequential fashion should exist, as possible, only within the subtasks to be executed on an individual cell. The allotted subtasks must then be assigned to the minimum number of cells. This assignment problem is similar to the problem of determining the minimum number of cells necessary to implement a system for a given set of requirements. This latter problem was discussed in Section 4 on parallelism; the discussion in that section indicated that an optimum assignment is difficult to achieve.

An additional problem exists once an assignment of programs to cells is made. The subtasks must be scheduled so that all of them can be accomplished in the required period of time. This requires consideration of allocation of separate periods to the global control programs and to avoiding conflicts on the intercommunication busses. Clearly the scheduling operation could also feed back on the assignment of subtasks to cells such that a new assignment is necessary.

From the foregoing discussions it can be seen that actually programming a subtask for a cell would not be much more difficult than programming the same subtask for a single computer system; however, if the programmer is required to subdivide tasks, assign them to cells, and setup the scheduling, the programming job will be relatively difficult. Therefore some important areas of development are software packages to automatically subdivide programs into tasks that can be executed by a single cell, to handle the automatic assignment of subtasks to cells, and to schedule automatically the global control programs and intercommunication on the busses. Section 11 will present some preliminary considerations of these topics.

It should be noted here that there are other possible approaches to obtaining very high reliability that have been and could be investigated. One approach is to use throughout the system encoded information that contains enough redundancy to correct multiple failures. However, this type of system requires considerable redundancy to withstand many failures; it will also totally fail after a fixed maximum number of failures have occurred. This maximum would be considerably less (for any practical system) than the number of failures that the DAMP System can tolerate. A second possible approach would be to develop a multiprocessing system with a number of processors and with memory modules that are subdivided into small individual blocks with separate sets of read/write and addressing circuitry. This type of system could gracefully degrade by losing part of a memory module or a processor with each failure.

5.2 OPERATION OF THE DISTRIBUTED ARRAY MEMORY AND PROCESSOR SYSTEM

The last section described the development of a number of computer organizations. The last such organization, the DAMP system, was shown to be well suited to advanced space missions due to its high reliability, easy fabrication and expandability, and ability to be turned on and off in small increments. The chosen form of the DAMP system is described in this section.

The distributed processor structure consists of cells. Cells are interconnected to form a group and a number of groups are interconnected to form the complete structure as shown in Figure 5-5. A cell may be thought of as a small conventional computer that has a relatively small memory. Parallel operation may exist between cells in a group and between the groups in the system. The structure thereby appears as a highly parallel computational facility. For the purpose of this study, the groups will be considered as being all located physically in one central location so that the organization appears as one large parallel system. (An alternative physically distributed system is discussed in Appendix F.)

The computational tasks that are presently required to be carried out are divided or assigned to various groups and finally to various cells in the groups. Groups of cells may carry out a complete task (such as the navigation functions in a spacecraft), a number of small tasks, or even part of a large function or program. One of the primary considerations used in dividing programs among the groups is to limit the inter-communication between groups as much as possible. A second consideration is to enable as much global control as possible to be used in order to take advantage of applied parallelism and thus maximize hardware utilization. This point was discussed in Section 4 on parallelism. As will be explained below, each group will be capable of executing a global program⁽¹⁾ by a number of cells in the group, this capability allows the system to take advantage of applied parallelism in the computations assigned to the group. It should be noted that at any one time, only one global program may be in execution in the group; however, since there are a number of groups in the system and the groups operate independently of each other, a maximum of one global program per group can be in simultaneous execution in the organization at any one time. Therefore, one of the considerations in dividing programs among the groups is to assign them so that as much applied parallelism in the total computational task is being utilized as possible.

The tasks assigned to a group are subdivided or assigned to individual cells within the group for actual computational mechanization. The actual task or tasks assigned to a cell will, in general, be a part of a larger task. Therefore, there may exist, in a group, a certain amount of sequential relationships or precedence restrictions on the execution of the tasks in cells that must be taken into account when assigning tasks. An example of a sub-task that might be assigned to an individual cell is to compute position and velocity from navigation sensor input data. This sub-task may be part of a larger navigational task assigned to a group. The results of this individual cell's sub-task may be used to initiate the start of another sub-task that might be in another cell, for example, update Kalman filter data for the state vector computation.

In addition to the groups of cells in Figure 5-5, the bulk memory shown is used only for loading programs and data since the cells themselves provide all the main system memory. The conditioners and sensors are shown to represent the peripheral equipment and controllers connected to the computation system.

The cells in a group and in the entire system are all identical in terms of hardware. However, at any particular time some cells will be functionally operated differently from other cells. Each group will have one cell designated as a controller

(1) A global program is simply a program executed by global control of a number of cells.

cell. The remaining cells in the group will be designated as working cells. The cell that is the controller cell will be responsible for controlling the inter-cell communication bus and providing the executive control of the group's operation. All the cells in a group are connected to the inter-cell communication bus. This bus will be a 1/2 word parallel and serves as the primary means of communication among the cells in a group. The inter-cell bus will be used for communicating data between cells and also for the transmission of global instructions and commands to cells. The cell that has the unique function of the controller cell will control the use of the inter-cell bus and will provide all the global instructions and commands.

The working cells in a group can either accept and execute global instructions that are sent on the inter-cell bus or they may fetch and execute instructions from their own memory. In fact, some cells may be using the global instructions from the inter-cell bus while others are using instructions from their own individual memory. Since all the cells are physically identical, this operation will be described by stating that the cells are capable of existing in different states functionally. In this way, both local and global control may be carried out simultaneously within a group. As a result, both the natural and applied parallelism inherent in a group's tasks may be efficiently carried out. As was noted above, the controller cell will control the inter-cell bus. The functions to be carried out on the inter-cell bus include communication of data between cells, the sending of global instructions and commands, and the sending of commands to individual cells. As mentioned above, only cells operating under global control will be using global instructions and commands from the inter-cell bus. However, all the cells in the group will be responsive to commands sent to individual cells via the inter-cell bus regardless of whether the cell is operating under local or global control (these commands are addressed to one particular individual cell). It may, therefore, be seen that the controller cell can command one or more cells operating under global control simultaneously while cells operating under local control must be commanded or talked to on an individual basis.

Global control implies that one or more cells are using this type of control. The controller cell provides all control information sent on the inter-cell bus. It may therefore be seen that cells that are being operated in a global control mode are highly dependent on the controller cell in that they are receiving instructions from it and are also responsive to its global commands. However, the cell operated in a local control mode is considerably more independent of the controller cell in that it fetches its instructions from its own memory and must be talked to or commanded individually. It can be seen that the controller cell has the overall control of the cells in a group and the degree of this control can be varied depending on the functional use of each cell. The above discussion is intended to serve only as a brief introduction to the operation of a group of cells, a considerably more detailed explanation will be given in Chapter 6.

A simplified block diagram of a cell is shown in Figure 5-4. Each cell is planned, as mentioned earlier, to be a single MOS/SOS chip, although, depending on the cell memory requirements for certain applications, more than one chip could be used. In fact, initial prototype systems could be constructed using more than one chip to make up a cell (see Section 3 on Technology). The cell consists of a memory and arithmetic and control section in a similar manner to conventional computers. In addition, the cell will contain logic for inter-cell bus communications and for its own identification; this is the part of the cell that differs significantly from conventional computers.

All storage within a cell is directly addressable and is divided into storage registers and a number of control registers. The storage registers hold the programs and data to be used by this cell and possibly other cells. The control registers are the general processor registers used in the execution of instructions. Instructions, whether they come from the cell's own memory or from the inter-cell bus, are decoded and executed in a manner similar to conventional computers.

In addition to the inter-cell bus communications path, another means of communication is provided for, this is the neighbor communications. Each cell contains a single serial connection to each of its 4 adjacent neighbors (north, east, south, and west), with wrap around connections at the edges of the array of cells. This communication means is primarily included to facilitate the global programming of a group where small amounts of data are required to be communicated between cells; e.g., a matrix multiply utilizing applied parallelism.

Storage in the cells of the distributed processing system is volatile semi-conductor memory; however, investigation of this feature showed that it should cause no problems. The primary power supply would be backed up and could be fed into the distributed processor on separate, isolated lines. These lines could then be connected to the computation system through separate voltage regulators with diode isolation. If one of the supplies were to go out of tolerance, the other would simply take over. In this manner, the power supply could be made more reliable than the computation system hardware. Another simple possibility would be to use presently available small Nickel-Cadmium batteries to provide backup power for the computation system. This would be practical since a DAMP System as discussed above would only dissipate a few watts of power.

The above discussion briefly describes the mechanization of a cell and how the cells can operate. The cell that operates as the controller cell will either execute or transmit instructions contained in its own memory; instructions identified as global instructions will simply be sent on the inter-cell bus for execution in cells operating under global control. In addition, to the functions outlined previously, the controller cell will contain various executive routines to control the operation of the group and the execution of the assigned tasks to the group.

The operation of a group was briefly outlined above; it was seen that the cells were capable of operating under either local control or global control. The organization consists of a number of groups interconnected by an inter-group communication bus as shown in Figure 5-5. Each of the groups can operate independently and simultaneously of the other groups. Therefore, this introduces another level of parallelism in the organization in that it is possible to simultaneously have more than one group in operation where each group may be utilizing local and global control to carry out its tasks. Each group is connected to the inter-group bus by a group switch as shown in Figure 5-5. The group switch will be controlled by the controller cell of the group.

One of the groups will contain additional functions in its controller cell that enables it to operate as the system executive group. This will primarily involve coordinating and controlling the communications that take place on the inter-group bus. Since all the cells are identical, the system executive can be located in any one or more groups. It is responsible for the following tasks: (1) Controlling the inter-group bus, (2) handling communication with the bulk storage unit, (3) handling data communication from group to group, (4) sending out commands to load the system, and (5)

allocating I/O time on the inter-group bus. The system executive will be discussed in Section 11 along with all the other software and executive aspects of the organization.

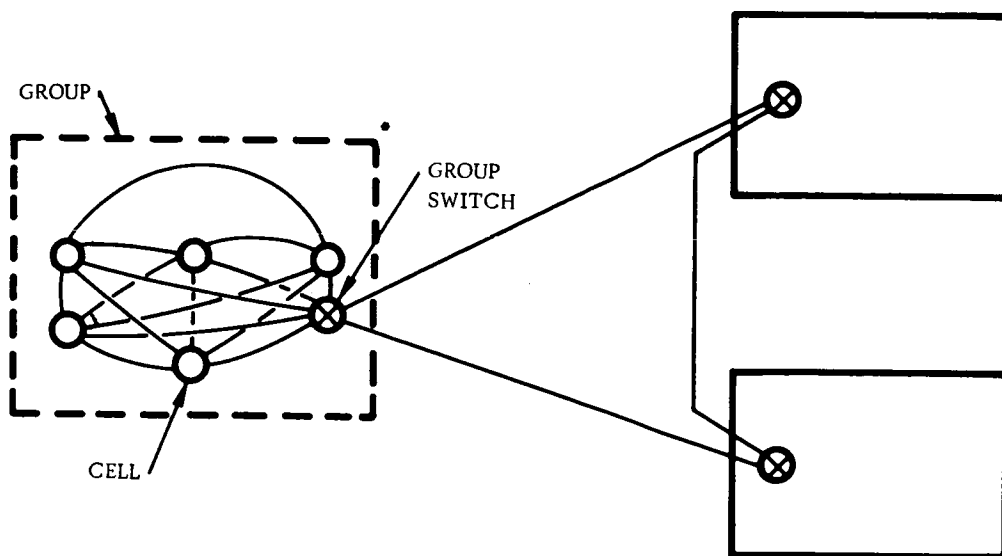
There are many factors in determining the number of cells in a group and the number of groups to be used. One of the influences on the number of cells in a group is the capacity requirements of the various independent system functions. For example, a task with very little parallelism could require many cells for its execution due to the need for a great deal of instruction and data storage. Another important influence on the group size is the degree of applied and natural parallelism within a task. These points are also discussed in Section 4 on parallelism.

From the requirements given in Section 2 and the parallelism studies of Section 4, it has been determined to use 4 groups of 20 cells each (512 words/cell). It should be noted that the system could be designed using a variable number of cells per each group; however, there is at least some impetus toward using fixed size groups. Such a structure is more symmetrical and as a result more capable of many levels of graceful degradation in small increments. For example, in a structure with 3 groups of 30, 15, and 15 cells, a failure of the bus in the group of 30 would eliminate one-half of the system computation power. Admittedly this will be an unlikely failure mode, but the situation is certainly not ideal.

Finally, it should be noted that input/output connections are provided directly to each cell and will be connected to the appropriate sensors and conditioners (some may not be used if the cell gets I/O over the bus). In addition, I/O devices may also be connected directly to the communication busses in the system. This section has served to introduce the general concepts of the DAMP System and its general operation. The remaining sections of this report will present detailed descriptions of features and operation of the system. The next section on Group Architecture will clarify the detailed operation of the group and many points presented here.

5.3 COMMUNICATION STRUCTURES

There are a large number of possible schemes for communication amongst the cells (and/or groups) in a distributed processor. For example, the Holland machine shown in Figure 5-2 is structured so that each cell can communicate directly with its four neighbors. This scheme is satisfactory as long as the communication amongst cells can be highly localized. However, if one cell must communicate instructions or data to another cell that is not located physically near, path building problems would generally result so that the communication delays may be intolerably long. In particular the DAMP System's use of global control requires some long communication paths; as a result, a simple neighbor communication scheme is not sufficient. There are other more complex neighbor communication schemes, such as the cobweb array connections shown in Figure 5-1, but these schemes still suffer from path building problems and are therefore not applicable to the DAMP System. The logical extension of the neighbor communication schemes would be a full inter-communication structure similar to that used in a multiprocessor system. Such a structure, shown in Figure 5-6, would provide no communication delays since each cell can communicate directly with any other cell. However, for a group of reasonable size (20 cells or so), the number of connections and extra circuitry required make the system impractical. This is especially true since the high communication rates obtainable with this system are not required. The Solomon system uses neighbor communication for data and a bus from the central control unit to all the cells for



*ONLY FIVE CELLS SHOWN IN THIS EXAMPLE.

Figure 5-6. Full Intercommunication Distributed Processor

the global instructions. This type of communication is well suited to the needs of a global structure with a central control unit. However, a global-local structure where any cell can act as the controller (the DAMP System) required a communication system of somewhat greater complexity than that of the Solomon machine. Two alternative structures are discussed.

Presently the most promising communication structure for the DAMP System is that shown earlier in Figure 5-5. This structure uses a common central bus in each group with each cell connected directly to the bus through a driver/receiver circuit. The bus and connections are byte parallel. The driver/receiver circuits in each cell will be designed such that a single component failure in these circuits will not short the bus directly to ground or to the supply. This will make the probability very small that a driver/receiver will fail and short out a complete group. It should be noted that this system does not require redundant driver/receiver circuits in the sense that a second circuit should be available within a cell if the first one fails. The only requirement is that a failure of the cell's driver/receiver does not short the common bus. Each cell will also carry out self checks such that if a failure is detected a flip-flop will be set that will inhibit driving the common bus. In addition, each cell will have a connection to each of its four neighbor cells. This will be a single serial line since its use will be at a relatively low rate (Section 6 will present reasons for including neighbor communications).

The foregoing communication structure between cells within a group would be all that is required if only one group of cells was used in the DAMP System. However, for reliability purposes a second central bus would have to be added thus doubling the

number of connections in the system. The approach taken for the DAMP computer is to subdivide the system into groups where each group contains one inter-cell bus connected as described above. This approach requires less total connections in the system, provides for more failures to occur before bringing the system down, and enables global programs to be flexibly scheduled so that the minimum number of cells will be required to execute a given set of requirements. (Dividing the cells into groups also significantly eases the problem of assigning tasks to cells.) The latter point is discussed in Section 4 on parallelism.

A group switch is used in order to connect together the buses within each group and still insure that all groups can simultaneously communicate within themselves. Each switch, shown in Figure 5-5, is connected to its inter-cell bus in the same manner as a cell; it is in turn connected to the inter-group bus that provides communication between all group switches. This switch then looks like another cell to the group controller cell since it must be monitored for communication needs and must be sent words that go to other groups. The group switch has the following tasks: it honors requests from other groups for communication to its group and it receives information from its group and places it on the inter-group bus.

A little reflection will show that the group switch is a single failure point for a group since if it fails a group can no longer communicate with some I/O devices, other groups, or the bulk storage unit. As a result the switch must contain some redundancy and must have a failure rate significantly lower than that of a cell. This low failure rate will be achieved by using redundancy within the chip that carries out the group switch function and by using redundant group switches in each group if necessary. It should be noted that this low failure rate should be relatively easy to achieve since the complexity of the group switch chip is significantly less than that of the cell wafer. It should also be noted that the central inter-group bus will have to be made redundant since a single failure of this bus would bring down the system.

The alternative to the chosen communication structure is that shown in Figure 5-7. This structure uses a new element, the arbitor, to handle communication between cells and between groups. The arbitor is connected to each cell on two-way byte parallel lines and to the arbitors of other groups on the same type of lines via a central bus. The arbitor simply carries out a round-robin (circular) scan of requests for communication from each cell or from another arbitor. When a request is found up, the arbitor reads a header word and connects the requesting cell or arbitor to the requested cell (or central bus). The simplest system would only allow one set of connections at a time, but would prepare for the next set of connections while two cells are communicating. (A more complex arbitor providing for more than one simultaneous cell to cell communication link could be used if this increased communication rate was necessary; however, such an arbitor would have somewhat lower reliability due to greater complexity.) The main advantage of the arbitor structure is that it would decrease the overhead associated with obtaining a communication link between cells. This is true since in the chosen structure with a group switch, the controller cell must sequentially monitor each cell for communication requests. This process requires words to be sent between each cell and the controller; therefore, if one cell was to make two requests in a row while no other cell was requesting service, a sizable delay would occur between obtaining the first and second requests (all the other cells in the group would have to be monitored in between). In this case the arbitor method of scanning request lines will be considerably faster. Of course how much time savings this amounts to depends on the communication rates necessary on

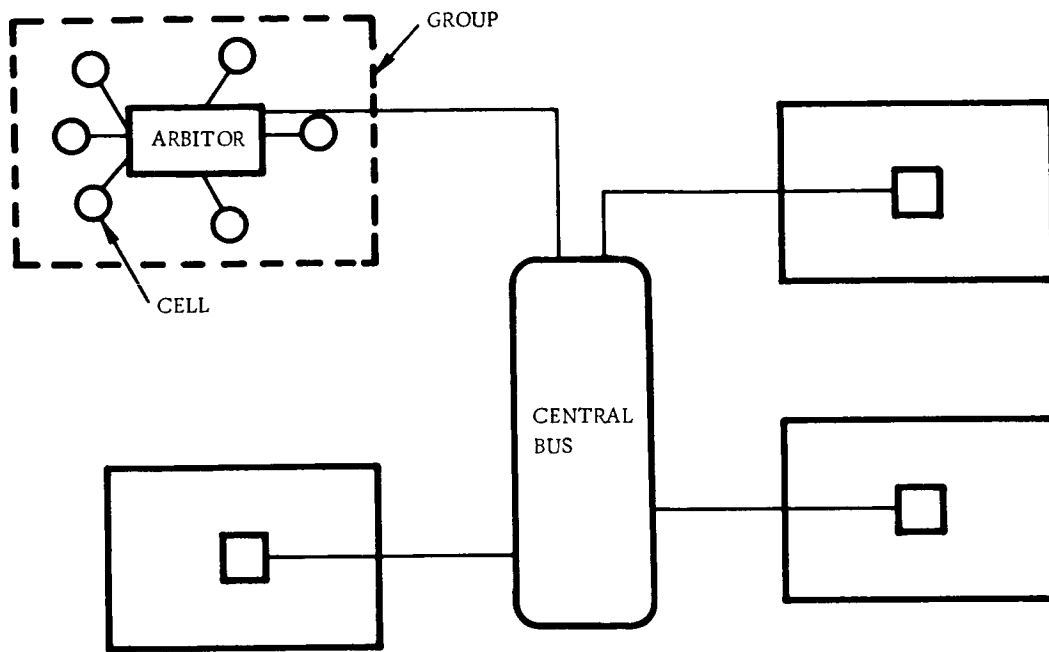


Figure 5-7. Arbitor Communication Structure

the buses. Another possible advantage of the arbitor is that additional hardware could also enable it to ignore requests from cells with failed communication lines or drivers. The real disadvantage of the arbitor structure is that it is a single failure point for a group. It will clearly be fairly complex (a few thousand FET's) and have a large number of external connections. As a result, its reliability will be significantly lower than that of the simple common bus used in the chosen communication structure. (This bus is also a single failure point within a group.) Since the arbitor also handles the task of the group switch it should be compared to it too. Again the arbitor is much more complex and has more connections. Of even more importance is the fact that although the group switch could be connected redundantly into the group bus if necessary to increase its reliability, it is relatively impractical to use two or more arbitors in each group since that would at least double all the connections in the system.

The preceding discussion points out the reasons behind the chosen communication structure for the DAMP System (Figure 5-5). The important point behind the choice was that the bus structure with a group switch offered considerably higher reliability; however, the calculations were not made to obtain the actual relative reliability of the arbitor scheme.

6 GROUP ARCHITECTURE

6.1 INTRODUCTION

Many different computer systems have been studied during the course of this study. A description of these systems was given in Section 5 of this report. The distributed array memory and processor system was found to be the most useful for the general computations needed on future spaceflights. This distributed system, shown in Figure 1-1, requires a unique architecture to make a capable and reliable system.

Architecture means the combining of software and hardware features to make a balanced useful system that will meet the requirements set upon the computing system. Some of the considerations, such as memory size and approximate processor capability, are based upon the ground rule to build a cell upon a single wafer. This section describing the group architecture will describe the features desirable to unify the cells into a working group.

The distributed processor system consists of groups, which are made up of cells. Because the cells in a group are connected by neighbor communication lines and the controller cell can send global instructions to cells, the group is the fundamental unit of the computing system. The software studies to date indicate the compiler must be aware of the cell memory contents, the cell bus loading, and the controller cell capabilities when compiling programs. For these reasons the architectural studies were applied to the group.

The features and characteristics of a group are described here. All these features may not be needed; future studies are needed to determine the useful features to be retained and the features of little value to be discarded.

6.2 CELL STATES

A fundamental ground rule has been to make all cells of identical hardware. Although all the cells are identical in hardware, a cell always exists functionally in one of seven different and mutually exclusive states. These states are listed in Table 6-1.

A permanently failed cell is placed in state 1 by a combination of software and hardware controls. These cells will not be used again.

State 2 is the power saving state for cells that are not needed presently. If standby power is applied to the identification registers and the cell bus gates, the main power to this cell may be turned on by the controller cell and switched to another state. The controller cell then may reload the cell's memory. If all the power has been switched off, a special restart procedure, using the neighbor communication lines, must be used.

Table 6-1. Cell States

1. Permanently failed - power off
2. Shut down - power saving state
3. Independent
4. Dependent under global control (Global State)
5. Dependent under local control
6. Dependent in wait state
7. Controller cell

Independent cells are functionally similar to a conventional computer. These cells fetch all instructions and operands from their memories. The cell that is in the independent state stays in this state until the controller cell sends a command on the inter-cell bus commanding this cell to change states. The independent cells can process problems that are not amenable to global processing.

Dependent cells respond to global instructions and global level commands sent out from the controller cell. A dependent cell exists in one of the states 4, 5 or 6. Which of the three depends upon the level of instructions being sent from the controller cell and the cell's level register contents. The concept of levels is described later in the next section under cell identification.

The concept of having both independent cells and dependent cells in a computer system is an important concept developed in this study. Other studies of similar computer systems require all cells to be independent or all dependent. With this improved system, problems may be solved more efficiently by using both independent and dependent cells as was noted previously.

A dependent cell in the global state (also called the active state) is receiving instructions from the controller cell via the inter-cell bus.

As mentioned above the instructions being sent from the controller cell to dependent cells are identified as being at a certain level. A dependent cell that is not at the proper level to receive global instructions can idle and not execute instructions. This is the wait state. Therefore, if the controller cell is servicing certain dependent cells, other dependent cells may wait their turn for service.

A dependent cell, instead of waiting for the controller cell to send the instructions for its level, may fetch and execute instructions from its own memory. This is the local control state of a dependent cell. This state may appear to be quite similar to the independent state of a cell in that both operate in a local control mode. The basic difference between them is that the dependent local control state implies that the cell is dependent and dependent cells respond to certain global commands on the inter-cell bus. Therefore, the dependent local cells are under more control of the controller cell than the independent cells are. This will be clarified later in this chapter. This state has advantages when one wants to use a cell partly for global programs and partly for local programs.

The capability of dependent cells to use local control means that the cell bus is not wasted on sending instructions when the instructions could be better stored in the cell's memory. With this feature, the cells can efficiently use local programs to correct for bad data and handle exceptional conditions. The cell can enter the dependent local control state, do some processing, and later inform the controller cell of the situation. It should be kept in mind that even though the cell may use local control in state 6, it is still a dependent cell. This means that it is basically under control of the controller cell and will respond to certain global commands.

The seventh state of a cell is the controller state. In this state, a cell controls the cell bus and may issue global instructions. The fundamental ground rule of making all the cells of the same hardware allows any cell to become a controller cell. This gives the advantage that the controller cell functions may be switched among several cells. Thus there is no requirement that all the executive and controller programs fit in one cell. At any time, there is only one cell in the controller cell state in a group. The reason is the controller cell controls the cell bus, and two cells cannot be allowed to issue conflicting commands. Software and hardware interlocks will be used to insure only one cell is in the controller cell state in a group.

The group switch, shown in Figure 1-1, is part of the group although it is not a cell. The group switch, like a cell, has an ID register. The group switch responds to control words containing the proper ID bits and will perform the operation given in the control word (CW). Thus the controller cell will operate the group switch.

6.3 CELL IDENTIFICATION

The distributed processor computer system has two methods for identifying the individual cells. One method identifies the cells by a level. A cell will be identified as being at one of eight levels. The other method results in each cell being given an identifier, also known as the cell address. Thus a cell has two "names", a common first name (level) and a unique last name (identifier). This concept of having two names is important when discussing the dependent and independent cells.

Independent cells use only one name, their identifier or cell address. The level (or first name) is not used and, although present in a level register, has no meaning unless the cell later assumes a dependent state.

Dependent cells use both names. The controller cell can send out information using a first name (level number) to all the dependent cells; all the dependent cells at this level will respond. If a last name (cell address) is sent, only the cell with this name will respond since each cell has a unique last name regardless of state. Thus it can be seen that the controller cell must communicate with independent cells individually or to more than one at the same time since the dependent cells respond to levels and more than one cell may have the same level.

The global instructions that are sent by the controller cell on the inter-cell bus to dependent cells always follow the sending of a name. The cells that responded to the name will receive the instructions. For example, assume a system with 7 cells as follows:

<u>First Name</u>	<u>Last Name</u>	<u>Dependent</u>	<u>Independent</u>
JOE	SCOTT	x	
BOB	ROSE	x	
HELEN	TRUMP		x
BOB	MILLER	x	
BOB	JOHNSON	x	
JOE	SMITH	x	
HELEN	DAVIS	x	

As an example assume the controller cell sends the following instruction groups. The results are explained below.

JOE: Load X, Store Y, BOB: Load A, Add X, Subtract B, Store Y,
HELEN: Load A, Store Y, ROSE: Add M, Add N, ...

Two cells (JOE) will Load X, Store Y. Three cells (BOB) will execute the next four instructions. One cell will execute the next two instructions. (The cell TRUMP is an independent cell and does not respond to first names.) The name ROSE is a last name, thus only one cell will execute the last group of instructions.

6.4 SOURCE OF INSTRUCTIONS

The traditional computer has instructions stored in a memory which is always available to the processor. The processor controls the instruction fetch sequence by using the program counter. In most modern machines, the instructions are

located in a random access core memory, and the program counter is incremented to fetch sequential instructions. A jump is performed by loading the program counter with the address of the next desired instruction.

Independent Cells:

The independent cell receives all of its instructions from the cell's memory, like the traditional computer. The program counter is used to control the fetch of instructions.

Dependent Under Global Control:

The dependent cell in the global state gets its instructions from the controller cell. These cells receive the instructions from the inter-cell bus and then execute them. The controller cell precedes the instructions with a name. The name (level number) is contained in a control word sent on the inter-cell bus. This control word is a prefix to a group of instructions. This prefix is the level of all instructions until a new level prefix is sent or another control instruction is sent. Thus, a group of global instructions is defined as that information contained between control words on the inter-cell bus when a certain type of control word precedes the group of instructions. The group of global instructions is variable in length. It will be mentioned here that the cell will execute the instructions in a manner quite similar to independent cells; the primary difference being that the source of the instructions is the inter-cell bus and not the cell's own memory. The particular details of the instruction execution for each state of a cell will be given later in this chapter.

A detailed discussion of the inter-cell bus operation will be given later in this report. However, in order to facilitate the discussion here a brief discussion will be given here. The bus is a half-word (8 bits) parallel. In addition one extra line is used and designated a control line; this line is used to define control words or commands on the bus. The commands or control words utilize levels or addresses (first or last names) to identify which cell or cells the command or control word is intended for.

Every dependent cell compares the level prefix sent by the controller cell to the level register contents contained in the cell. If the prefix and the level register contents are different, the cell ignores all the instructions, data, etc sent by the controller cell, until a new level prefix (or other control word) is put on the bus by the controller cell.

An example is given in Figure 6-1. Note that every cell is required to examine every control word, but will not perform the control word operation if the cell is at a different level (only the dependent cells examine the level), or has the wrong ID (cell address).

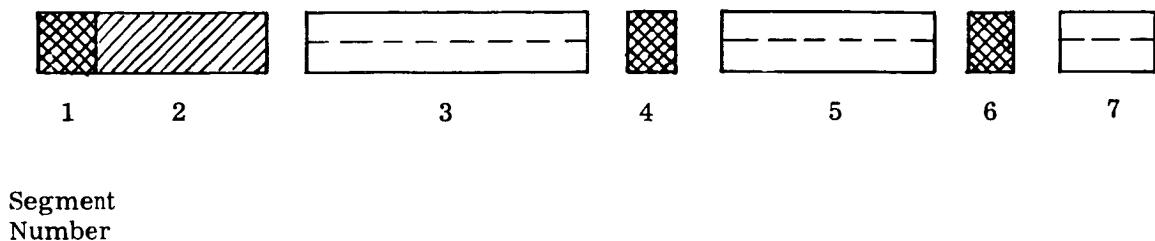


Figure 6-1. Bus Operation Example

-  : Control Byte (CB) All cells will examine this byte. If the cell(s) matches this CB (either by level or cell address), the cell(s) will receive the control word.
-  : Control Word (CW) This word includes the CB, and defines an operation to be performed by the cell. Often the CW consists of only a CB.
-  : Data. In this example, it shall be assumed that the Control Word specified that instructions are contained here (note that the number of instructions can be variable).

When segment 1 in the example occurs, all the cells in the system will examine the CB. Assume the CB is a type that specifies a level. Thus, all the dependent cells at this level will be ready to receive the CW (segment 2) and are automatically placed in the dependent active (global) state. These cells in the global state will receive the CW (segment 2) and will receive the instructions and/or data following (segment 3). No other cells will receive any instructions or data (segment 3) from the bus until the next CB occurs (segment 4 in the example).

When segment 4 comes on the bus, all the cells will again examine the control byte. In the example, it shall be assumed the CB specifies a different level. The following actions will occur:

In dependent cells that were active (global state) the CB at a new level will set these cells to the dependent wait state.

In dependent cells that were not active and are at the new level indicated in the new CB (segment 4), these cells will become active and will receive the data (instructions) following (segment 5). All other cells are left unchanged.

Thus, it can be seen that many sequences of global instructions may be sent to many sets of cells at very low overhead cost to switch between sets. The low overhead is advantageous when many cells are at each level and short sequences of instructions are to be transmitted to each. Also the bus is used very efficiently. This is primarily possible because of having the dependent local and dependent wait state in addition to the dependent global state. Since dependent cells respond to level commands, it is possible to switch many cells quickly and efficiently from one dependent state to another dependent state.

To summarize, a dependent active (global) cell is a cell that is receiving global instructions and data. By definition, a global cell is at the same level as the global instructions being sent on the bus. Actually, the level is in the prefix CB, there is no level transmitted with each instruction. The term "global instructions level" will refer to this prefix, although the term is not exactly correct.

Dependent Under Local Control:

The dependent cell not receiving instructions from the inter-cell bus may fetch instructions from its own memory. This cell is in the dependent local control state. The fetch of instructions is identical to an independent cell. The difference in this cell being that it is constantly examining the bus for a command at its level as noted in the above example.

Controller Cell:

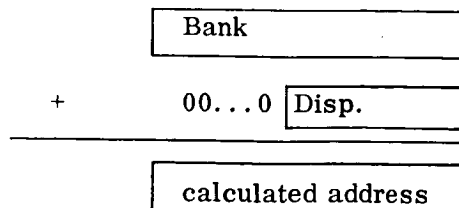
The controller cell always fetches instructions from its own memory. The instructions destined to be executed by the dependent global cells are not executed by the controller cell. All other instructions are executed by the controller cell and are not sent to the global cells. This is explained further in the section describing the controller cell instruction execution (Section 6.7).

6.5 SOURCES OF ADDRESSES

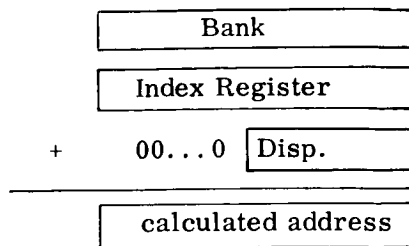
The computer technology has developed over the years many ways of specifying a memory address. The early machines had the operand address given in the instruction. Later, index registers were used to modify the instruction address and memory banks were used to save instruction bits. The traditional computer had several ways of determining the final (or effective) address that was used to address memory.

The cells in the distributed processor computer also have several ways to specify an address. All the ways will be described here, although some are not used by cells in certain states.

The address may be specified by adding the instruction displacement and the bank register (also known as a base register). The bank register is 9 bits long since a cell will have 512 words of storage. The displacement is obtained from the instruction. Since the displacement is less than 9 bits, a bank register will always be added to the displacement when a full (9 bit) address is required.



The sum is called here the calculated address. If an index register is specified, it is also added.



These two calculated addresses use the registers located in the cell.

Independent cells will obtain all the parameters that make up the address from the cell itself. Dependent global cells will obtain the displacement from the instruction that was sent on the inter-cell bus, the bank and index register are always from the cell itself.

In addition to the calculated address, a new concept of a given address is used. A given address is an address that is used instead of the calculated address. This will be explained below.

A dependent global cell recognizes a given address by a special control instruction received on the inter-cell bus. This special instruction is called a GC instruction. The GC instruction is sent from the controller cell to signal the dependent global cells that a given address, in addition to an instruction, is to be sent on the cell bus. This particular GC instruction is called a format instruction. It will be seen later that there are several types of format instructions, the particular one under discussion here is called a format-given address. The sequence is as follows:

<u>Time</u>	<u>Contents of Cell Bus</u>	<u>Length</u>
↓	GC Instruction	(8 bits)
	Format-(given address)	(16 bits)
	Given address	(16 bits)
	subsequent instructions	(16 bits)
	.	
	.	
	.	

Section 10 contains a detailed description of the inter-cell bus, however it will be mentioned here that the bus is 8 bits in parallel (1/2 word). The global cell normally expects to receive 16-bit instructions. However, this normal sequence is altered by a format instruction. This instruction is a control byte and tells the dependent global cells that something new has been added. In this case, that a 16-bit address follows the next 16-bit instruction. The global cell will execute the instruction using the given address instead of the calculated address. Thus the controller cell may send an address to all the global cells instead of having the cells calculate the address.

The independent cells (and the dependent cells under local control) may use the GC instruction. In this case, the format instruction is located in the cell's memory. After the format instruction is executed, the processor knows the type of data contained in the following memory locations. An example is given below.

<u>Location</u>	<u>Contents</u>	<u>Length</u>
START	GC Format Instruction- address is given	16 bits
+1	Instruction	16 bits
+2	Given Address (subsequent instructions)	16 bits 16 bits
	.	
	.	
	.	

Here, the instruction at START +1 is executed using the given address instead of a calculated address. The use of the GC format instruction is called instruction modification. The modification is usually not a change in the operation of the instruction, but rather a respecification of the address. As seen from the above examples this modification can be done in any cell that executes an instruction.

6.6 SOURCES OF DATA

The cell in the distributed processor computer system can obtain data from many sources. Some sources are available to all cells irregardless of their state, others are available only to cells in a particular state.

All cells have access to data stored in their memory. The present cell concept has no division of memory into data areas, read-only areas, etc. Thus any location in a cell is available to the processor.

All cells may obtain data from their neighbors. The neighbor to neighbor data transfer is independent of the cell state, and is described separately in Section 6.8 and 9 of this report.

Cells may receive data from outside the group via the cells' I/O line. This system is described in the section on Input/Output Operation (Section 7.1).

Cells may receive data from outside the group or from other cells in the group via the inter-cell bus. This system of a cell communicating directly with another cell via the inter-cell bus is described in the section on the communication bus operation (Section 10).

Some special cases of how a dependent global cell can receive data over the inter-cell bus will be given below. Dependent global cells may receive data from the controller cell. A GC format instruction may be used to indicate to these cells that data is being transmitted in addition to instructions. The GC format instruction is used in a similar manner to that described in the preceding section, Sources of Addresses. In the preceding section, the format instruction was called, given address. The format instructions described below will deal with data and will be identified differently as will be explained below.

A 16-bit data word may be sent to dependent global cells by sending the following sequence on the inter-cell bus. The GC format instruction is called a D16 format:

<u>Time</u>	<u>Contents of Cell Bus</u>	<u>Length</u>
↓	GC Format - 16 bit data follows	8 bits
	Instruction	16 bits
	Data	16 bits
	(subsequent instructions)	16 bits
	.	.

This format instruction indicates an instruction is followed by a data word of 16 bits. The instruction is executed by the cell. The operand used, however, will be the data word received from the inter-cell bus and not the data word usually fetched from memory. More details concerning the operands and data are given in the section on instruction execution (Section 6.7). Having data sent by the controller cell means that the individual cells do not each have to store constants. To have 20 cells all store pi, e, and other constants is an inefficient use of cell memory, whereas the controller cell has to store the constant but once and send it out when it is needed. The constants are sent at the time they are used; thus they need not be saved in the individual cells memory.

A 32-bit data word may be sent to dependent global cells. The GC format instruction is now called a D32 format:

<u>Time</u>	<u>Contents of Cell Bus</u>	<u>Length</u>
↓	GC Format - 32 bit data follows	8 bits
	Instruction	16 bits
	Data	32 bits
	(subsequent instructions)	16 bits
	.	.

This situation is similar to the D16 format. The instruction will use the 32-bit data word in performing its operation.

It is possible to have the dependent global cells use as data, the displacement field in the instruction. Naturally, the magnitude that may be sent depends upon the length of the displacement field in the instruction. This is accomplished by using another format instruction called the I (Immediate) format. The I format is especially useful when loading registers with small values. The I format is received by a dependent global cell as shown below:

<u>Time</u>	<u>Contents of Cell Bus</u>	<u>Length</u>
↓	GC Format-Immediate	8 bits
	Instruction	16 bits
	(subsequent instructions)	16 bits
	.	.

As an example, if the instruction is a Load Index Register 3, the displacement field of the instruction, preceded by zeros, will be loaded into index register 3.

The last GC format instruction that concerns data is the DS format. This is a very special format whose usefulness is yet to be entirely determined. It was designed to rapidly move data from the controller cell to a number of cells or a cell. The sequence received by a dependent cell is as follows:

<u>Time</u>	<u>Instruction and Data</u>	<u>Length</u>
↓	GC format - DS	8 bits
	Instruction	16 bits
	address	16 bits
	data word 1	16 bits
	data word 2	16 bits
	data word 3	16 bits
	.	
	.	
	.	
	data word N	16 bits
	GC format-End of DS	8 bits

The dependent global cells will receive the DS format instruction. The instruction following will be executed using the given address and the first data word. The given address will be incremented by one, and the instruction will be repeated using the second data word. The operation will continue until the GC format byte, End of DS, is received instead of a data word at N+1. The DS is seen to be useful if the instruction is a store or compare to memory type of instruction.

The above description of instruction modifiers (GC format instructions) to allow the controller cell to send data to the dependent cells applies to dependent global cells. The instruction modifiers may also be used by cells in other states. Of course, the format instructions must then be stored in the cell's memory, and are not sent over the intercell bus. Section 6.7 describing the instruction execution should be consulted for more details. An example of how one of the above GC format instructions can be stored in a cell's memory and used to modify instructions is given below:

<u>Location</u>	<u>Contents</u>	<u>Length</u>
START	GC Format-D16 data follows	16 bits
+1	Instruction - Load Acc 1	16 bits
+2	data word 1	16 bits
+3	Instruction - Load Acc 2	16 bits
+4	(subsequent instructions)	16 bits
	.	
	.	
	.	

The instruction at location START is a CC instruction that will generate a GC Format - D16 and indicates that the next instruction is followed by a word of data. The load accumulator 1 instruction will load accumulator 1 not with the contents of the memory location specified by the calculated address but with data word 1 located at START +2. It is seen that the GC format instruction is used here to respecify the location of the data to be loaded into the accumulator. The instruction at START +3, because it is unmodified, is executed in a normal manner.

MOST instructions may be modified by a GC format instruction. This will be explained in the next section where table 6-3 will be presented that gives a list of all the instruction types and how they are affected by modification.

6.7 EXECUTION OF INSTRUCTIONS

The instruction execution in the distributed processor computer system is a complex subject. The execution depends upon the state of the cells and upon where the addresses, instructions, and data are located. This section does not contain the detailed instruction list or details on the operations in a cell to execute the instructions (see Section 9); the intent of this section is to present the general concepts on how the different types of instructions are treated. Instead of first presenting the detailed processor section of the cells, it will be stated that it contains the program counter, instruction decoding logic, adder and several registers as in a conventional machine. The registers are accumulators, index registers, and base registers. The registers may be located in an addressable section of the cell's memory, or they may not be addressable by the programmer. The cell also contains a cell address register and a level register for purposes of cell identification as discussed in Section 6.3.

Instead of treating each instruction, the instruction list was divided into several general categories as shown in Table 6-2. The instructions in a category are executed quite similarly in most states, however the differences will be pointed out in this section. An explanation of the execution for each of the categories will be given below for each state of a cell. Table 6-3 contains a summary of the execution of instructions.

Two instruction categories have been defined that are quite unique, namely (1) CC (Controller Cell Instructions) and (2) GC (Global Control Instructions). Actually both of these categories are mechanized with only one operation code in the processor by using an operation code extension scheme. However, they will be considered separately for purposes of clarity. Table 6-4 summarizes the types of CC instructions and Table 6-5 summarizes the types of GC instructions. The term instruction is used freely here and may not be exactly correct. The CC instructions are conventional type instructions in that they are stored in the cell's memory that executes them (this will normally be the controller cell). However, GC instructions are sent over the inter-cell bus to the cells by the controller cell; the controller cell can send out these GC instructions only by executing a CC instruction. This area of discussion is quite complex and the explanation of instruction execution in this section should serve to clarify it. In addition, the types of CC and GC instructions will be explained in this section (some of the GC - format instructions have already been discussed in this chapter).

A summary of the GC and CC instruction execution is given in Table 6-6 for each of the cell states. This table will be explained in detail in the remainder of this chapter. It should be noted that in Table 6-6, that the communications bus I/O commands, C, are listed as GC instructions while not being listed in Table 6-5. This is due to the fact that they are sent by the controller cell by the execution of a CC instruction therein in much the same manner as those of Table 6-5 are. They are not truly GC instructions but rather inter-cell bus I/O commands. They are not discussed in this chapter; chapter 10 contains the detailed mechanization of these commands and the operation of the inter-cell bus.

Table 6-2. Instruction Categories

1. LR	Load Register from a memory location
2. STR	Store Register into a memory location
3. OPR	An operation is performed between a register and a memory location contents, the results are in a register.
4. RR	An operation is performed between one register and another register.
5. R	Single register operation, such as shift.
6. EXEC	Execute an instruction in a memory location.
7. COMP	Compare the contents of a memory location (or register) with a register. The results of the comparison are saved in the COMPARISON flip-flops.
8. SKIP	Test the contents of a memory location (or register) with a register or implied value. The result is true or false.
9. JUMP	A new sequence of instruction is begun. The jump may be combined with a test to make a conditional jump.
10. CC	Controller Cell instructions.
11. GC	Global Control instructions. These instructions control the states and levels of all cells and dependent cell execution of global instructions.
12. IO	Input-Output instructions. These instructions initiate and control I/O operations.

Table 6-3. Summary of Instruction Execution

Instruction Category	Cell State	Source of Instruction	Source of Address	Operand 1	Operand 2	Notes
JUMP	DG	Cell Bus	Calc (G-M)	R ₁	R ₂ (D16, D32, I, DS-M)	Usually not executed by global cells
	DL	M				The program counter is changed. Operands are used for conditional jumps.
	Ind	M	Calc (G-M)	R ₁	R ₂ (D16, D32, I, DS-M)	"
CC	CC	M	Calc (G-M)	R ₁	R ₂ (D16, D32, I, DS-M)	"
	DG	M	None, Opcode EXT used			Not executed by dependent global cells
	DL	M	None, Opcode EXT used			Only the Format Modifier and part of the State and Level Control instructions may be executed.
GC	IND	M	None, Opcode EXT used			Only the Format Modifier and part of the State and Level Control instructions may be executed.
	CC	M	None, Opcode EXT used			All CC instructions except part of the State and Level Control may be executed
	DG	Cell Bus				All GC Instructions are valid
IO	DL	Cell Bus				State Control via Levels and State and/or Level Control of Individual Cells are valid
	IND	Cell Bus				State and/or Level Control of Individual Cells are the only GC Instructions that may be executed
	CC	Cell Bus				Invalid
Input/Output	CC					Invalid
	DG	M	Calc (G-M)	M (D16, D32, DS)		Operand 1 is I/O Data
	DL	M	Calc (G-M)	M (D16, D32, DS)		Operand 1 is I/O Data
	IND	M	Calc (G-M)	M (D16, D32, DS)		Operand 1 is I/O Data
	CC	M	Calc (G-M)	M (D16, D32, DS)		Operand 1 is I/O Data
Abbreviations used in table						
DG Dependent Global DL Dependent Local Ind Independent CC Controller Cell M Cell Memory Word R ₁ Register in a Cell R ₂ Register in a Cell (X, -Y) Modified Instruction. X's are possible alternatives, -Y is source of the alternate data or address. G Given address, used with instruction modification D16 Data - 16 bits, used with instruction modification D32 Data - 32 bits, used with instruction modification I Immediate Data, used with instruction modification DS Data List and address, used with instruction modification						

Table 6-3. (Cont)

Instruction Category	Cell State	Source of Instruction	Source of Address	Operand 1	Operand 2	Notes
LIR Load Register	DG	Cell Bus	Calc (G-Cell Bus)	M (D16, D32, I - Cell Bus)	R ₂	Operand 1 is loaded into the specified register
	DL	M	Calc (G-M)	M (D16, D32, I - M)	R ₂	"
	Ind	M	Calc (G-M)	M (D16, D32, I - M)	R ₂	"
	CC	M	Calc (G-M)	M (D16, D32, I - M)	R ₂	"
STR Store Register	DG	Cell Bus	Calc (G-Cell Bus)	M	R ₂ (D16, D32, DS-Cell Bus)	Operand 2 is stored in memory location M
	DL	M	Calc (G-M)	M	R ₂ (D16, D32, DS-M)	"
	Ind	M	Calc (G-M)	M	R ₂ (D16, D32, DS-M)	"
	CC	M	Calc (G-M)	M	R ₂ (D16, D32, DS-M)	"
OPR Operation Performed Between Memory and Register	DG	Cell Bus	Calc (G-Cell Bus)	M (D16, D32, I, DS-Cell Bus)	R ₂	The operation is performed between operand 1 and operand 2
	DL	M	Calc (G-M)	M (D16, D32, I, DS-M)	R ₂	"
	Ind	M	Calc (G-M)	M (D16, D32, I, DS-M)	R ₂	"
	CC	M	Calc (G-M)	M (D16, D32, I, DS-M)	R ₂	"
RR Register-to-Register	DG	Cell Bus	-	R ₁	R ₂	Register to Register instructions
	DL&Ind	M	-	R ₁	R ₂	"
	CC	M	-	R ₁	R ₂	"
	DG	Cell Bus	-	-	R ₂	Single register instructions
R Register	DL&Ind	M	-	-	R ₂	"
	CC	M	-	-	R ₂	"
	DG	Cell Bus	-	-	R ₂	"
	DL&Ind	M	-	-	R ₂	"
EXEC Execute	DG	Cell Bus	Calc (G-Cell Bus)	M	-	The instruction fetched from M is executed
	DL&Ind	M	Calc (G-M)	M	-	"
	CC	M	Calc (G-M)	M	-	"
	DG	Cell Bus	Calc (G-Cell Bus)	M or R ₁	R ₂ (D16, D32, I, DS-Cell Bus)	The level register may be changed by a compare, the compare flip-flops are always set.
COMP Compare	DL	M	Calc (G-M)	M or R ₁	R ₂ (D16, D32, I, DS-M)	"
	Ind	M	Calc (G-M)	M or R ₁	R ₂ (D16, D32, I, DS-M)	A compare may cause a jump, the compare flip-flops are always set.
	CC	M	Calc (G-M)	M or R ₁	R ₂ (D16, D32, I, DS-M)	"
	DG	Cell Bus	Calc (G-Cell Bus)	M or R ₁	R ₂ (D16, D32, I, DS-Cell Bus)	The level register is incremented if the Test is true.
SKIP Test and Skip	DL	M	Calc (G-M)	M or R ₁	R ₂ (D16, D32, I, DS-M)	"
	Ind	M	Calc (G-M)	M or R ₁	R ₂ (D16, D32, I, DS-M)	The next instruction is skipped if the test is true.
	CC	M	Calc (G-M)	M or R ₁	R ₂ (D16, D32, I, DS-M)	"
	DG	Cell Bus	Calc (G-Cell Bus)	M or R ₁	R ₂ (D16, D32, I, DS-Cell Bus)	"

Table 6-4. CC Instructions

1. Control inter-cell bus I/O logic
2. Generate GC and Bus I/O commands to be sent over the inter-cell bus
3. Transmit mode, single
4. Transmit mode, all
5. Execute mode, single
6. Execute mode, all
7. Do not change mode
 - 7.1 No Operation
 - 7.2 Format - - - - - (F)
 - 7.2.1 A - Given Address
 - 7.2.2 D16 - 16 bit Data Word
 - 7.2.3 D32 - 32 bit Data Word
 - 7.2.4 A, D16 - Given Address and 16 bit Data
 - 7.2.5 A, D32 - Given Address and 32 bit Data
 - 7.2.6 I - Immediate
 - 7.2.7 DS - Given Address and Data Stream
 - 7.3 State and Level Control - - - - - (SIL)
 - 7.3.1 Controller Cell State - Set Level
 - 7.3.2 Independent State - Set Level
 - 7.3.3 Independent State - No Level Change
 - 7.3.4 Dependent Wait State - Set Level
 - 7.3.5 Dependent Wait State - No Level Change

Table 6-5. GC Instructions

1. Format - - - - -	(F)
A	Given address follows the next instruction
D16	16 bit data word follows the next instruction
D32	32 bit data word follows the next instruction
A, D16	Both a 16 bit data word and given address follow the next instruction, the address comes first
A, D32	Same as A, D16 only the data word is 32 bits long
I	The displacement field of the instruction is the data
DS	A number of 16 bits words preceded by a given address follow the next instruction.
END DS	Indicates the end of DS data words
2. State Control of Dependent Cells on the basis of levels - - - - -	(SL)
Level, G	All dependent cells at this level go to global state; Instructions follow.
Level, L	All dependent cells at this level go to local control
Level, W	All dependent cells at this level go to wait state
Level, R	All dependent cells at this level reply on inter-cell bus with a constant
Level, R, DG	All dependent global cells at this level reply on inter-cell bus with a constant
Level, IND	All dependent cells at this level go the independent state
3. State and level control of Individual Cells - - - - -	(SIL)
IND, Level	The cell is made independent and level register set to the value specified
IND	The cell is made independent with no change in the level register
DG, Level	The cell is set to the dependent global state and the level register set to the value specified
DG	The cell is set to the dependent global state with no change in the level register
DW, Level	The cell is set to the dependent wait state and the level register set to the value specified
DW	The cell is set to the dependent wait state with no change in the level register
DL, Level	The cell is set to the dependent under local control state and the level register set to the value specified
DL	The cell is set to the dependent under local control state with no change in the level register
CC	The cell is made the controller cell

Table 6-6. GC-CC Instruction Execution

State	Fetch'd from Own Memory (CC Instr)	Rcv'd from Bus (GC Instr)	Sent Over Bus (GC Instr)
CC	ALL ⁽¹⁾	X	F, SIL, SL, C
IND	F, SIL ⁽²⁾	SIL, C	X
DL	F, SIL ⁽³⁾	SIL, SL, C	X
DG	X	F, SIL, SL, C	X
DW	X	SIL, SL, C	X

Legend:

X: Not Applicable

F: Format

SIL: State Control, Level Control, or both of Individual Cells

SL: State Control of Dependent Cells on the basis of Level

C: Communication Bus I/O Commands

(1): Except 7.3.2 thru 7.3.5 of Table 6-4

(2): For Independent Cells only a Level Control CC and no State Control is allowed from its own Memory (7.3.2 in Table 6-4)

(3): For Dependent Local Cells only State Control to the Dependent Wait or Independent State is allowed from its own Memory (7.3.2 thru 7.3.5 in Table 6-4 are allowed)

6.7.1 Dependent Global Cell

A dependent cell may receive instructions, data, and commands from the inter-cell bus. The global cell, or active cell, is receiving instructions and executing them as they are received; the level prefix placed before the instructions by the controller cell is the same as the contents of the level register in the global cell.

Although the global cell receives instructions from the intercell bus, the registers, addresses, and data are usually from the cell's memory. Thus several global cells will receive the same instruction, but all may use different addresses and process different data. The exceptions are indicated by the use of a GC Format modifier byte preceding the instruction. This concept was explained in the previous sections.

A description of the execution of each instruction category is given below.

1. LR instructions are all instructions that fetch an operand from a memory location and load the contents into a register. The address of the memory location is calculated by adding the base register, index register (if one is specified) and the displacement from the instruction. Only the displacement is received on the inter-cell bus. The low-order nine bits are used to address memory; the remaining seven high-order bits are ignored. Of course, if the memory in a cell were greater than 512 words, more bits would be used. The operand is fetched from this cell's memory and placed in the specified register.

The LR instructions may be modified with a GC format byte. This byte, when transmitted by the controller cell just before the LR instruction, modifies the address or the source of the operand. The A (address) modification forces the cell to use the given address instead of the calculated address. The D (data) modification forces the cell to load the register with the data word sent on the inter-cell bus. The I (immediate) modification will load the register with the displacement field of the instruction. The DS modifier is invalid.

No matter what the source of the address, the address always specifies a word in the cell's memory. Of course, the A and D modifications can not both be used with LR instructions. The controller cell may use a D modification to send the same data to all cells.

2. STR instructions store registers into memory. The address is calculated, and the contents of the specified register are placed in the addressed memory location.

A GC format byte, when received just before a STR instruction, will modify the instruction. If an A modification is used, the given address is used instead of the calculated address. A data word, either 16 or 32 bits (D16 or D32), may be specified. In this case, the register contents are ignored and are not used. The data word from the cell bus is placed in the specified memory location. Thus words may be placed directly in a cell's memory without changing register contents. Both A and D may be given. In this case, the controller cell sends out both the address in which the data is to be placed, and the data to be stored. This serendipitous result is used by the controller cell to start up cells that have had their memory cleared for some reason, such as a reconfiguration.

The DS modification, when used with a store instruction, is similar to using a GC format with an A and D. The difference is the instruction and address are not repeated with each data word, they are sent to the cell but once. Each time a data word is sent to the cell, the data is stored and the address is incremented. An End DS GC format byte will end the sequence.

The I modification can not be used with store register (STR) instructions, because the displacement field is needed for an address.

3. OPR Instructions. The address is calculated in the normal manner, using the base (and perhaps an index register), along with the displacement in the instruction. The contents of the memory location specified by the address are obtained, and are

used as operand 1. Operand 2 is always obtained from a register. The instruction specifies what operation is to be performed with the two operands. The results are always placed in a register or registers.

The following modifications are allowed.

- A A given address may be specified, which will be used instead of the calculated address.
- D A data word is sent on the bus, which becomes operand 1. No address is used. Depending upon the operation, the data word may be either 16 or 32 bits in length.
- I The displacement field from the instruction sent over the cell bus becomes operand 1. No address is used.
- DS The DS modification may be used; however, the address is not used. Because the same operation is performed with each word of data, this DS modification may not be very useful.

4. RR instructions operate exactly as in the independent state. No modifications are possible. Because both registers are in the same cell, no transmission of data by the cell bus is required. Only the instruction itself is sent on the cell bus.

5. R instructions are the same in both dependent and independent cells. No modifications are possible with register (R) instructions.

6. EXECUTE instructions can be sent from the controller cell to the dependent global cells. In this way every global cell can execute a different instruction. The address is calculated, the contents of the specified memory location are obtained and executed as an instruction. The fetched instruction may be any legal instruction for a dependent cell. The address (A) modification is allowed, in this case, the address of the memory location would be sent by the controller cell. No other modifications may be used.

7. COMPARE instructions are executed much differently in dependent cells than in traditional computers. In the traditional computer, a comparison is made between two values; one is located in a register. The comparison results set some flip-flops. In some computers, a separate instruction tests the flip-flops and jumps or otherwise modifies the program counter. In other machines the same instruction actually modifies the program counter. Sometimes, of course, the instruction does not modify the program counter, depending upon the results of the comparison.

The dependent cells in the global state do not use the program counter, thus another means of using the comparison results is needed. The concept adapted here is to change the level register instead.

The instruction is received from the inter-cell bus. The address is calculated and the specified word from the cell's memory is fetched. This word (operand 1) is compared with the contents of a register (operand 2). The results of the comparison will set a pair of flip-flops to one of 4 states. These will probably be overflow, greater, equal, less than.

Another instruction will test the state of these flip-flops and take some action. Sometimes, the same instruction may compare, set the flip-flops and take some action. The action to be taken may be one of the following. How many are mechanized would require further study.

- a. Continue at this level.
- b. Increment level register by 1.
- c. Increment level register by 2.
- d. Decrement level register by 1.
- e. Decrement level register by 2.

Any level register change will always discontinue the reception of instructions from the inter-cell bus.

The conditions above are several that could be used. A compare instruction could state:

If Flip-flops are 00 or 01 or 10
THEN increment level register by 1,
ELSE continue at this level.

Many other combinations are possible. How many would require software studies to determine their usefulness.

It is seen that, because the compare instruction uses data that may be different in each dependent global cell, some cells may change levels and thus discontinue receiving global instructions. In these cells, data processing will be continued at a later time when the controller cell sends out a GC for the new level.

Some compare instructions may use several words of memory, or perhaps several words from the inter-cell bus as a DS modification. The setting of the compare flip-flops and the subsequent action is the same as an instruction that uses only two operands.

The modification possibilities have not all been explored. Some possibilities are given here.

- A The given address is used instead of the calculated address.
- D The data sent on the cell bus is used instead of a register operand.
- I The displacement in the instruction is used instead of a register operand.
- DS The words of memory starting at the given address are compared with the data words sent on the data bus. If the comparison changes the level register, the reception of data words is discontinued. If no level change is made, the flip-flops are left at their last state.

8. SKIP instructions are really test and skip. A test is made between two operands, the result of this test is always True or False. The true state will always increment the level register by 1. The reception of instructions from the inter-cell bus will be discontinued immediately and the cell is placed in the dependent wait state. The cell will remain in this state until a CG command is sent indicating instructions of the new level are being sent on the cell bus. The false state will not change the level register; reception of instructions will continue. If an address is required, it is calculated and the operand is fetched from memory. One operand is usually from a register. Some modification possibilities are:

- A A given address is sent on the intercell bus.
- D The data word sent on the intercell bus is used instead of the register operand.
- I The displacement in the instruction is used as an operand instead of using a register operand.
- DS The words of memory starting at the given address are tested against the words sent on the inter-cell bus. If any test is true, the level is changed and data reception is discontinued. If every test is false, the GC format command which indicates the end of the data string will be received; the level register is not changed.

9. JUMP instructions are very seldom sent to dependent active cells because they have no meaning. A Jump instruction will always be preceded by a GC format, because a JUMP, by itself, is never sent over the inter-cell bus. The GC format will indicate a special operation is to be performed. One operation is to load the program counter with a value. The program counter is not incremented or used as long as the cell is in the global state.

10. CC instructions are not sent over the inter-cell bus.

11. GC instructions are received by the dependent global cells. Dependent global cells may receive any of the GC commands: format, state and level control of individual cells, and state control via levels.

The GC format instructions have been described in the preceding sections on addresses and data. These GC instructions describe what is to follow on the cell bus. Eight categories are possible (Table 6-7).

The other GC instructions that may be received control the levels and states. These instructions may control the state of cells on the basis of levels or may control the state and/or level of an individual cell.

The GC instructions for state control via levels are given below:

GC level, G

All dependent cells at this level to go the active global state. Instructions for this level normally follow. If the cell is in the dependent global state when this

instruction is received, two cases occur the levels match or don't match. If the levels match the cell continues in the dependent global state and this is essentially treated as a no op. If the cell's level is different, the cell will go to the dependent wait state; essentially a new set of cells are then made dependent global.

GC level, L

All dependent cells at this level go to the dependent under local control state. Dependent global cells whose level matches will change state, the others will remain dependent global. If the cell goes to the dependent local state, the next instruction for this cell will be taken from a fixed location in the cell's memory.

GC level, W

All dependent cells at this level go to the dependent wait state. The procedure is the same as above except no next instruction is fetched for the dependant global cells that change state here.

GC level, IND

All dependent cells at this level go to the independent state. This instruction will set all the dependent cells at this level to the independent state; the cell or cells will begin by initializing the program counter from a predetermined location in the cell's memory.

These four above GC instructions will force all dependent cells at the given level to change state. Of course, independent cells are not changed, neither are cells that are at a different level from the level number in the GC instruction sent over the inter-cell bus.

Another instruction is the GC reply.

GC level, R

All dependent cells at this level will respond to this GC instruction. The controller cell will now allow the dependent cells to transmit on the cell bus. All cells which responded will now return a constant number to the controller cell. Because all cells are setting the cell bus lines to the same value, the hardware design is simple.

This instruction is used by the controller cell to determine if one or more dependent cells are at a certain level. The cells may switch levels dependent upon the value of the data processed by a cell. Thus some cells may or may not be at a specific level. To enable the controller cell to quickly and at low overhead determine if there are any cells at a given level, this reply instruction is included. If no cells are at this level, no cell will send back a constant to the controller cell, and the controller cell will not receive the constant reply number. The controller then may not need to send this level program. If at least one cell replies, the controller cell will note this and send out the program to process this level of cells. There is no way for the controller cell to know how many cells are receiving the instructions at a level without interrogating each one individually. With the reply instruction, the controller cell knows only that there is at least one cell at this level.

Table 6-7. GC Formats

Cell Bus Bits 5-7	Description
1	D16 (16 bit data word)
2	D32 (32 bit data word)
3	A (Given address)
4	A and D32
5	A and D16
6	I (Immediate)
7	DS (Data is being sent)
8	Instructions follow (end of DS)

In addition, another GC reply instruction is:

GC level, R-DG

All dependent global cells at this level will reply as in the above instruction. However dependent local and wait cells will not respond. This instruction is particularly useful to determine if any dependent cells are still in the global state.

The GC instructions for state and/or level control of individual cells will be given below.

These instructions were listed in Table 6-5; the general form is:

GC State, Level
or GC State, No Level Change

The GC command is sent over the inter-cell bus by the controller cell. This command is addressed to an individual cell regardless of state and only the addressed cell will pick up the command and execute it. The command is executed similarly in all cells regardless of their initial state. The addressed cell will change state as indicated by the command and also change its level as indicated by the command. If the cell is already at the indicated state and/or level, no change occurs. Therefore, it is possible for example to keep a cell at the same state and change its level.

The procedures for changing states depend on the initial state and the final commanded state in the cell. Since this section is concerned with the execution of instructions in dependent global cells, only this initial state will be considered.

To enter the dependent local state the cell will fetch its next instruction from a location in the cell's own memory given by a fixed address. Entering the dependent wait state will result in the cell entering an idle mode and suspension of reception of instructions over the inter-cell bus until a GC instruction is received forcing it to again change states.

If an independent state is indicated, the cell will place the contents of a location in the cell's own memory, given by a fixed address, into the program counter. The cell will then begin fetching instructions from its own memory under control of the program counter. This location may simply be the address saved when the cell was last independent and told to go to the dependent global state, an address placed there by the controller cell, etc.; the possibilities here are numerous on how to restart the cell in an independent state.

The controller cell state is entered by having the cell fetch its next instruction from a location in its own memory given by a fixed address. One of the first things the new controller cell will do is to change the old controller cell into the independent state.

12. IO instructions are described in the section on input-output (see Section 7). Dependent global cells will usually not execute IO instructions, since the inter-cell bus is being used for global instructions. Dependent global cells will normally be switched to local control to execute IO instructions.

6.7.2 Dependent Cells - Local Control State

The dependent cell can have its level register at a different value than the instruction and data level that is being sent over the cell bus. These dependent cells that are not active and not receiving global instructions may (1) idle or (2) execute instructions from its own memory. The first case is called the wait state, the second is local control and is discussed in this section.

The execution of instructions from the local memory will continue until one of the following events occurs:

1. A CC instruction is executed that puts the dependent cell in the wait state.
2. A GC instruction is received from the inter-cell bus that specifies this level and causes a change of state.
3. A GC instruction is received on the inter-cell bus specifying this cell's address and causing a change of state.

In the second case, the global instructions will always be used whenever they are identified as being at the same level as the level register. The programmer is responsible to be sure that the local control program is at a completed state before global instructions at this level are sent from the controller cell.

Because the instruction execution is similar to the dependent global cell, only the differences will be noted below. Note that any GC format instruction modifiers must be stored in the cell's memory, preceding the modified instruction. The sequence of instructions is controlled by the program counter.

<u>LR</u>	The instruction is fetched from the cell's memory, the register is loaded from the memory location specified by the effective address. The calculated address is used unless a CC format modifier instruction is present. The D, A, or I modifier may be used.
<u>STR</u>	The instruction is fetched from the cell's memory, the register is stored in the memory location specified. The calculated address is used unless a CC format modifier is present. The D or A modifier may be used.
<u>OPR</u>	The operation is performed between the memory location contents and the register. The CC format modifier can specify an A, D, I, or DS modification.
<u>RR</u>	These are executed exactly the same way in all cells. No modifications are possible.
<u>R</u>	The register instructions are executed the same way in all cells.
EXECUTE	Execute instructions are executed as in an independent cell. The address is calculated and the contents of the specified memory location are executed as an instruction. The fetched instruction may be any legal instruction for a dependent local control cell. A CC format modifier may specify a given address.
COMPARE	instructions are executed in the same way as in a dependent global cell. The comparison is made in the same way, only the instruction and all data are obtained from this cell. The flip-flops are set in the same way. The level register is either changed or will remain the same. If the register is changed, the cell will automatically go to the wait state. If the register is unchanged, the cell will continue to execute instructions in the local control state. The compare may be modified as given in the dependent global cell description.
SKIP	These instructions are executed exactly as in a dependent global cell. The results of the test, if true, will increment the level register and force the cell into the wait state. The false state will not change the level register and thus the program will continue and fetch the next instruction. The skips may be modified, as described in the dependent global cell description.
JUMP	These instructions will usually be executed as in an independent cell. The new value of the program counter is calculated and replaces the present program counter value. Conditional jumps are also possible. One jump will take place depending upon the comparison flip-flop setting.

CC

A very restricted part of these instructions may be executed by the dependent local cells. This includes the Format modification (7.2 in Table 6-4) and part of the state and level control instructions (7.3 in Table 6-4). It should be kept in mind that these CC instructions will be located in this cell's memory. The following state and level control instructions may be executed.

- Ind, Level
- Ind, No level change
- Dep Wait, Level
- Dep Wait, No level change

Any other CC instructions (1 through 6 in Table 6-4) will not be executed in this state.

The use of the format instructions has been described in the previous sections.

The use of any of the state and level control instructions will result in a change of state and possibly a level change. The procedures for entering the Independent or Dependent wait state are the same as these presented for entering from the dependent global state.

GC

Format GC instructions sent over the inter-cell bus are ignored by dependent local cells.

The GC instructions that control state on the basis of level and those that control state or level on the basis of cell address may be executed by dependent local cells.

The GC instructions for state control via levels will be executed only if the level register matches that of the instruction.

- GC level, W
- GC level, DG

The execution of the above two instructions will result in the cell assuming the new state. Going to the global state means that the cell is now ready to accept instructions via the inter-cell bus.

- GC level, IND
- GC level, R

The above two instructions are executed as in the dependent global cells (see Section 6.7.1).

The GC instructions for state and/or level control of individual cells are handled quite similarly to that described for dependent global cells; the differences will be brought out in the discussion below.

To go to the dependent global state, the present instruction will be completed, the program counter saved, and the hardware accumulator stored in its memory location. Then the state is set to dependent global and the cell is ready to receive instructions over the inter-cell bus.

The dependent wait state will simply result in the cell entering an idle mode as before. The comments as to the present instruction, program counter and accumulator in the above paragraph apply here also and to every change of state for the dependent local cell.

The same procedure is followed to enter the independent and controller cell states as given for the dependent global cell.

IO These instructions are executed as in an independent cell. They are described in Section 7.

6.7.3 Dependent Cell - Wait State

This cell is not executing instructions. The program counter is not being incremented.

The dependent cell in the wait state is always examining the cell bus. When a GC instruction that controls states via levels is received which has the same level number as the contents of the cell's level register, the cell will switch to the commanded state. A GC instruction may also address this cell individually to change state and/or level. The procedures followed to change state are identical to that described previously.

6.7.4 Independent Cell

The cell operating in the independent state is described below. The independent cell operation is very similar to the traditional computer operation. These cells fetch all instructions and operands from the cell's own memory. The instruction fetched is located at the address contained in the program counter.

The independent cell cannot set its ID register. The level register, although it is not used by an independent cell, may be set to any value via a CC instruction.

An independent cell will respond to GC commands received on the bus that specify this cell address (last name). Independent cells do not respond to commands based on level. The cell that is in the independent state must stay in this state until the controller cell sends a command on the inter-cell bus with a cell address equal to the contents of the cell's ID register to change states. Thus each independent cell must be addressed individually. The independent cell concept is an important part of the distributed processor system. Other similar computer systems require all cells to be independent or all dependent. It should be noted that the capability for having independent cells switch themselves to dependent states was not included. The reason for this is that it is felt to be of little use since a dependent local cell acts very much like an independent cell and programs that may want to change states and be performed locally may be done in a dependent local cell. This approach simplifies the executive procedures in the controller cell.

LR instructions are all instructions that fetch an operand from a memory location and load the contents into a register. The address of the memory location is calculated by adding the base register, index register (if one is specified) and the displacement from the instruction. A CC format modifier instruction may be used, however, the address is always in the same cell. The D, A and I modifications may be used.

STR instructions are the reverse of the LR, the register contents are stored in the given memory location. An A or D format modifier may be used.

OPR instructions are similar to LR, only the present contents of the register are combined with the memory location contents according to the operation code. The results of the operation are placed in the register. OPR instructions include add, subtract, multiply, divide, AND, OR, etc. An A, D, I or DS modifier may be used.

RR instructions are all instructions that use two registers, and place the results in one register. Add accumulator 1 to accumulator 2 is an example. No memory operations are required (unless the registers are stored in main memory). No modifiers may be used.

R instructions are all single register operations, such as shift, complement accumulator, etc. No modifiers may be used.

EXEC instruction is the traditional computer execute instruction. The specified memory location contents are treated as an instruction; this fetched instruction is executed. The independent cell can only execute instructions located within its own memory. A CC format modifier may be used to specify a given address.

COMP instructions compare two values. One is located in a register, or is understood (such as zero), the other is located in another register or in the cell's memory. The result of a comparison will set two flip-flops to a certain state, which will be one of 4 states. An equal comparison, for example, may set the flip-flop to 00, greater to 01, and less than to 10. The compare may be modified as in the previous descriptions; A, D, I or DS modifications are possible.

SKIP instructions are always a test and conditional skip. The value tested may be in a register or in memory. The result of a test is always true or false. The A, D, I or DS modifiers may be used.

The independent cell will modify the program counter contents based upon the results of a skip test. If the test results are true, $P + 2$ replaces the contents of the program counter P. If the test results are false, $P + 1$ replaces the contents of the program counter P. In other words, the following instruction is skipped if the test results are true:

JUMP instructions are either conditional or unconditional. Additional operations may take place in addition to the jump, such as storing the program counter in an index register.

The jump is implemented in an independent cell by replacing the contents of the program counter with a new value. This new value is the location in the cell's memory where the next instruction to be executed is located. The calculation of this new value is dependent upon the type of instruction; however, in all cases, a new value replaces the old value. Conditional jump instructions make a test, usually on some register. If the test results are true, the program counter contents are replaced. If the test results are false, the program counter is handled in a normal manner, i. e., the program counter is incremented by one. The A, D, I or DS modifiers may be used.

CC A very restricted part of these instructions may be executed in the independent cell. This includes the format modification (7.2 in Table 6-4) and one of the state and level control instructions (7.3 in Table 6-4). The only state and level control instruction that may be executed is to change the level in the independent cell:

IND, Level

Any other CC instructions will not be executed by an independent cell.

The use of the format instructions has been described in the previous section. The instruction to change level simply results in the level register being set to the new value specified.

GC The only GC instructions on the inter-cell bus that will be executed by an independent cell are those that control state and/or level on the basis of cell address.

If the cell address in the GC command matches the cell's ID register, the command will be executed. The procedure to change states is identical to that given previously for the dependent local cell; namely, the present instruction is completed, the accumulator stored in its memory location, and the program counter saved in the location used for interrupts. Then the state is changed as described previously. In this case the procedure in going to the dependent local state is similar to that of going to the independent state for the dependent local cell, namely, the next instruction is fetched from a location given by a fixed address.

IO Input-Output instructions are executed normally, as described in Section 7.2 of this report. In fact, the Input/Output operation is the same for all cells except those that are failed (obviously cannot perform I/O) and those in the power saving state (there are no memory words, the memory is shut off). In all other states the I/O is the same.

6.7.5 Controller Cell

The controller cell is the most difficult to describe. This cell has the characteristics of the independent cell and of a storage bank. The controller cell controls the inter-cell bus. The bus is used for local communication between cells and global control and communications. The local communication operation between cells is described in the section on the communication bus (Section 10).

As far as instruction execution is concerned, the controller cell can operate in basically two modes: the transmit mode and the execute mode. That is, the controller cell can either send out instructions over the inter-cell bus or execute them internally.

The controller cell supplies instructions and data to the dependent cells. The controller cell will first be described in the transmit mode. In this mode the controller cell will be transmitting instructions to the global cells; the instructions are executed only in the dependent global cells and not in the controller cell.

The transmit mode is entered by executing a particular controller cell instruction (see Table 6-4). This instruction will be discussed below when a description of the CC instructions is given. Essentially this instruction causes the controller cell to place the subsequent memory words on the inter-cell bus. The program counter controls the fetch of instructions.

Most fetched instructions that are transmitted are NOT executed by the controller cell. The transmit mode causes non-execution (by the controller cell) of the instruction categories shown in Table 6-8. An appropriate delay between transmissions is made so the dependent global cells will have time to execute the instructions before the next instruction is sent.

JUMP instructions are not sent out unless they are preceded by a GC format modifier instruction. JUMP instructions are normally executed by the CC. The program counter is usually modified to start a new sequence of instructions as in an independent cell. Conditional jumps are executed using data from the controller cell. The address and registers used (if required) are always from the controller cell.

Table 6-8. CC Transmitted Instructions

LR	Load Register
STR	Store Register
OPR	Operate on Register
RR	Register to Register
R	Register
EXEC	Execute
COMP	Compare
SKIP	Test and skip

In the execute mode, the controller cell fetches and executes instructions like an independent cell. All the instructions except the CC and GC instructions are executed identically to the independent cell. Therefore, only the execution of the CC and GC instructions will be discussed below.

CC instructions are listed in Table 6-4. There are seven basic types of CC instructions.

1. Control inter-cell bus I/O logic - Instructions of this type control the logic associated with the inter-cell bus communications section of a cell. Some instructions of this type are: prepare to input from the bus, prepare to output over the bus, etc.
2. Generate GC and Bus I/O commands - CC instructions of this type when executed by the controller cell will result in the formation and transmission of control words on the inter-cell bus. All of the GC commands listed in Table 6-5 are generated by this instruction. This includes the format, state control via levels, and state and/or level control of individual cell commands. In addition, the commands to control local communications between the cells are generated by this instruction. Section 10 contains a detailed discussion on how the GC and Bus I/O commands are formed from this instruction.
3. Transmit Mode, Single - This instruction forces the controller cell to enter the transmit mode until one instruction (including any GC modifiers, given address, etc.) has been transmitted and then return to the previous mode.
4. Transmit Mode, All - Same as above except the transmit mode is retained until one of the two CC instructions given below is executed.
5. Execute Mode, Single - This instruction forces the controller cell to execute the instruction (including any format modifiers) that follows in the controller cell. Then the controller cell is to revert to the previous mode.
6. Execute Mode, All - Same as above except the execute mode is retained until one of the CC instructions, 3 or 4 above, are executed.
7. Do Not Change Mode - These instructions are used to control the execution of instructions in the controller cell and to change its level. The present mode of the controller cell will be retained.
 - a. No Operation
 - b. Format - These are the format modifiers previously described (A, D, I, DS). They are used in the controller cell to modify the next instruction it executes (not transmits). These are not sent over the inter-cell bus. Format modifiers sent over the bus are generated by the CC instruction of type 2 described above.

- c. State and level Control - The only one of these instructions (Table 6-4) that the controller cell may execute is "Controller Cell State - Set Level." This instruction may be used by the controller cell to change its level register.

GC instructions are sent over the inter-cell bus by the controller cell. Therefore, the controller cell will not normally be receiving and executing any GC commands from the bus. The one exception occurs when the controller cell is to change state. This is accomplished by the controller cell executing a CC instruction of type 2 above. The particular GC command generated by the execution of this instruction and sent over the inter-cell bus will be a command for a particular cell to change its state to the controller cell state. The original controller cell will now be waiting for a command over the inter-cell bus to tell it to change states. A timer is set so that this command is received in a certain maximum allowable time. The command will of course be a GC command sent by the new controller cell. If this command is not received in this maximum allowable time, the original controller cell will interrupt and enter a software routine to diagnose the no response condition.

One point should be made with regards the execution of global programs. It is possible to be sending out a global program at a certain level and only have part of the cells in a group at that level be executing this global program. Recall that a cell will be executing the global program if it is in the dependent global state. This situation could occur due to several cases:

1. The cell is independent - its level has no meaning and may therefore be the same as the level of instructions on the inter-cell bus.
2. The cell is dependent local or dependent wait - a cell may be dependent local and initially at a different level than that of the instructions on the bus, then the cell may change level and/or state under its own control and be at the same level as the instructions on the bus. However, the cell will not switch to the dependent global state under this condition.
3. Only one cell is made Dependent Global - It is possible for the controller cell to send out a GC command addressed to an individual cell and place it in the dependent global state. This is the DG or DG, Level command in Table 6-5. The level register may or may not be changed by this command. In any case, if the controller cell sends out instructions over the inter-cell bus following this command, the cell will execute them and assume they are at the same level as its level register. Other cells may be at the same level as this cell, in fact they may even be dependent local or dependent wait; however, they will not enter the dependent global state since a GC to change state of an individual cell was used.

6.8 NEIGHBOR TO NEIGHBOR COMMUNICATIONS

This section will discuss the reasons for including neighbor communications in the distributed processor system. Neighbor-to-neighbor communication is a means for one cell to communicate with a restricted number of neighboring cells. These communications paths are separate from the inter-cell bus. Because the bus is

always needed for global instructions (applied parallelism), the question arises, should special communication paths be provided to neighboring cells, or should all communication go via the intercell bus?

The advantages of neighbor-to-neighbor communication come from the problems of using the bus and the fact that certain computations may be placed with a geometric relationship (matrix and vector manipulations, etc.) that may efficiently utilize neighbor communications. The inter-cell bus requires control as to which cell gets to use the bus, how long the cell may use the bus, and how control is to be passed from one cell to another. A controller cell must have means of controlling transmission priorities. Even if some of this control can be done by hardware, the fact remains that there will always be an overhead in software and storage. To address any word in a group requires 14 bits; thus to send a 16-bit data word requires 14 bits of storage to hold the destination address. Thus for a word of data, a word of address is required. Adding words of program to control the transmission will require additional words of storage.

The time delay in transmitting via the inter-cell bus will depend upon the bus design and how long it takes to establish control. If the bus is 8 data bits wide, for example, about 6 clock times are required to transmit a single 16 bit word. When many words are transmitted in a single message, the clock times per word will approach 2 (one clock time per 8 bits). A detailed discussion of how communication is carried out on the bus may be found in Section 10.

The time delay in transmitting is seen to be a problem, especially when few words are transmitted. A more serious problem is the delay in obtaining control of the intercell bus. The time delay between the time a cell needs the intercell bus and the time a cell actually gets control of the bus is called the request delay. The longest delay will occur when all the other cells have transmissions to make and the requesting cell must wait its turn. The shortest time can be almost zero. Somewhere between these two times will be the average request time delay. This delay is variable, unknown, and could be very large. For any program where the timing is critical, the unknown times could make the programming difficult. In many cases where data has to be passed to neighboring cells the timing is critical, such as in the navigation and guidance routines. It should be noted that the more applied parallelism that is being utilized, the longer the request delay. Since this may mean that more cells are involved in the neighbor communications; if they all had to be serviced over the inter-cell bus, the average request delay can be very large.

It is of course possible to do without neighbor communications providing one does not run into a time problem and can complete the computations in the allotted time. However, as noted above, it would prove very inefficient and difficult to use the intercell bus where small amounts of data have to be passed between neighbors as in the use of applied parallelism.

To improve the system, neighbor-to-neighbor communication is needed. The system implemented must provide the following advantages:

1. Low overhead (no more than one instruction to execute a data transfer)
2. High reliability
3. Known times to implement a transfer
4. Adaptable to reconfiguration

The disadvantages to including neighbor communications are two: (1) additional connections are required to each cell, and (2) reconfiguration is more difficult. The number of connections required per cell increase by four; while this is not a large increase, it does of course provide more failure points. The most serious disadvantage appears to be that of reconfiguration since now the organization is spatially oriented. One approach to this problem is to require the programs to be set up using small independent sets of cells so that one may pack the sets of programs around a number of failed cells in a group. This problem requires further investigation to determine how effectively it can be solved.

While a firm answer as to whether or not neighbor communications are required cannot be given at this time, the advantages in including it appear to outweigh the disadvantages that may be encountered. In addition, inclusion of the neighbor communications provides an organization with increased capabilities (notably speed) so that it may be applicable to a wider variety of applications (for example a high speed video data reduction problem). Therefore, for purposes of this design neighbor communication will be included.

6.9 ADDITIONAL TOPICS

The concepts described in this section result in a powerful system with many options on how operations can be performed. Further study may show how features can be changed or added to improve the system. A number of alternate concepts are given in this section to show some other ideas that were considered.

The given address type of format modifier, as described above, was used without modification by the receiving cell. An alternate idea, which may aid in the software, is to have the cell always add a base register to the given address before it is used. Because each cell can have its base register set to a different value, the placement of programs and data in the cells may be made easier by using a base register with all the given addresses.

The use of a special instruction (GC) modifier was selected here because it used the least amount of time on the inter-cell bus. Another way to modify the instruction execution is to reserve a bit or so in each instruction (or memory word) to indicate its length and any special address modifications, and how much data was attached. The advantage is that each instruction carries its own length code, etc. The disadvantage is that each instruction must be made longer. Using the GC modifier makes only the modified instructions longer, but they are very much longer than they would be in the other method. The present decision was to make the unmodified instructions as short as possible, even though this made the modified ones long. The net result should be a storage saving, especially in the independent cells.

The two modes of a controller cell, transmit and execute, are only one way of selecting which controller cell words are instructions to be executed in the controller cell or in the dependent global cells.

Another method is to place extra bits on each memory word, telling what the word is. This method requires many extra memory bits in all cells.

Another method is to use two program counters. One program counter controls the fetch of instructions to be executed by the controller cell, the other program counter controls the fetch of instructions to be transmitted to the dependent global cells. This two program counter idea allows the modes to be discarded. The resulting system is now much more elegant and powerful. However, the processor now requires much more hardware.

In addition, another method would be to store in the controller cell, as input/output data for the inter-cell bus, all the instructions and data that are to be placed on the bus. This approach is very inefficient since the controller cell must identify control words by setting the control line in the bus; it has no means of distinguishing between instructions and data since they are all stored as I/O data. Of course one approach as noted above would be to have extra bits stored with each word identifying it as instructions or data with the resultant penalty of increasing the number of bits used for storage. Another method here would be to store the count of the number of words between the control words and use these counts to identify the control words. This approach requires storing a number of counts, additional control hardware and makes program modification difficult.

The above are only some alternatives in the design of the group architecture. Many other possibilities undoubtedly exist. The chosen architecture is believed the most efficient and flexible compromise among many alternatives and possibilities. However, further investigation into some alternatives could result in modification and improvements to the present design.

7. INPUT/OUTPUT

7.1 INPUT/OUTPUT OPERATION

This section of the report discusses the operation of the input/output system in the DAMP Computer. Input/output is handled by the hierarchical structure shown in Figure 7-1. This figure shows that the interface to the computer consists of serial and parallel digital lines. The conditioners C_1 through C_N each have a number of sensors connected to them. The sensors provide a variety of signals to the conditioners and the conditioners, in turn, accept these signals and provide a standard digital interface to the computer. Some devices contain their own conditioner circuitry and are connected directly into the computer; these devices will generally be connected in a full word parallel format. The bulk storage memory unit will be one of these devices; other parallel devices may include items such as buffers for video sensor data, etc.

The I/O structure described above was chosen over a completely centralized I/O structure which would absorb the conditioners into the computer and have the sensors interface directly with the computer for the reasons given below:

1. A completely centralized I/O structure is generally used to gain a more efficient hardware utilization by the consolidation of common signal conditioning functions. In this computer, reconfiguration is possible around a number of failures (down to the cell level). Since some of the I/O signals are connected directly into the cells as will be explained later, reconfiguration around a number of failures now makes a completely centralized I/O structure inefficient. This is due to the fact that all the cells would be required to have the capability for interfacing directly with any of the sensors. This approach would result in a large amount of redundant hardware and not provide an overall hardware savings.
2. The conditioner structure is 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 redesign the cells and replace the entire computer with new chips.
3. The conditioner I/O structure also provides ease of adapting the computer system to various vehicles between missions and within missions such as a command module and a lander module of a Mars Lander Mission. These vehicles will have significantly different sensors. 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 each vehicle.

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

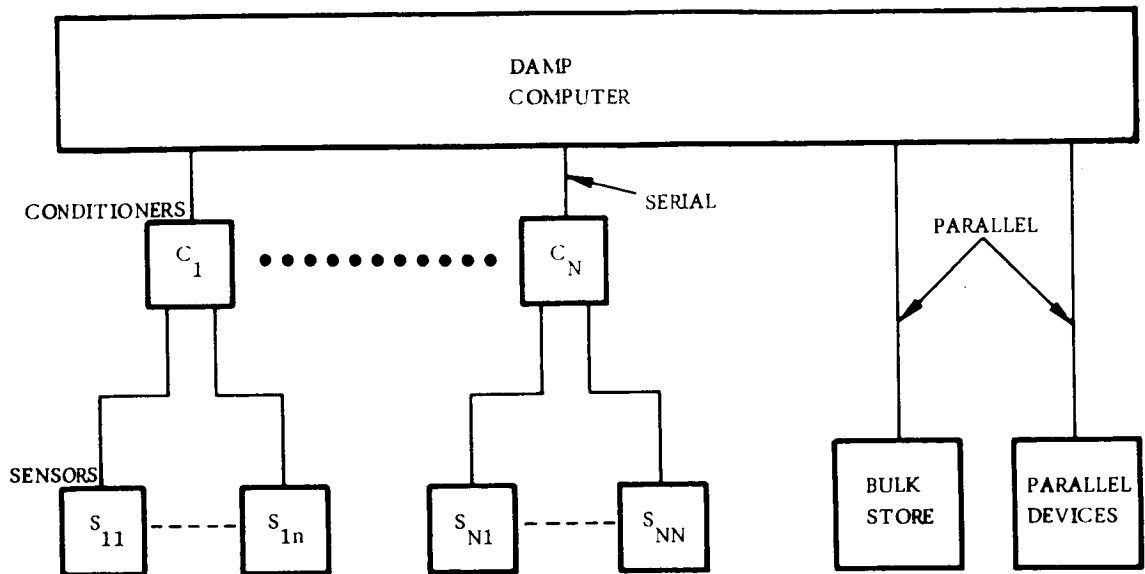


Figure 7-1. I/O Structure

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 operation of the I/O system.

The method in which the I/O ties into the block called the computer in Figure 7-1 will now be developed. A discussion of the possible methods of handling the I/O internally will be given first and then further details will be given on the selected method.

7.1.1 With the Intergroup Bus

Figure 7-2 shows the structure of the computer using an intergroup input/output scheme. The intergroup bus actually consists of two redundant half word parallel busses. Groups in the computer organization use these busses for communications amongst themselves, therefore the I/O devices attached to the busses will appear as groups functionally as far as the computer organization is concerned. The conditioners will transmit serial data to the computer; therefore, a conversion from serial to parallel is required before they get on the intergroup bus. This is accomplished in the blocks labeled I/O Cell in Figure 7-2. Two of these I/O Cells are shown in this figure, one to each of the intergroup busses, this connection provides the reconfiguration flexibility required. Conditioners handling data from critical sensors are

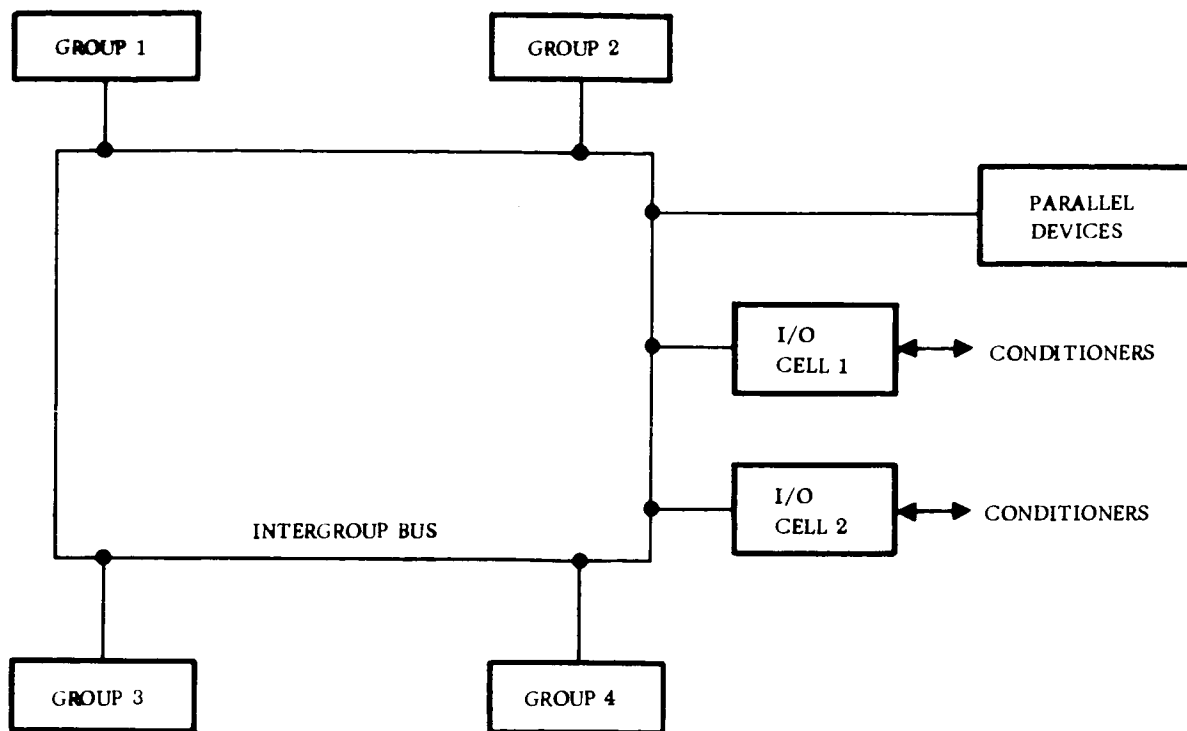


Figure 7-2 . Intergroup Bus I/O Scheme

duplicated and connected redundantly into each of the I/O Cells. Therefore, if a failure occurs in the bus, I/O cell, or conditioner, the other conditioner - I/O Cell - bus connection can be used as a backup.

As noted above, sensors supplying critical data are connected to both I/O Cells; the non-critical sensors such as experiment data will be connected to only one of the I/O Cells. Two alternatives are present here: (1) to have all the non-critical sensors connected to one I/O Cell, or (2) to divide the non-critical sensors between the two I/O cells; these two schemes are shown in Figure 7-3. It should be kept in mind that Figure 7-3 shows only the sensor connected in a serial manner through conditioners to the I/O cells, there are also the parallel sensors which connect directly onto the bus. Each of the two schemes results in a different operation of the I/O system in the computer.

The first scheme where all the non-critical sensors are connected to one I/O cell results in using Bus 1 primarily and Bus 2 serves only as a backup in case of failure of Bus 1, I/O Cell 1, etc. This is due to the fact that the only I/O connections to Bus 2 are redundant connections from the critical sensors. With the second scheme both busses must be used since the non-critical sensors are divided between the two busses. The first scheme requires a higher communication rate capability on the bus when compared to the second scheme. However, note that in case of failure and reconfiguration the first scheme will be able to offer a full reconfiguration since either bus can handle the total I/O requirement. The second scheme besides reducing the communication rates required on the bus introduces a new executive method of handling the I/O. Since the I/O control and data are now divided onto two separate busses, the executive in charge of I/O has the additional task of scheduling and

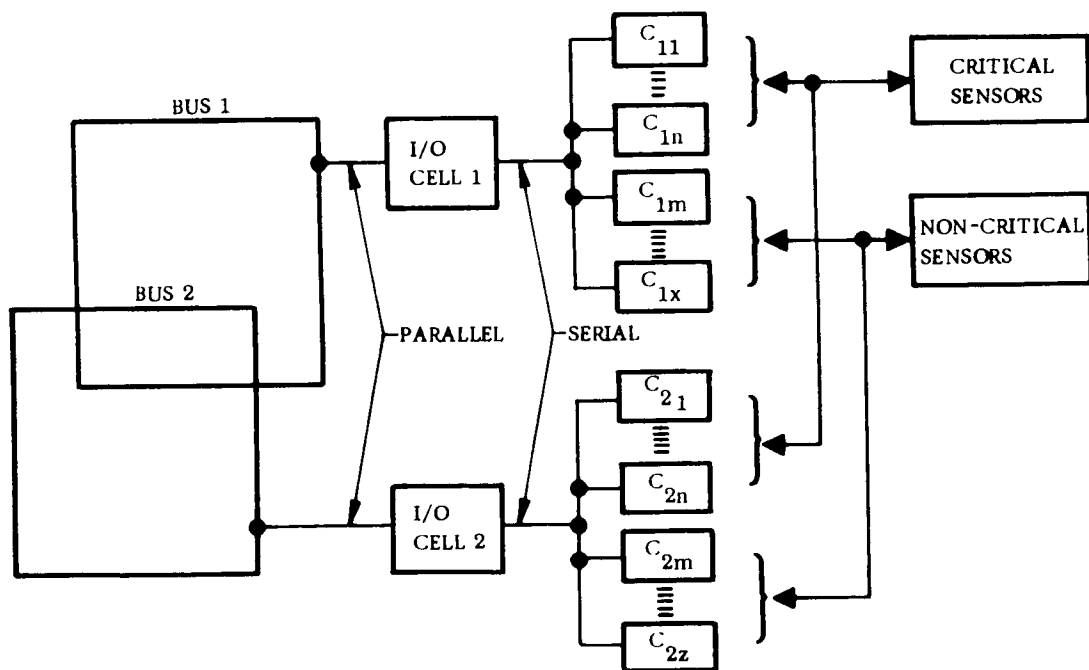
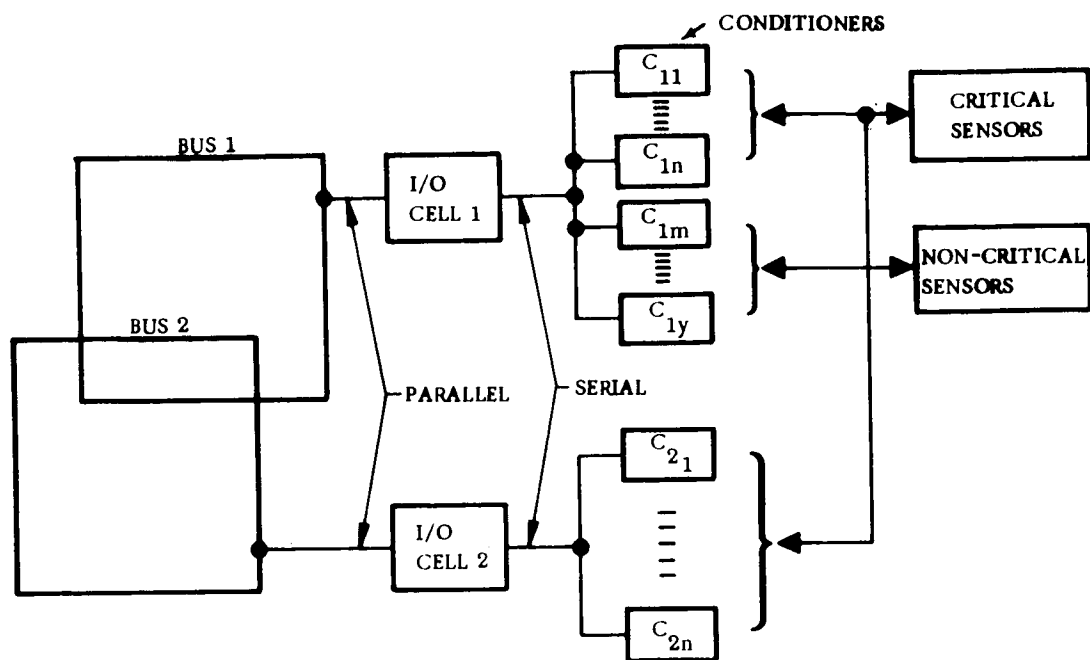


Figure 7-3. Two Methods of Inter-Group Bus I/O

interleaving I/O into the groups and avoiding conflicts of simultaneous I/O on the two busses into one group. Of course the scheme with two busses also offers more flexibility in handling the I/O from an executive viewpoint since it is possible to handle periodic high priority on one bus and background type of I/O on the other bus, etc.

It should be noted here that the I/O cells are identical to the cells used in the groups in the distributed processor. The connections to the bus are now directly made through the cell and not a group switch as is the case in the groups. The serial connection to the conditioners is mechanized with the neighbor communication line. It should be noted that it is possible to use each of the four neighbor lines of a cell to sets of conditioners; this could be done to increase the communication rate capability to the conditioners (serial lines) if this became a bottleneck problem.

The cell contains processing and storage ability and can therefore function very well as an I/O processor since it has general purpose computer capability. Memory in the cell can be used to store programs for inputting/outputting data, some of these programs may be permanent and some may be loaded into the I/O cell by the executive group. The memory may also be used to store data, thereby acting as a buffer device.

7.1.2 With the Intercell Bus

In Figure 7-4 the I/O concept introduced above is extended to the intercell bus. The I/O scheme is quite similar to that given above and only the differences will be pointed out here. Only one bus is required in the group for the intercell bus, whereas two are required for the intergroup bus; therefore, only one I/O cell is shown in the figure. The sensors connected to the I/O cell may be critical sensors, non-critical sensors, or a combination of the two. Redundant inputs of critical sensors would be connected to another group, thereby providing the reconfiguration capability required if a group failed. The I/O cell is physically the same chip as the other cells in the group. It is connected on the cell bus just as the other cells are; however, its neighbor communication lines are not used with neighboring cells but for communication to the I/O conditioners. The same comments apply with regards using all four neighbor lines for I/O conditioner connections.

This approach has a communication advantage over the previous one in that the intergroup bus now is not tied up with all the I/O data since the data is now fed directly into the group where it is being used. It also does not require the overall or intergroup bus executive to be concerned with handling and scheduling the I/O; it is simply handled by the executive directly in a group. The disadvantage with the approach just presented is that the I/O cells in Figure 7-4 are somewhat specialized in comparison to the other cells in the group since the neighbor lines of the I/O cell are not connected to neighboring cells. This then makes reconfiguration difficult if the I/O Cell in the group should fail. With this approach it is also difficult to handle data which may be required by any or all the groups, e.g., data from the bulk storage device. If it is not possible to connect such devices to more than one group then a group may be required to transmit data from such devices to other groups via the inter-group bus. This obviously can place quite a burden on an intercell bus.

The logical extension of the two approaches thus far discussed would be a combination of the two: connections to the cell bus as presented above and connections to the group bus similar to that discussed previously. There are many possibilities open here. One scheme may be to connect noncritical sensors and one of the inputs from

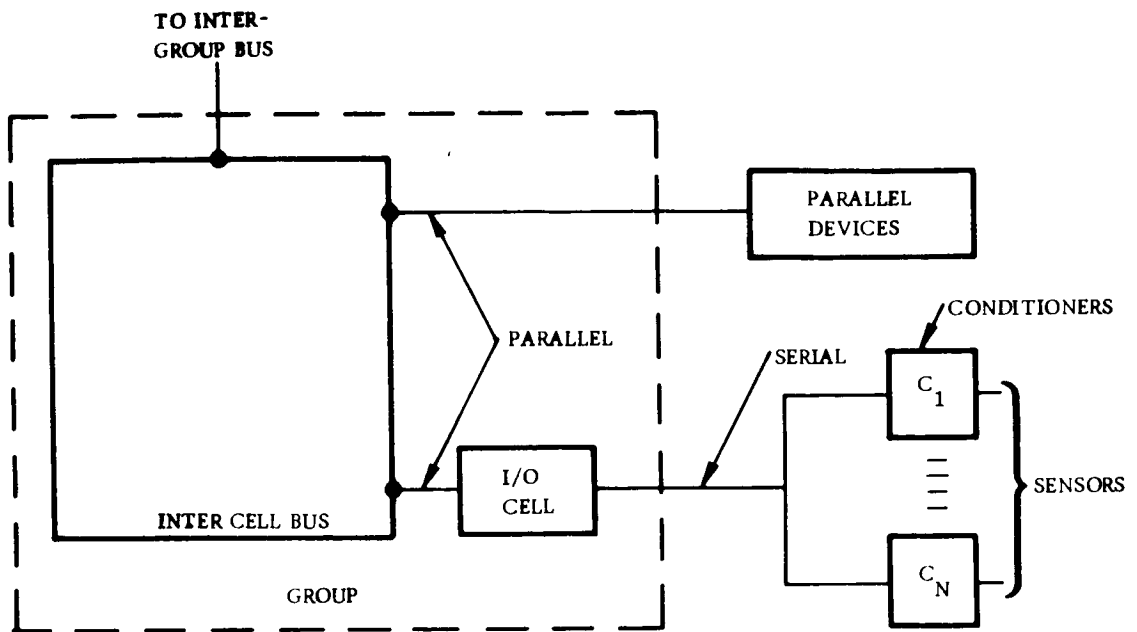


Figure 7-4. Intercell Bus I/O

critical sensors directly to groups and the other input from critical sensors and devices such as the bulk store directly to the intergroup bus. An alternative may be to connect the critical sensors directly to different groups; there are numerous possibilities here, each with certain advantages and disadvantages.

7.1.3 With the Cells Directly and the Communication Busses

Another scheme considered and the one selected is to provide connections directly to the cells in the groups for I/O and also connections to all busses in the system. This I/O approach is shown in Figure 7-5.

It should be recalled that with the previous I/O approaches of communicating to the busses; a cell had to be provided for connecting the serial conditioners into the system (I/O Cell). This cell was identical to the other cells in the system. However, it was specialized in that it was not connected in the regular pattern (array) of cells in the groups. Therefore, since the I/O cell was specialized it could not be replaced by the other cells in case of failure. The selected approach requires a separate lead to be brought out from each of the cells in the groups to an I/O connection panel; this is similar to another neighbor communication lead. Every cell therefore has the capability for handling I/O. Connections from the busses are brought out to the panel also. This scheme requires no additional or specialized I/O hardware in the system. All that is required is to provide an additional connection from each of the cells in the system (the I/O hardware can principally be thought of as connections here). Since the I/O is handled by cells directly in the group, reconfiguration around failures in terms of I/O is relatively straight forward. This is due to the fact that any of the

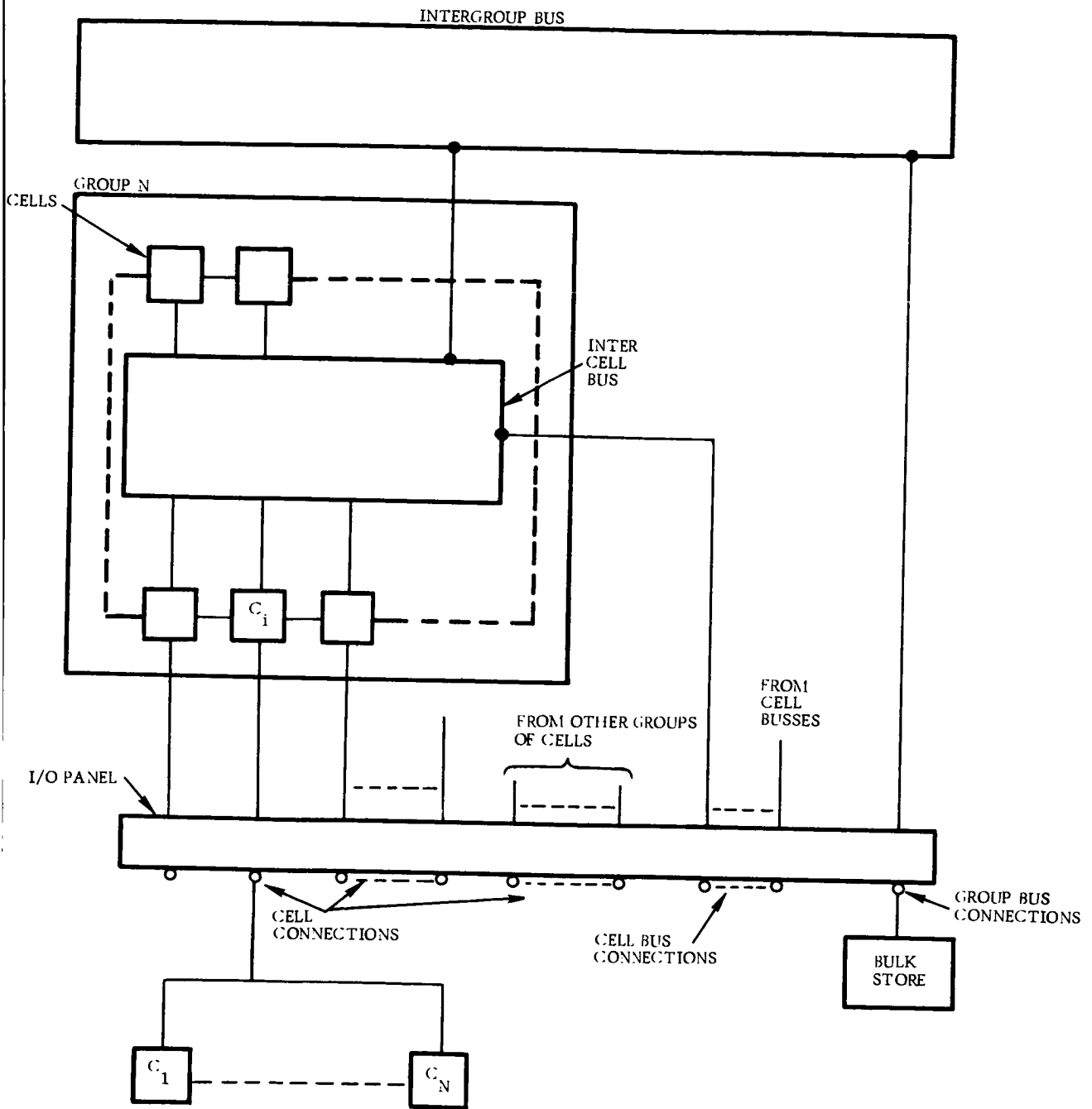


Figure 7-5. Selected I/O Approach

cells can handle I/O functions. It may be necessary to unplug - plug connectors on the I/O panel. However, a group will not necessarily then be lost due to the failure of an I/O cell.

Another possibility along these lines that was given some thought, was that of using one of the four neighbor leads for I/O as shown in Figure 7-6. In this approach connections are made to half of the neighbor lines and brought out to an I/O panel as in Figure 7-5. This eliminates the extra connection to each cell as described in Figure 7-5.

Some of the neighbor lines are now shared between neighboring cells and I/O conditioners. The problem that arises in doing this is that of avoiding conflicts of usage on the common line. If only a single line is used (bidirectional channel) for neighbor communications, then an additional connection must be added to the cells to serve as a request/acknowledge line for use of the common line between I/O and neighbors. This is required since a request/acknowledge approach must be used -- otherwise, communications over the common line would be random and meaningless. It is impossible to use only one line with a request/acknowledge approach, since the request signal from one source would naturally interfere with any established communications by the other source sharing the line. It should be noted that since it is now required to add the request line, one might just as well use this extra connection for a separate I/O line as in Figure 7-5.

If two unidirectional lines are used to mechanize the neighbor communications, the situation is somewhat different. A request/acknowledge type of approach is still required since there still would be possibilities of simultaneous usage of the communication lines (although not as probable as with one line). With a request/acknowledge approach and unidirectional lines it is possible not to require any additional lines since the scheme can be mechanized on the two unidirectional neighbor lines. This scheme would require using one of the lines for notification purposes to inhibit a request from the non using source. If two requests occurred simultaneously the proper acknowledge signal can be inhibited.

It is thus seen that if bidirectional lines are used for the neighbor communication lines, no advantage results in tying the I/O directly into the neighbor lines. However, if unidirectional lines are used, this approach could save two connections per cell.

Further explanation of the selected approach shown in Figure 7-5 will be given below. Each cell contains communication lines as shown in Figure 7-7. The I/O line shown in this figure is similar to an additional neighbor line added on to a cell. Each of the I/O lines are brought out to the I/O panel as shown in Figure 7-5. This results in an increase in the number of connections in the system, which, of course, degrades the reliability of the system. However the actual increase in terms of connections actually used is not as great as that provided (approximately 80), since only a small portion of the I/O connections to the cells will be used (approximately 10), as will be pointed out below. Therefore, this scheme actually results in only a small number of extra connections in the system.

As shown in Figure 7-5, a number of conditioners, C_1 --- C_N , are connected to a cell's (C_i) I/O line. This connection hierarchy is the same as described in the preceeding approaches. Connections are brought out from both the intercell busses

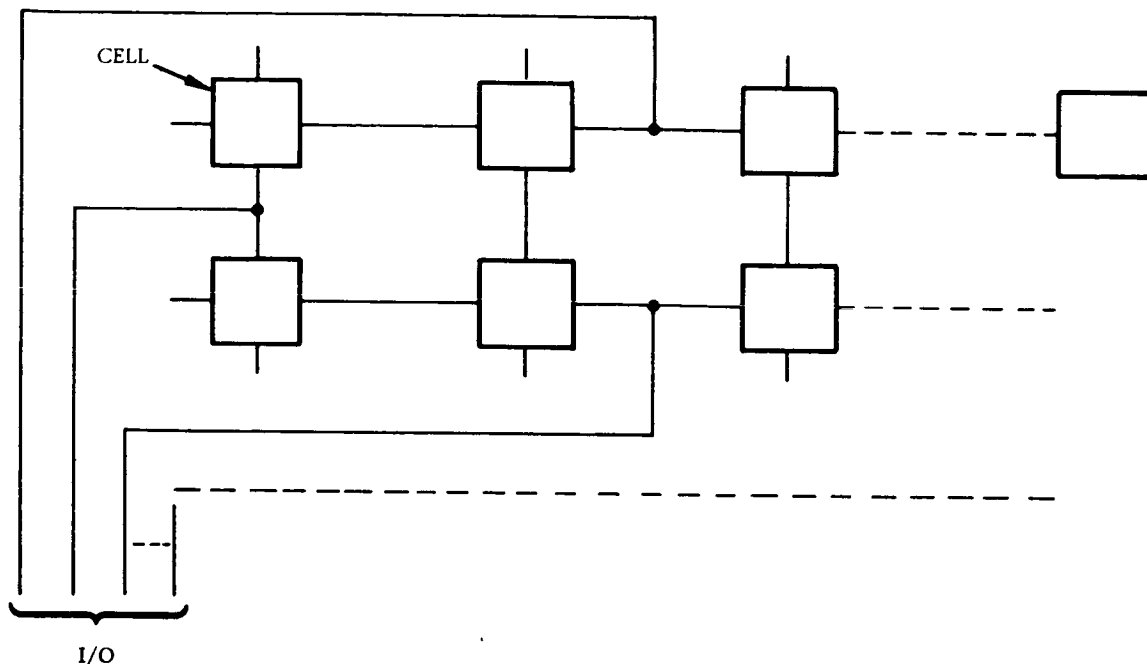


Figure 7-6. I/O Using Neighbor Lines

and the intergroup bus to the I/O panel. These are parallel connections and the type of devices that will be connected here are, e.g., the bulk storage unit, special buffers for scientific experiment data, high rate experiment sensors with large quantities of data, etc.

As mentioned previously, only a small number of the cell I/O connections will be utilized. There are a number of reasons for taking this approach. If one tended to use many of the cell I/O lines then many cells would be associated with particular conditioners or sensors. This places a severe restriction on reconfiguration. Consider reconfiguration due to phase changes for example from midcourse cruise to midcourse velocity correction; if the sensors are closely associated with particular cells one is now constrained as to where new programs may be placed in the system. It would be undesirable to have the I/O information coming into many different cells which may not even need this information and then have to be placed on the inter cell bus to cells that require it. It may be necessary to unplug-plug sensors to affect a reconfiguration in this manner.

Another disadvantage with this approach is that the reconfiguration of the system around cell failures is difficult. While the probability of a single cell failing may be quite low, there are many cells in the system (approximately 80). Therefore, by using I/O connections to many cells the probability of having a failure now associated with I/O signals is increased. If a cell fails that is being used with an I/O connection and the conditioner is connected only to this one cell, then this conditioner has to be unplugged-plugged to affect reconfiguration. The reconfiguration cannot be handled by software alone.

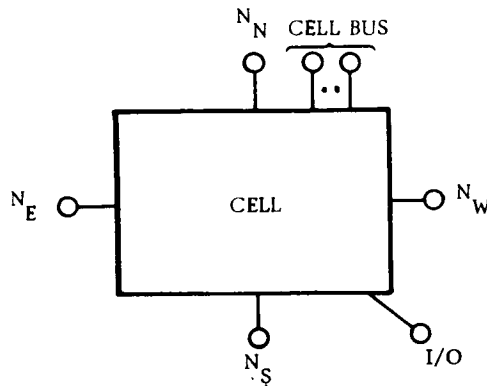


Figure 7-7. Communications Lines to a Cell

The advantage in using more connections is that more of the I/O can now be brought into cells that will use the data directly, thereby reducing the amount of I/O that has to be handled over the busses or the neighbor lines. In order to provide the reconfiguration flexibility, the approach of limiting the number of I/O connections to cells was taken.

One or more cells per group will be connected to I/O conditioners as in Figure 7-5 (typically 2 or 3 cells). The I/O control in the group may be handled in a number of ways. One approach is to have the I/O handled by both the individual cells and the cells connected to the conditioners. The individual cells will have small I/O routines for calling and accepting I/O data from the cells acting as I/O cells. The I/O cells will contain routines for servicing requests from other cells and for generating requests to other cells; they may also contain autonomous routines such as for I/O of periodic sensor data. The I/O cells will also have some memory available for buffering of I/O data.

Other possibilities include having the controller cell in the group supervise the I/O programs and/or also contain a good portion of I/O routines within itself. This may be useful for I/O data intended for more than one cell. It is also possible for the I/O cell to use its neighbor lines to pass I/O data to its four neighbors, thereby eliminating the use of the bus for certain I/O data; this could prove useful for precisely periodic sensor data. The intent here is to present the general concept of the I/O system; further definition of the requirements and in particular the sensors is needed before an exact approach is designed.

As mentioned previously, connections are provided to the busses. Devices connected to the intergroup bus such as the bulk storage unit will effectively function as groups. The system executive will control the communication (I/O) between such devices and the groups in the system. Devices connected to the intercell bus will effectively function as cells; the controller cell in the group may control I/O to such devices and/or the individual cells may also provide requests for I/O action to such devices.

One other point should be mentioned here, critical sensors will be connected to I/O cells in two different groups. This will provide for the required reconfiguration capability if the intercell bus or I/O cell that a critical conditioner is connected to should fail.

7.2 I/O MECHANIZATION

A preliminary design of the I/O system was completed to get an estimate of the I/O mechanization requirements. This section presents a summary of this investigation.

The I/O cells are to carry out their communications (both in and out) with the conditioners over a single line connected between the I/O cell and all conditioners that are to communicate with the cell. The I/O cell has control over this line and operates under control of an internally stored program.

I/O is performed by executing the I/O instruction in the processor section. This instruction will be presented in Section 9 where the entire instruction set will be discussed. However, it will be introduced here so that the mechanization of the I/O operations can be clarified. It will be stated here that the basic instruction length is 16 bits with 6 bits used for the operation code. The I/O instruction uses a long instruction format; this means that two 16 bit words are used to form the instruction (a 32 bit instruction). The contents of the I/O instruction are shown below:

bits:

6	1	9	7	9
Op Code	I/O	Address	Cond/Device	Word Count

The first word of this instruction identifies the operation as either input or output and also contains the address of the location in the cell's memory where the first word will either be output from or inputted to. The second word identifies the conditioner and device that the I/O is to be carried out with. Seven bits are provided here for identification of the conditioner/devices. This provides the capability of handling up to 128 devices per I/O cell. If this proves too few devices, one may possibly use one of the word count bits for additional conditioner/device identification. A preliminary feeling for the breakdown is to provide for 8 conditioners and 16 devices per I/O cell. Nine bits are provided for the word count. This results in the capability of inputting/outputting up to 512 words (full memory) with one instruction.

The sequence of events occurring in executing the I/O instruction will now be explained. A combination of hardware and software will be used and the relative usage of each to accomplish the instruction may be varied; the description given is for preliminary design purposes only.

The following sequence of operations are carried out to execute the I/O instruction: (by hardware)

- Place Address in a certain index/bank register
- Place proper bits in buffer register
- Shift out buffer register over I/O line
- Place word count in a certain location in memory
- Transfer to the input or output (depending on rd/write bit) software routine

The address of the input/output routine could be hardwired or previously setup in a certain index/bank register. As noted above hardware and/or software may be used to execute the I/O operation described above. A preliminary feeling is that the above may be carried out most efficiently by hardware.

The buffer register contains the following information in it when it is to be shifted out:

bits:	1	1	7	8	1
	Sync	Cont/data	Cond/Device	Spare	Rd/Write

The buffer register is seen to be 18 bits in length. This is due to the fact that two control bits must be added to utilize only one line in the I/O mechanization; these are the sync and control/data bits. The sync bit is always a one and is used to synchronize the conditioners. The control/data bit identifies the following 16 bits as either a control word or a data word. The control word shown for the conditioners uses 7 bits for the conditioner/device identification--8 bits are not used (actually the word count may be placed in here if this is deemed useful to the conditioners) and one bit for identification of the operation as input or output.

The conditioners are clocked with the I/O cell and utilize the sync bit to set a counter. Upon counting to 18 the conditioners reset and are ready for the next transmitted word. Upon detection of a word identified as a control word, each conditioner will examine the conditioner/device bits to determine if the control word is for this particular conditioner. If it is then this conditioner will lock on to use the I/O line for receiving or transmitting the data that follows.

A preliminary description of the output routine will be given below. This routine will be described as a software approach. It should be kept in mind that hardware may be substituted to achieve a faster execution if this is found to be necessary.

Output routine:

- Load Accum. with first word (address specified by previously setup index/bank reg.)
 - Transfer Accum. to Buffer Register (with this the buffer is automatically started shifting out)
 - Load Accum. with word count
 - Subtract 1
 - Test and transfer on 0
 - Modify index/bank reg. for 1st instruction
 - Jump to some location back in program
 - Jump to master I/O routine
- = 0

To get back to the output routine (when the buffer register is shifted out), an interrupt is sent from a counter associated with the buffer register; this transfers the program to the output routine (location of this routine could be a hardwired location or previously loaded in an index/bank register).

Input Routine:

- Transfer Buffer to Accumulator
 - Store Accum. in first word (location setup by index/bank reg)
 - Load Accum. with word count
 - Subtract 1
 - Test and transfer on 0
 - Modify index/bank register used in 2nd instr.
 - Jump to some location back in program
 - Jump to master I/O routine
- = 0

To get back to the input routine (when the buffer register is full again), an interrupt is sent from the same counter associated with the buffer register (the I/O instruction sets a mode flip flop to determine if the input or output routine is to be entered).

The above description results in an I/O system completely under control of the I/O cell. The conditioners cannot interrupt or request I/O operation independently from the I/O cell. To facilitate requests from devices for I/O operations (for example a request from the astronauts panel or a buffer holding experimental data), it is possible to insert in each (or possibly only several) conditioners a request register that may be sampled periodically by an input operation by the I/O cell. Then the I/O cell may decide how to handle the requests, if any.

Another possibility is to add a separate request line as shown in Figure 7-8. This line passes serially through the conditioners. Periodically, the I/O cell sends out a request pulse. If no requests are present in the conditioner, the connection will be successively completed to the last conditioner where the request line will be grounded. This, then, signifies no request is present. The first conditioner with a request that receives the pulse will not complete the circuit. The I/O cell recognizing this knows a request is present but not from which conditioner. The conditioner with the request that received the pulse will now send a control word to the I/O cell indicating what I/O action is desired with the proper identification.

The advantage with this scheme is that the conditioners may be sampled very quickly and easily (minimum software in the I/O cell) by the I/O cell to handle requests by the devices for I/O operation. Its disadvantage is that it costs an extra connection. Note however that if a failure occurred in this connection one could always revert back to the original approach without this scheme.

For purposes of this design the request line scheme was not incorporated. Further definition of the requirements is necessary if it is to be justified. It was seen that software routines were proposed for actually inputting and outputting data. Further study is required to determine if some or all of the functions of these routines should be handled by hardware. This depends on what operations can actually be interleaved with the I/O operation.

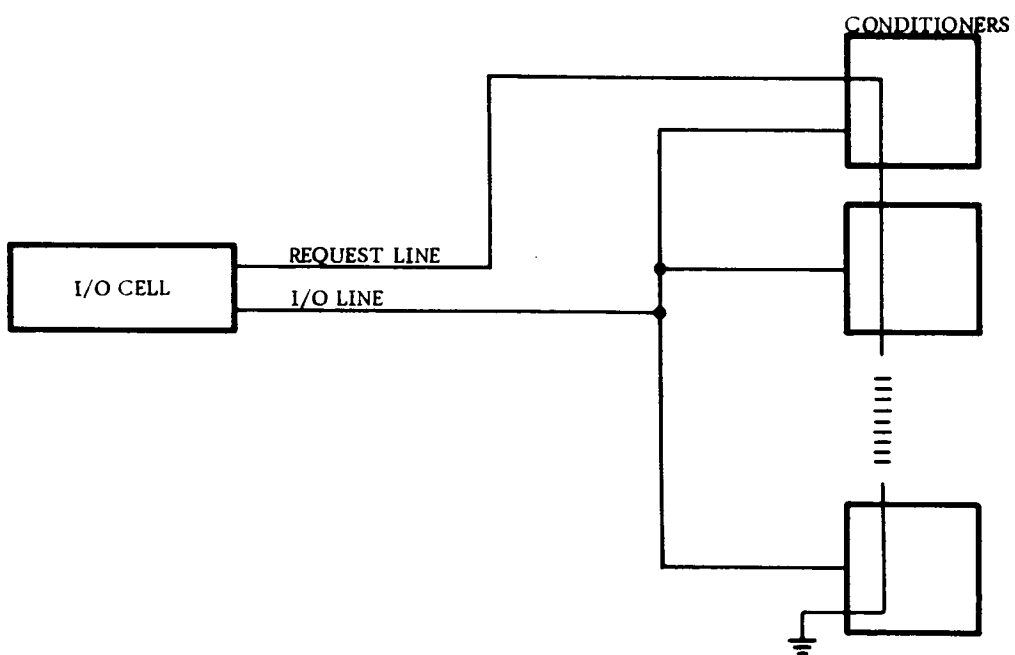


Figure 7-8. Request I/O System

8. FAILURE DETECTION AND RECONFIGURATION

The ability to detect failures and reconfigure the distributed processor around failures is required in order to meet the probability of mission success and availability goals. This section of the report presents the design of this capability.

Some of the basic guidelines forming a framework for this section are given below. The time from the occurrence of an error to the time the system is reconfigured and functioning properly should not exceed 0.1 seconds for critical mission phases. Similarly, the time from the occurrence of an error to the time it is detected should not exceed 0.1 seconds for critical maneuver phases. This time applies to the critical mission phases and can be much longer for the non-critical mission phases such as transplanetary coast. Crew participation can be considered for the following functions (a) reconfiguration during non-critical mission phases, (b) turn-on and requests for checkout of idle stand by equipment, and (c) replacement of failed equipment with a spare, verification of the repair, and insertion of the equipment back into the system.

8.1 FAILURE DETECTION METHODS

There are many approaches to detecting the failures that may occur in the system. This section will discuss the approaches considered and select the most promising one. Two important guidelines should be stated before proceeding with the discussion: (a) it is desired to detect intermittent type of errors and (b) it is desired to detect and isolate failures to the cell level.

There are two basic approaches that may be considered: hardware and software. The selected approach consists of a combination of hardware and software methods. Hardware methods will be used where a large percentage of errors are detected for a reasonable increase in hardware complexity. Software methods will be used primarily where it is difficult or inefficient to implement hardware detection and also to supplement hardware detection where a small amount of software checks a large percentage of hardware. It should be noted that the desire to detect intermittent errors results in relying more heavily on hardware methods. Intermittent errors may or may not be detected by software methods when they occur. As time increases from the occurrence of an intermittent error, the probability that it will be detected increases. However, this probability may not increase fast enough to guarantee detection and reconfiguration during a critical mission phase. A discussion of the selected approach and others considered will be given below.

The first approach to be considered involves a "floating test-cell" concept. In this approach, 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, this cell then tests itself during the next program sequence. Thus, the testing is multiplexed, with all cells eventually executing the self-test. A drawback of this method is the storage limitation of an individual cell. It is questionable whether a very comprehensive self check program can

be placed in one cell. However, it should be possible to use a neighbor cell as additional storage to hold the program if this becomes a problem. The most serious drawback however is that the geometric properties of the array of cells are disrupted. Neighbor communications assignments would change as the operational programs move and secondary connections would be required for I/O signals since I/O programs would have to be reassigned as cells performing I/O are tested.

A second approach uses active redundant cells in a multiplexed manner. Referring to Figure 8-1, 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. This may readily be performed by global control of the testing cells. If disagreement exists the error is reported to the controller cell. 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, all the neighbor communication paths are not checked. Fourth, errors in circuitry peculiar to the execution of a particular instruction are detected only if that instruction is being executed while the cell is tested. Fifth, the symmetry and efficiency of test cell utilization is geometry dependent. For example, in Figure 8-1 each test cell uses all four of its neighbor 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. Finally, the controller cell would be required to contain a program for global control of the test cells and to move data between the operational cell tested and the test cell for disagreement detection.

A third approach reduces the amount of 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. It also will require some overhead in storage of the controller cell to globally control the interchange of programs, the interchange of redundantly computed data and the programs to detect disagreement.

A fourth approach uses a central self test program that will be executed periodically by the controller cell and sent out to the other cells in the group for simultaneous execution. This program may be fairly lengthy for a comprehensive check and may be

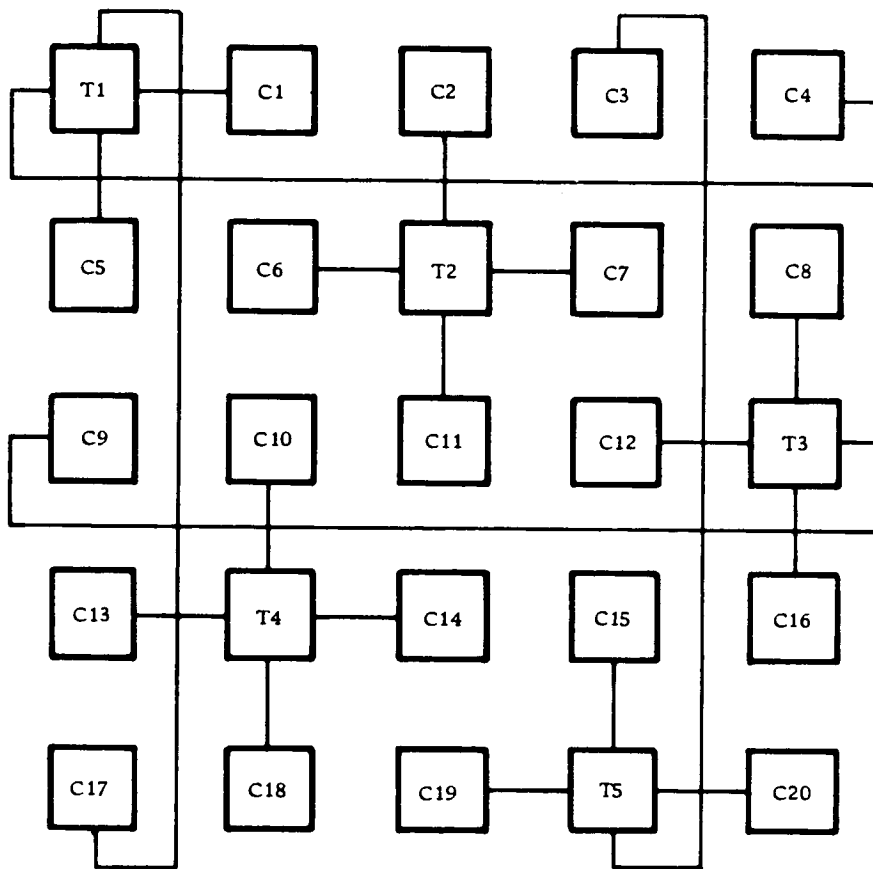


Figure 8-1. Active Redundant Test Cells Within a Cell Group

contained in a neighboring cell due to storage limitations in the controller cell. The test routine would be sent from the controller cell; it would be executed by all the cells in a global manner and also by the controller cell itself.

The cells of the group would execute a test problem in the following manner. Each cell would perform the same instruction at the same time, transmit computed results to each of its four neighbors, check the data received from each of its four neighbors against its own computed result, and report to the controller cell 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 possibly it will identify all of its neighbors as having failed. The controller cell will decode the failure reports and perform further tests if necessary to isolate the failed cell.

Finally, the last approach is to incorporate fault detection hardware into each cell. No special testing mode would be required. Checks would be performed continually along with the operational problem. Parity checking, special error detecting codes, feed-back checks on decoders, are typical of the type of fault detection hardware that would be incorporated. The disadvantage with this approach is the amount of hardware that must be incorporated to achieve a high probability of detection.

It should be pointed out here that the approach of placing self test programs that would permanently reside in each cell was not considered. This is due to the relatively small amount of storage available in each cell.

The selected approach is a combination of the last two approaches discussed above. Hardware detection methods would be used where they are fairly efficient in terms of the amount of hardware checked compared to the amount required for checking. A failure detected by hardware would set an error flip-flop which is sampled periodically by the controller cell. Software detection methods would be used to supplement the hardware methods to achieve a very thorough test and a high probability of error detection. The self test programs are sent out from the controller cell as noted above. All of the other cells in the group are in the dependent global state when these programs are executed. In general a neighbor cell will detect a faulty cell since neighbor responses are checked as part of the program. Note that this also checks out the neighbor communication paths. Part of the program may be to see if the error FF has been set. If it has, then this cell will not transmit a certain response to its neighboring cell. Detection of an error will cause a cell to change its global level. The controller cell can test to see if any cells have changed level at the end of the self test program. If none have changed the group resumes its normal operational function. If an error has occurred, the controller cell may simply ask the suspected faulty cell for the contents of its accumulator which can contain a predetermined value only upon successfully passing the self test program.

It should be noted that this approach will also detect errors in the controller cell since it can also execute the same instructions. However, the controller cell does not have another "controller cell" to report to. To resolve this problem, the approach taken was to require the controller cell to report to the group switches.

The group switches will contain BITE timing circuitry. This timing circuitry simply consists of an out of tolerance voltage detector. The setting and resetting of a flip-flop at a certain rate will produce a voltage of a certain value. If the voltage is too high or too low this will cause an error signal to be generated. The timing circuitry flip-flop can only be set by a predetermined fixed code word. This is the code word that the controller cell will contain and send to the group switches only after it successfully completes the self test program (note that if the hardware error detection FF is set the software self test program will not be completed).

The timing circuitry has been included primarily for the purpose of reconfiguration during critical mission phases as will be explained later. The executive group can also check on another group's status by interrogating the controller cell response register in the group switchers. This approach was selected over that requiring the executive group to interrogate the controller cell directly. While the latter approach may save some hardware in the group switches, it makes reconfiguration more complex. It should be noted that the controller cell response word is used to indicate failures of the controller cell only. Failures in the working cells of the group are of primary concern to the controller cell in that group. However, periodically the executive group may interrogate the controller cells of the other groups to determine what is the status of the individual group resources.

Present organizational concepts call for using two inter-group busses and two group switches per group to reduce the possibility of single failures bringing down the entire system. This results in having the controller cell response word reported to two group switches. The executive group can then check the group switches to determine if a failure is in the group switch or the controller cell. Conflicting reports from the group switches would indicate a failure in one of the group switches.

The executive group contains the highest level executive in the system. In most likelihood a backup minimal executive will be contained in another group. This backup executive can monitor the executive group for failures just as the executive group monitors other groups.

The group switches will contain logic to shut off power to the cells in their group if a failure is indicated by the controller cell response word. The reason for including this function is to prevent a group from malfunctioning in a mode that will tie up or bring down an inter-group bus. An example of this situation is where the executive group fails in a mode such that it continually outputs erroneous information over the inter-group bus.

8.2 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 results in a high probability of mission success and availability because an individual group can continue to operate in the presence of a failed cell(s), and because the system can continue to operate in the presence of a failed group(s). 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.

The mission phases were presented in Section 2 of this report and for purposes of reconfiguration analysis may be classified as consisting of three basic types: (1) Non-Critical, (2) Critical, and (3) Mars Orbital. The organizational configurations for each phase are listed below:

1. Non-Critical:	3 group; on, 18 cells per group
2. Critical:	4 groups on, 16 cells on per group
3. Mars Orbital:	4 groups on, 18 cells on per group.

These configurations are based on the requirements in Section 2 plus an estimated 30% for overhead functions. The reconfiguration plans will be based on the phase in which the failure occurs.

8.2.1 Non-Critical Phase

Reconfiguration for this phase as well as the other two phases will depend on the type of failure. The types of failures may be classified as follows:

1. Working cell fails
 - a. In a manner not affecting other cells
 - b. In a manner affecting the other cells, either bringing down the inter-cell bus or contaminating information in other cells
2. Controller cell fails
 - a. In a manner not affecting other cells
 - b. In a manner affecting other cells
3. Group switch fails
 - a. In a manner affecting only internal operation
 - b. Bringing down one of the two busses connected to it
4. Executive Group fails

Either by the controller cell in this group failing; by a working cell failing, and affecting other cells; or by a group switch failing and bringing down the intercell bus of this group.
5. Conditioner/Sensor fails

The configuration during this phase is to have three groups turned on. The computational functions are distributed among the groups. One of the groups will contain the overall system executive.

Working Cell Failures

Reconfiguration around a working cell failure will be considered first. Assuming the failure has been detected and does not affect another cell, the controller cell

will be informed of the failure and will go into its reconfiguration executive routine. It will scan its cell assignment and status tables to determine what the failed cell was assigned to do.

The controller cell does not know at this point whether the failed cell could have contaminated data in other cells of the group. This could happen if the cell failed and transmitted data to another cell. Therefore, the controller cell will assume that any data transmissions that took place from the failed cell since its last self check routine resulted in the transmission of erroneous data. As part of the reconfiguration routine the controller cell will determine which cells the failed cell has communicated to and inform these cells to ignore these data or any computed results based on these data (if this is not possible to mechanize, it may be required to reinitialize the entire group).

Continuing through the reconfiguration routine the controller cell determines if there are any spare cells available in this group. If there are any spare cells available, the controller cell will enter its full capability assignment routine. This will involve assigning the failed cell's program (reloaded from the bulk storage unit) to either a spare cell or moving cell assignments around to maintain certain geometrical relationships between assignments (this will occur when neighbor communication capabilities are being used).

If there are no spare cells left in this group the controller cell can take one of two courses of action: (1) it can ask the executive group if this group's functions or part of the functions can be reassigned to another group or (2) it can reduce or eliminate some of the functions being performed in this group according to some degradation procedure. Obviously the executive group must maintain tables of active/spare cells in the various groups and also tables of a certain level of functions assigned within the groups with degradation procedures. It should be noted that degradation may take place in two forms: (1) lowest priority functions may be completely eliminated from the computations or (2) the computational rate may be reduced on some functions, i.e., the programs may be loaded in periodically from the bulk storage unit and essentially time share storage in the computer with other low priority programs.

The reconfiguration executive programs are discussed in the software section on the distributed processor executive routines (section 11).

A working cell failure that affects other cells in the group will be discussed next. If it fails in a manner bringing down the inter-cell bus, the controller cell will not be able to respond to the group switch with its status word. The group switch will then indicate a failure in this group. The executive group will attempt to isolate the failure following the procedure outlined below for a controller cell failure. This type of failure will cause the entire group to be rendered useless (unless the bus is not affected when power is turned off to the individual cell).

If the working cell contaminates information in the controller cell then the controller cell will indicate a failure when it enters the self test mode and the group switch will indicate a failure in this group. How this failure is treated will be discussed below under controller cell failures.

Controller Cell Failures

Failures of the controller cell whether or not they affect other cells will be treated in the same manner. The group switches will not receive the proper response word and this will be detected by the executive group when it samples the group switches. The executive upon detecting the failure has no further information as to what the specific failure in the group is. It must assume that the entire group is down. An example of an entire group being brought down is a failure in the controller cell that results in erroneous global commands to all the cells in group. As a result the cells can be in totally false states or levels.

The executive group therefore has to assume that it has no capability to access the cells in the group. Communications with the failed group will be initiated by an external source, similar to that used to initially start up each group. Upon detection of the failure, a light on the display panel will be lit indicating which group has failed. A new cell in the group will then be initialized as the controller cell. This is accomplished by initializing the cell to the controller state via this cell's I/O line following the normal start up procedures. It should be noted that this may require the astronaut to plug in a new connection to the computer I/O panel. This can be circumvented if an alternate connection is initially provided that would be used whenever a group failure is indicated. In any case a new cell is attempted to be started up as a controller cell by having the executive group load in a self test routine to be exercised by this cell. This routine could be contained in the executive group or called in from the bulk storage unit.

If the new controller cell is functioning properly it will issue the proper response word to the group switches as is required in normal operation. The executive group will check the group switches for the proper response word. If the cell is functioning properly thus far, the executive group may attempt to communicate with the new controller cell to further check out the cell. When the cell is finally determined to be functioning properly, the executive group will command the controller cell to load up and check out the remaining cells in the group. It will also give the controller cell a list of cells that were known to have failed prior to the group failure so that the controller cell can avoid testing cells already known to have failed.

The controller cell will then begin turning on and naming cells via the neighbor lines. It will checkout one cell at a time until all cells, exclusive of those listed by the executive group, have been tested and their status determined. After a fixed period of time the executive group will interrogate the controller cell to determine what the failure was. The executive group working with the controller cell will now reconfigure the group. The reconfiguration routine will consist of one similar to that described for the working cell failure where the programs are attempted to be reinstated in the group and secondly in other groups if they cannot be efficiently mechanized in this group.

It may be possible that the failed cell that caused the group to indicate a failure, could have transmitted erroneous data to another group. The procedure that will be followed is exactly identical to that followed by the controller cell when a working cell fails. Namely the executive group will inform other groups to ignore data received from the failed group or computed results based on such data since the time

the last self check was run by the group that failed (the executive group will always interrogate the controller cell in a group as to the time of its last self check prior to allowing it to transmit data on the inter-group bus).

It should be noted that the failed cell may bring down the bus when it is turned on. Should this occur the controller cell will turn it off via the neighbor communication lines. This should be accomplished in time for the controller cell to continue its reports to the group switch.

If the initial cell identified as the new controller cell fails to respond properly to the group switches, another cell will be attempted to be initialized as the controller cell. This procedure would be followed until some number of cells in the group have been attempted to be started up by the executive group. If it is not possible to get the cells started, the executive group will declare the group as having completely failed. This situation can exist if a cell fails in a mode permanently bringing down the bus.

This procedure will require the executive group to output the results of testing the group to a display panel and inform the astronaut of the necessity to initialize new cells as controller cells.

If the group is declared as having totally failed, the executive group will go into a specific assignment program to reinstate the program of the failed group. This will involve first checking to see if any spare groups are available in the system (the non-critical phases will initially have 1 spare group since only 3 groups are required here and 4 in the most heavily loaded phase). If a spare group is available, it will be started up using the normal start up procedure. However, if no spare groups are available an assignment program will be entered that will allocate the programs to the other groups in the system. This will probably result in a degraded mode of performance. The assignment program will be quite similar to that used when a working cell fails except that it must contain specific information on all the tasks currently assigned to the system.

Group Switch Failures

If one of the group switches fails and does not affect any of the busses, it will simply not be used. As long as spare group switches are available the group can continue functioning. The executives in all the groups must be informed of the failed group switch so that they may update their communication routines to not use this group switch for communications with this group.

A group switch may fail in a manner bringing down the intercell bus of its group. The executive group will detect this failure when it samples the group switches for the controller cell response word. This failure then appears identical to that treated above for the controller cell failure. An identical procedure will therefore be followed to start up the failed group. One group switch will be turned on first to check out the cells in the group. Assuming this is the failed group switch, it will be impossible to get the group initialized with a controller cell. This group switch will then be turned off and another one, if available, turned on and the procedure repeated. If this group switch is functioning properly and the failed group switch does not affect

the intercell bus when it is turned off, the group will be initialized. The executive group has then isolated the failed group switch and proceeds as discussed in the preceeding paragraph. If the failed group switch affects the intercell bus when it is in an off state, the entire group will be declared faulty.

A failure of a group switch that brings down an inter-group bus will be detected by the executive group when it tries to communicate to other groups over this inter-group bus. The group switches connected to this inter-group bus will then be turned off one at a time until the bus is operative once again. The failed group switch will then be kept off and the executive group will reconfigure around the failure as described for the previous two cases above. If the failure results in bringing down the inter-group bus when the failed group switch is in an off state, the inter-group bus will simply not be used.

Executive Group Failure

A failure in the executive group of the type described above under "controller cell failures" will result in the executive group's group switches reporting a failure. First it will be assumed that there is a minimal backup executive in the system contained in some other group. This backup executive monitors the executive group for a failure. Upon detecting the failure, it will output a signal to the display panel informing the astronaut of the failure. The astronaut can check this report against the go/no-go signal from the executive group's group switches. Assuming the two reports agree, the astronaut will proceed to reconfigure around the failure of the executive group by having the minimal backup executive enter its reconfiguration routine (which is what the backup executive primarily consists of). This routine has no alternative but to assume the executive group could have contaminated data and control in any of the groups in the system. (The minimal backup executive cannot be altered therefore any failures external to it should not affect it). Therefore it assumes the entire system must be reinitialized.

The minimal backup executive will now load in from the bulk storage unit a new executive group routine into the group in which it is residing, preserving a record of where the executive group resided when it failed. The new executive group will first check to see if any spare groups are available. If a spare group is available, the system is reloaded as is done when initially starting up the system. After the system is started up, the new executive group will inform the astronaut that the previously failed executive group should be checked out in order to isolate the failure and possibly return the group to a spare status. The procedure for checking the failed group is exactly identical to that described above for a controller cell failure (working group failure). The routine for assisting in checking out the failed group is of course a normal part of the executive group functions.

However, if the situation occurs that a spare group was not available, the new executive group would proceed to check out the failed group prior to loading up any other groups. In this manner it would know whether the failed group can be placed back in service. If it has completely failed, then the new executive group would proceed to its degraded mode reconfiguration routine as previously described above.

If a minimal backup executive is not contained in the system then the astronaut must reinitialize the system just as if the system were being first turned on. The primary difference here is of course that he would not use the failed group in reconfiguring. He would load up a new group as the executive group and give it the location of the failed executive group so that the same procedures as outlined above may be followed by the new executive group.

It may not be very difficult for the astronaut to essentially act as the minimal backup executive. However, the minimal backup executive should be relatively simple and would probably fit into a single cell.

In any case, a failure of the executive group may be considered as a hard core type of failure. It essentially requires the system to be completely reinitialized as from a cold start.

Sensor/Conditioner Failure

It is possible for sensor/conditioners to be connected to the cells, the inter-cell busses, and the inter-group busses. Assuming the failure has been detected and isolated the system would continue operating in a mode not using I/O associated with the failed device until a new one is plugged in to the I/O panel replacing the failed device.

A failure may occur in a sensor/conditioner bringing down the part of the system it is connected to. For example, bringing down the inter-cell bus, causing a group failure indication. In such a case, if the group cannot be started up following the procedure discussed above for controller cell failures, the sensor/conditioners would be disconnected by the astronaut one at a time thereby isolating the failure. Similarly the same procedure would be followed for failures on inter-group busses and the cells themselves.

8.2.2 Mars Orbital Phase

The configuration during this phase requires 4 groups of 18 cells each. One group will be designated as the executive group with the system computational functions divided up among the groups.

Working Cell Failures

The same procedure is followed as in the non-critical phase. There will be a minimal Navigation and Guidance function being performed in another group. If a cell fails which is concerned with the full Navigation and Guidance computation, the minimal program will take over to prevent complete loss of data until the full navigation and guidance programs are reassigned. A complete loss of data will result in a reinitialization of navigation and guidance parameters which could take approximately a half hour. This could be detrimental say in correlating video data with position information. While this is not a critical type of backup function, it is one that can enhance the system availability.

Controller Cell Failures

The same procedure as outlined for the non-critical phase is resorted to. Note that if the failure is in the group performing the full Navigation and Guidance function, part of these functions would be loaded into the group with the minimal backup Navigation and Guidance function. Also note that this phase requires the full computing capability; therefore there may not necessarily be spare groups to resort to in case a group is declared faulty.

Group Switch Failures

Same as non-critical phase.

Executive Group Failures

Same as non-critical phase. Note however that there is less likelihood of having a spare group in this phase.

Sensor/Conditioner Failures

Same as non-critical phase.

8.2.3 Critical Phase

The configuration during this phase consists of 4 groups turned on. Groups 1 through 3 are considered the primary system, one of these groups will be identified as the primary executive group. Group 4 will be considered the secondary system. It will contain its own executive and the critical mission computations. Note that this also implies that there are redundant connections to the critical sensor/conditioners.

Working Cell Failures

The cell that failed or the cells that it affects can be involved with critical or non-critical computations. Failures are detected as before in the other two phases. If the failure is involved in non-critical computations, the controller cell will determine if any spare cells are available in the group to assign the failed computations to. They would be loaded in from the bulk storage unit. There would be no reassigning of programs in the group to affect a certain geometrical reconfiguration of the programs as may be done in the other two phases. If no spare cells are available, the failed cells computations would be simply suspended. To reassign programs to other groups would probably require plugging/unplugging sensor/conditioners to effect a reconfiguration and it is doubtful this would be done in a critical phase for non-critical computations.

A failure associated with critical computations will result in the controller cell issuing a faulty response word to the group switches which will indicate group failure. The group switches will output the group failure signals to switches that control the steering of output signals to the critical sensors as shown in Figure 8-2. This will result in switching over control to the secondary system as far as the critical sensors are concerned. The executive group in the meantime will have also noted the failure.

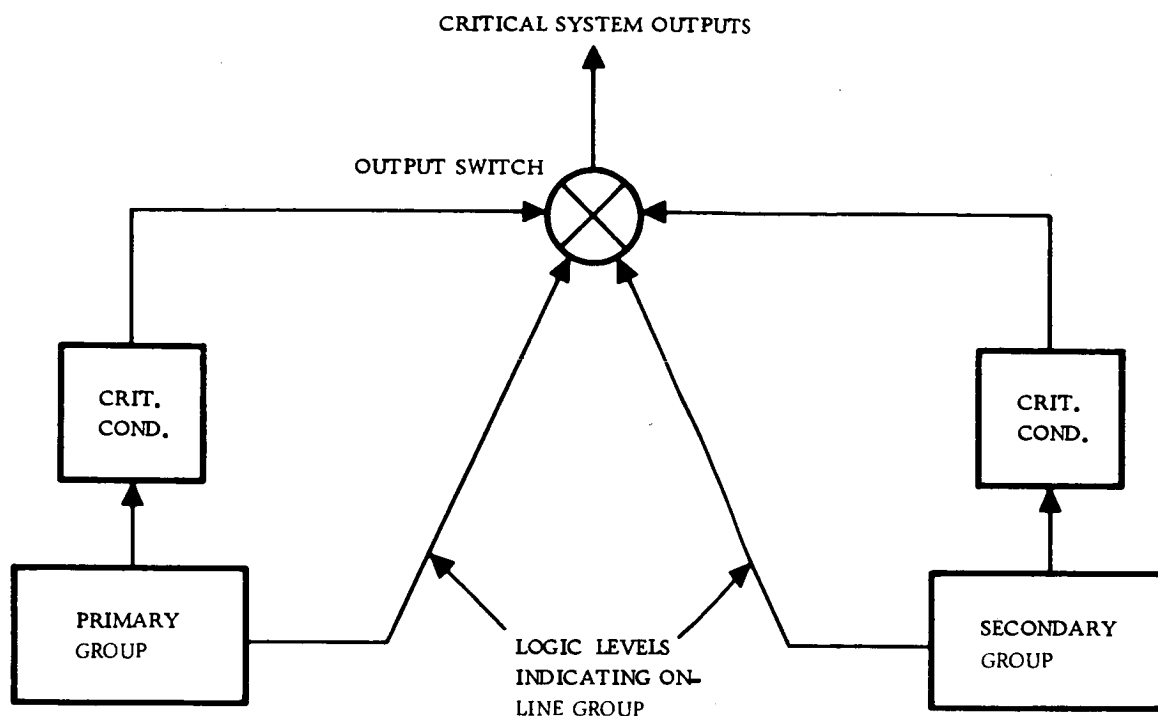


Figure 8-2. Output Switching of Critical Conditioners

At this point it does nothing. The controller cell will attempt to reconfigure within the group so that the critical computation can be reinstated and serve as an additional backup if the remaining system failed.

The executive group will therefore wait a fixed amount of time to determine if the group has been reconfigured internally. If the group has been reconfigured, it must then be restored to operational status by providing it various computational parameters. 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 in another cell or group during operation, or by obtaining the latest values from the system which did not fail.

If the group has not been able to reconfigure itself, the failure will be treated as discussed under controller cell failure given below.

Note that the above discussion applies to the primary and secondary systems except that in the secondary case no actual output switching takes place. It should also be noted that the executive group reconfiguration routines are simpler than those in the prior two phases.

Controller Cell Failures

As above the group switches will control the output switches to switch control to the operating system for the critical sensors if the failure involves a group with critical computations. The executive group will follow a procedure similar to that

given for the prior two phases as far as attempting to check out and isolate the failures in the failed group. Unless alternate connections were provided for in each group, so that the astronaut could quickly initiate such action upon a signal from the executive group, it is doubtful that the astronaut would plug in the necessary connections, although it could be done.

If the failed group were reinitialized, it could be loaded up with critical computations to serve as a backup for further failures. In the meantime, prior to checking out the failed group, the executive group will reload one of the other groups with a set of the critical computations from the bulk storage unit if the failed group contained critical computations. In this way another group could serve as a backup to further failures in critical computations. Note that this capability would require redundant I/O connections to groups unless the astronaut plugged/unplugged the connections during the critical phase (this may not be practical for him to do).

Group Switch Failures

Same discussion as in prior two phases applies here. The controller cell failure procedures outlined above will be followed here in attempting to reinstate the failed group.

Executive Group Failure

If the executive group's group switches indicate a failure and a minimal backup executive exists as discussed in the prior two phases, the backup executive can follow the same procedure as discussed previously. It would probably reload the executive group function in its own group (assuming it initially contains no critical computations) and attempt to bring up the failed group as is normally done by the executive group function. Once again the comments about connections to the groups and requiring the astronaut to make connections applies here. Note that the backup executive can only be located in one specific group since the other two non-failed groups would be associated with critical computations.

If a backup executive were not provided the above would not be possible except by astronaut participation. However, the two groups assigned the critical computations would continue operating normally.

Sensor/Conditioner Failures

Failures associated with non-critical functions result in suspension of the associated computations. However, if they affect a critical function, the controller cell would simply indicate a group failure and result in switching over control to the operating system. Unless redundant connections were provided, the astronaut would have to plug/unplug connectors to reinstate another backup system for critical functions.

It should be noted that in the above discussion for the critical phase, failures in one group affecting other groups are not mentioned. The reason for this is that the programs in the groups are expected to be well isolated and contain check routines to validate any data received from other groups. They will also have routines to

validate requests for any control changes or locations that data may be read into. In this manner each group essentially locks out other groups except for certain predetermined functions which will not affect this group if another group failed that it is communicating with. The group switches will also contain logic to inhibit their group from getting on the inter-group bus should they fail to get the proper response word from the controller cell. In this manner, it is very unlikely that one group's failure will affect the remainder of the system.

It may be seen that the organization contains many possibilities for sustaining multiple failures during a critical phase. This is particularly true if a number of extra I/O connections are provided to the system as discussed above.

8.3 BACKUP EQUIPMENT ASSURANCE

There may exist at any time in the system a number of spare cells in each group. These cells will be periodically tested along with the other cells in the group to determine their status.

A spare group will exist during the non-critical phase. This group will be turned on periodically by the executive group and told to run the standard self check routines. Therefore, the executive group will be informed of its status for possible use in reconfiguration. This status will also be required prior to entering the critical or the Mars orbital phase.

8.4 EXTERNAL STATUS REPORTS

The executive group is the primary central source for status reports. It will issue reports to the display panel that a group has failed and a new controller cell must be initialized. It will also indicate the number of failed and operable cells in each group. Each group will also have a connection from its group switches to indicate a failure in a group. This report could be compared with the executive group's report for any discrepancies in the system. The executive group will also issue status reports on the group switches in the system and on the inter-group busses.

The cell that the backup executive is in will also have a connection to the display panel (via the cell's I/O line) to indicate any failures in the executive group.

9. CELL AND GROUP SWITCH DESIGN

This section presents a detailed description of the cell. First the features are presented and secondly some detailed aspects of the design are discussed. Finally some design considerations of the group switch are given in 9.3.

9.1 PROCESSOR FEATURES OF THE CELL

This section will describe the general features of the processor section of each cell. In particular, the word length, accumulators, index-bank registers, and the instruction word format will be discussed. Since the requirements to be designed to are not very firm at this point in time, some of the alternatives discussed cannot be explicitly chosen as optimum. In addition, some general considerations with regards to the instruction set will be given.

9.1.1 Accumulators and Index Registers

This sub-section will discuss the use of multiple accumulators and multiple index registers in terms of their ability to save storage. Since it is not necessary for the processor to operate at high speed, as many of the processor registers as possible will be stored in the memory. This enables many registers to be used for the processor at a small increase in system complexity or lowering of wafer yields. This is the case since registers in the memory can be easily fabricated with high yields by using discretionary wiring or similar techniques. On the other hand, the registers constructed in the processor are generally part of lower yield complex logic. (If this differences in ease of fabrication is eliminated in the future the multiple accumulators and index registers can be included in the processor section of the chip. The instruction execution time would then be decreased.) Because of the above points there will be just one accumulator in the processor section of a cell and any additional accumulators will be accessed from a specified area of the memory. (The chosen processor uses four accumulators.) The index-bank registers will also be contained in a fixed area of memory.

Accumulators

The use of more than one accumulator can save a significant amount of execution speed and, of more importance for this application, storage. The storage savings comes about since intermediate results do not have to be stored in the memory or in hot storage. As intermediate results are obtained they are simply left in the accumulator in use and another accumulator is brought into use. In this way it is not necessary to store the first accumulator while further operations are carried out before the intermediate value is again needed. The accumulator to be used in any operation is simply specified in the op-code. As a result when the instruction is to be executed the proper accumulator is pulled out of the memory and exchanged with the accumulator in the hardware location or if the hardware accumulator is the one specified no exchange is necessary. This process clearly takes a longer time than if the accumulators were in the processor hardware itself; however, for the processor of interest the main interest is in saving storage and as a result the processor hardware that would be devoted to multiple accumulator registers can be eliminated or used in other fashions, e.g., for complex macro instruction control, etc.

The usefulness for memory savings of a second, third, or more accumulators for intermediate storage must be evaluated for any given application. However, an evaluation that was carried out (Ref. 29) showed that for guidance and navigation in an avionics system addition of a second accumulator reduces the instruction count by as much as 8 percent and the inclusion of six or more accumulators bring this reduction to as much as 12 percent. How much the saving is for scientific experiments, telecommunications, etc. has not been determined, but it is clear that the use of at least 2 accumulators is a very valuable asset for saving storage. Since these accumulators are in memory, their only hardware cost is additional control circuitry. It can be seen that the majority of the advantage of multiple accumulators is accrued with the addition of the second accumulator, as a result each processor cell will have at least two accumulators. The inclusion of additional accumulators depends on both the availability of instruction bits to specify to which accumulator a particular op-code is being applied and the relative usefulness of additional accumulators versus additional index registers. These points are made clearer by the discussion on each possible word size given later.

Indexing

Full word length index and bank registers are used. Therefore, there is no real distinction between the two since they both accomplish address generation and address modification in the same manner. They will be referred to as index, bank, or index/bank registers throughout this report.

Indexing in each processor of the cells will be carried out by memory index registers. For an instruction that requires banking or indexing, the proper index/bank register or registers are accessed from the memory, loaded into the memory buffer register, and added to the memory address register (the memory address register holds the address displacement obtained from the initial instruction word). From this it can be seen that an instruction that needs to be banked and indexed before picking up the operand would require four memory cycles, including the memory cycle to pick up the instruction itself (accumulator may be in memory).

The advantage of indexing in terms of memory saving has been discussed in many places. Two such discussions are given in References 29 and 1. From these references and from an investigation of the requirements, it can be seen that the inclusion of at least three index registers along with one or more bank registers will provide significant storage savings (20 percent or more); however, the addition of more index registers than three provide significantly less storage savings. As a result, at least three index registers will be included in the processor of each cell, the use of additional index registers depends on the availability of instruction word bits and on a comparison of the value of these additional registers to the value of additional accumulators. The index registers could also be used for temporary storage; however, use of the index registers in this manner would provide no memory saving unless register to register instructions were included (instructions carrying out basic operations such as add from one index register to another index register). The reason for this is that the index register used as temporary storage cannot be addressed directly by bits in the op code if an operation is to be carried out between the temporary storage and a memory operand. This is clear since there are index register bits in the instruction word that must specify the indexing of the address for the memory operand. Therefore, only operations that address two index registers used as temporary storage and then add or subtract them, etc., could provide

an instruction saving over the use of memory addressing for intermediate results. On the other hand note that if accumulators are used for temporary storage, they may easily be addressed by additional bits in the op-code portion of the instruction word since these registers will not be used for indexing a memory address. Investigation of the usefulness of register to register operations versus the usefulness of providing accumulators that may be used for temporary storage has shown that the register to register operations find very little usage in comparison to the accumulators. As a result using accumulators instead of index registers for temporary storage provides a much greater storage savings. (Reference 1 discussed the use of hardware index registers for temporary storage in order to provide increased execution speed by using these registers for temporary storage. Note that this increased speed cannot be obtained for the processor specified above since the temporary storage or index registers are located in the memory and must be accessed with a memory cycle in a similar fashion to any other operand held in the memory.)

Indirect Addressing

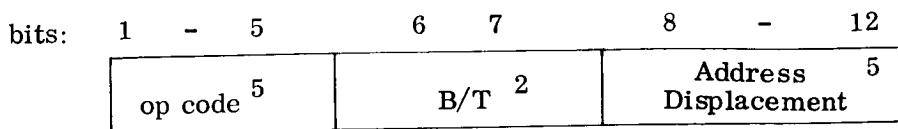
Investigations of the usefulness of indirect addressing have been carried out and are discussed in reference 1. It was found that indirect addressing in a machine with a number of index registers had a limited usefulness; however, when it was used, it provided some storage savings. (The primary use was for sub-routine linkage.) As a result of this limited usage it is not recommended that an indirect bit be added to the instruction word, but that instead where applicable, certain instructions may use only indirect addressing or may have the facility to use indirect addressing if desired.

9. 1. 2 Word Size

The desire to save storage (to use the least number of bits in the memory possible) gives a reason for considering small word sizes. If a small instruction word can include enough features, it may provide enough flexibility such that it would require only a slight increase in the number of words for instructions in the memory, over that for a larger instruction word. This may then result in a smaller number of bits in each cell's memory for instructions. Larger instruction words are generally used to offer more flexibility and increased processing speed. The increased processing speed is not required here, but the amount of storage saved by increased instruction word flexibility must be investigated. It is also necessary to determine the amount of extra data words that would be necessary with a small word size. A small word would require increased double precision and possibly some triple precision operations and would result in smaller byte sizes for storage of multiple bytes per word.

Twelve-Bit Word

A 12-bit instruction and data word was first investigated to see if it would offer enough flexibility for a savings in the number of bits in the memory over a larger word. The chosen 12-bit instruction word is shown in Figure 9-1. Five op-code bits should be sufficient to offer a reasonably large and flexible instruction repertoire including the ability to use 2 accumulators. More than 32 instructions can be made available by the use of op-code extension on instructions that do not require a memory address. Two uses of the tag bits B/T, are shown in Figure 9-2. Figure 9-2a would have a bank register, B, contained in the memory added to the address bits in the



B/T = index/banking bits

Figure 9-1. 12-bit Instruction Word

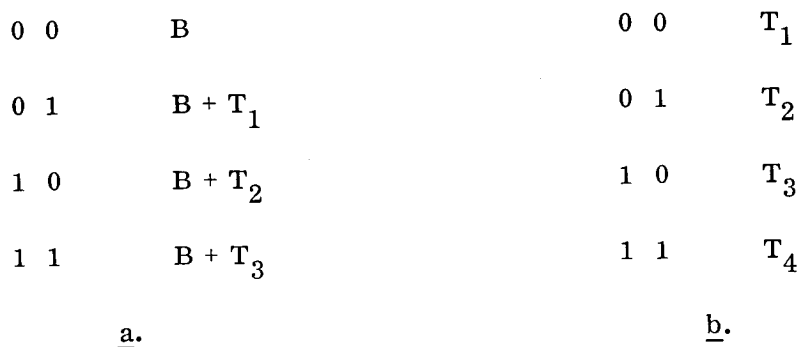


Figure 9-2. Use of Two B/T Bits

instruction word for every memory access. In addition, one of three memory index registers can also be added to the address. Figure 9-2b proposes using the index/banking bits to specify one of four index/bank registers. Since this scheme does not enable multiple indexing to be carried out, it will require a few more instructions for execution of loops. However, it does not have the disadvantage of the scheme specified in 9-2a, of requiring that the index register contents be changed any time the bank register contents are changed. None the less, the scheme in Figure 9-2a, seems to be somewhat more flexible and would be chosen for a 12-bit word.

The last part of the instruction word provides 5 bits for specification of an address within a bank. Clearly this is only a 32 word bank; as a result it is probably too short to hold the majority of the programs and/or their associated data. This means that a reasonably large number of load bank commands would have to be inserted into the instruction stream both for jumping to separate parts of a program that is located in a number of banks and for picking up data for separate banks. A complete investigation of the programs is necessary to determine the percent increase in storage due to these load bank commands; however, an investigation (reference 30) showed that for navigation and guidance programs a 32 word bank could cause substantial increases in the amount of memory required. This short bank is especially inefficient in a 12 bit word where the index/banking scheme is relatively inflexible.

There are a number of additional problems with a 12-bit word. One of these would be the inability of a memory word to contain the complete address for any word in a group. A group may typically contain on the order of 20 cells of 512 words per

cell. This would amount to at least 10,000 words per group, whereas a 12-bit word can only address 4,000 words. As a result, a 12-bit word would require two locations to hold this address and the second location would only need two of the 12 bits for additional addressing. This could amount to a substantial inefficiency.

In addition, the use of a 12-bit word would certainly require triple precision operations to be carried out in the navigation and guidance routines. This could cause an inefficiency of bits in the data word and would also require the addition of triple precision software. It also appears from the requirements that the half word size or byte size of 6 bits would be somewhat small for the needs of many of the scientific experiments and other operations that will use byte manipulations. In particular, a seven to eight bit byte would probably be necessary to offer sufficient flexibility. A precise answer to the size of the byte required for efficient utilization of data storage is difficult to provide; however, 7 or 8 bits would certainly offer more flexibility to meet the requirements for byte manipulation when they are explicitly specified.

It should be noted when discussing byte manipulation that the number of bytes per word should be a power of two for the most ease of operating on bytes. (The most flexible byte manipulation is when the number of bits in the word is a power of two.) If this is the case bytes can be manipulated and obtained from data words by simple shifts of the addresses of the bytes. In addition, indexing with respect to words or with respect to bytes can be done simply by adding or by shifting and adding the specified address to an index register.

The addition of a number of half word or byte instructions to the instruction repertoire, such that bytes are directly accessed and operated on and then replaced in memory without affecting the remainder of a memory word, may save a considerable amount of storage (e.g., in the scientific experiments in a spaceborne application). The addresses of these instructions could handle the additional length required due to byte specification since they would be relative to an initial address of a word in a list. (The initial address would be held in an index register.) Therefore, since a considerable amount of byte manipulation is expected in at least the scientific experiments, only word lengths that are a power of two and a multiple of some useful byte length, such as 6, 7 or 8 bits, will be considered, i.e., 12, 14, and 16 bit word lengths. Longer words that are a power of two and a multiple of a byte and that would hold two instructions per word could also be considered from the above standpoint (24 bit words, 28 bit words or 32 bit words). This may result in problems in trying to pack data into the word for efficient byte manipulation. However, the most important point is that for the requirements considered for the given space missions, the majority of the words tend to be 12 to 16 bits and as a result a processor with a considerably longer word length can result in inefficiencies of storing data. From the above it can then be seen that there appear to be no real gains from a long word and there are some losses.

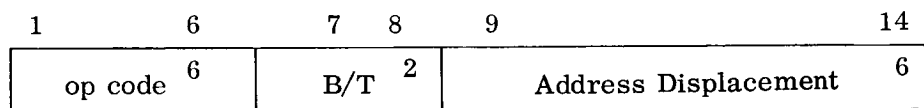
Because of the considerations given earlier a 12-bit instruction word has been eliminated as a possibility. It does not seem to provide sufficient flexibility to save storage over a longer word length. In fact, it appears that it would require considerably more storage primarily due to the addressing and short bank problems.

14-Bit Instruction Word

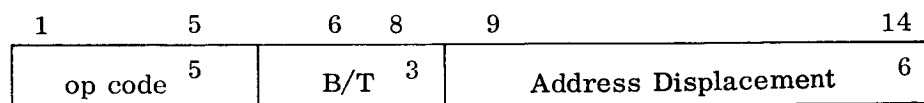
Two examples of the 14 bit instruction word are given in Figure 9-3. Figure 9-3a shows 6 bits used for the op-code. This should easily be sufficient to offer instructions to take advantage of the multiple accumulators (as many as four would be practical), to provide byte manipulation instructions, and even to provide some complex macros. Two bits are provided for index/banking using the same scheme as discussed in relation to figure 9-2. The address section of the instruction word, provides six bits for a 64 word bank. This bank should be sufficiently long for some flexibility since full length bank/index registers are used. However, since only a maximum of 4 index bank registers are available from the B/T bits, this 64 word bank could provide some inefficiencies.

The addition of more index/bank registers could alleviate much of the possible addressing problem due to the 64 word bank. Such a scheme is shown in the instruction word of Figure 9-3b. Here the op-code bits have been decreased from 6 to 5, but the B/T bits have been increased to three to offer the usage schemes shown in Figure 9-4. There are clearly other possibilities of using 3 B/T bits, but those shown in Figure 9-4 offer the most flexibility. The advantage of the scheme in Figure 9-4a over than shown in Figure 9-4b is simply the availability of more total registers that can be used for index-banking purposes. However, Figure 9-4b offers 5 registers and a considerable amount of flexibility in terms of multiple indexing. In particular the index registers do not have to be adjusted every time a bank register is changed. (This is the case in the scheme in Figure 9-4a). Because of this multiple indexing flexibility and the possibility that it may save some storage, the scheme in Figure 9-4b would be chosen. The instruction word in Figure 9-3b also has six address bits for a 64 word bank.

An explicit decision to choose between the instruction words shown in Figure 9-3a and 9-3b cannot be made until an investigation into the use of various op-codes is carried out. After this a relative evaluation of the use of an additional bit for a 6-bit op-code or for a three bit B/T specification can be evaluated so that either Figure 9-3a or 9-3b can be chosen for a 14 bit instruction word. The question of course would center around which scheme provided the most storage savings; however, there are additional considerations, such as, the ease of programming, etc.



9-3a



9-3b

Figure 9-3. 14-bit Instruction Word

0 0 0	B	0 0 0	B_1
0 0 1	$B + T_1$	0 0 1	$B_1 + T_1$
0 1 0	$B + T_2$	0 1 0	$B_1 + T_2$
.	.	0 1 1	$B_1 + T_3$
.	.	1 0 0	B_2
.	.	1 0 1	$B_2 + T_1$
.	.	1 1 0	$B_2 + T_2$
1 1 1	$B + T_7$	1 1 1	$B_2 + T_3$
<u>a.</u>		<u>b.</u>	

Figure 9-4. Use of Three B/T Bits

A 14-bit word may be an efficient choice for the computation system in the space missions under consideration, however, it does have some inefficiency problems due to its relatively short length. For example, double precision operations of 28 bits will not be sufficient for some navigation and guidance systems. As a result, triple precision will be required, but a triple precision word containing 42 bits will offer greater accuracy than that required and consequently will waste some amount of data storage area. In addition, the use of triple precision will require triple precision software to be added. There is also some question as to whether a seven bit byte will be sufficient for byte manipulations in the scientific experiments.

There is, therefore, some push toward a 16-bit instruction word due to additional flexibility in the instruction word, the use of 8 bit bytes, and more flexibility in bit manipulation. The latter point will be made clear by a renewed consideration of the discussion of byte manipulation given earlier. If a considerable amount of bit manipulation is necessary in the computations (in other words manipulation of bytes that can vary in length from one bit to eight or more bits), the use of an instruction word with a number of bits that is a power of two would be useful. These varying length bytes can then be packed into words and can be accessed by instructions by simple shifts of the address in a fashion similar to that for the half word bytes discussed earlier. All that is necessary is to place the address of the first word in a list in an index register and then to address all varying length bytes relative to this initial address by using an address in the instruction word that represents the bit number in the list. For example, a 16-bit word would simply require that the bit address be shifted four positions to the right and then indexed with the initial word address in the list. This would give a word address of the word that contained the required bits. The four bits that were shifted right would be saved and used to choose the particular starting bit in the chosen word. (The bit address can also be indexed if desired.)

Clearly, if a fairly large amount of bit manipulation was necessary, a word size that is a power of two could provide substantial storage savings in that simple instructions could easily be included to automatically carry out the bit manipulations. For example, a single instruction to load the accumulator with some desired byte would simply pick up the address bits from the instruction word, shift them four positions to the right, save them, index the shifted address with the register specifying the initial address in the word, pickup the word, and then left adjust it to the specified starting bit location given in the instruction. Additional instructions to add another set of bits to the ones that were loaded could also be implemented. These byte manipulation instructions would take a reasonable amount of time; however, they would only require a very small number of instructions for very complex manipulations.

The actual amount of byte manipulation required in the programs would have to be extensively investigated before the above discussion could be used as a strong reason for choosing a 16-bit word over a 14-bit word. However, there are also a number of additional points to be considered when trying to choose a 16 or 14 bit instruction word for the cell processor and memory. A word size increase from 14 to 16 bits would increase the number of memory bits 14 percent if the same number of total words were required. Therefore, in order for the 16-bit word to be more efficient in terms of storage savings than the 14 bit word the additional features and flexibility gained with 16 bits would have to make up 14 percent or more of the memory. The factors contributing to a decrease in the number of memory words with a 16-bit word are the following: less instructions would be required due to the ability for additional indexing or longer banks and/or additional op-codes, there would be no requirement for triple precision operations for data storage with a 16-bit word, and there may be more flexibility of byte storage as discussed above. A precise comparison of the storage usage of a 14- and 16-bit word cannot be carried out until the requirements are explicitly specified in the future. However, a rough evaluation seems to show that there would not be a sizable memory difference between the two approaches since the additional features possible with a 16-bit word would off-set the increase in bits per word by a reduction in words required. As a result, a 16-bit word will be chosen for the present design of the distributed processor cell since it will offer additional flexibility in terms of meeting a variety of requirements while providing somewhat greater programming ease. If in the future when a distributed processor is to be designed explicitly for a specified mission or set of missions and the requirements can be clearly specified, the precise trade-off can be carried out to decide between a 14- and 16-bit word. The features used in a 16-bit word are given in the next paragraph.

16-Bit Word

Two useful instruction word formats for a 16 bit word are shown in Figure 9-5. Clearly, there are other formats that are possible, but the two chosen appear to be the most applicable to the space mission requirements. One other variation that would use seven op-code bits and a lesser number of address or B/T bits may be of some interest. The additional op-code bit could be used to take full advantage of multiple accumulators (four or more) and to provide an extensive set of macros to try to save storage. However, a preliminary evaluation of the instruction set and macros seems to indicate that a six bit op-code would be sufficient; as a result, only the two 6-bit op-code formats are shown in Figure 9-5. The instruction word in Figure 9-5a uses 6 op-code bits, three bits for index banking (either of the schemes shown in Figure 9-4 could be chosen), and a 7-bit bank. The seven bit bank should provide very

few inefficiencies in terms of requiring load bank commands for jumping from program to program or from one data bank to another. The instruction word in Figure 9-5b uses a 6-bit banked address and uses the additional bit to obtain 4 B/T bits. These bits can then be used for two bank registers and any one of seven index registers. This gives a powerful banking indexing and multiple indexing capability, such that a 64 word bank may cause very few inefficiencies. Further evaluation is certainly necessary to obtain an accurate tradeoff between these two 16 bit instruction word formats. However the indication thus far is that the 7 bit bank will be more important than 4 B/T bits. Therefore the instruction word shown in Figure 9-5a will be chosen. It should also be pointed out here that a number of instructions will be added to the op-code list in order to provide a reasonably flexible byte manipulation capability in each cell. As mentioned earlier, these will not require extensive processor hardware when a 16 bit instruction word is used (a power of 2).

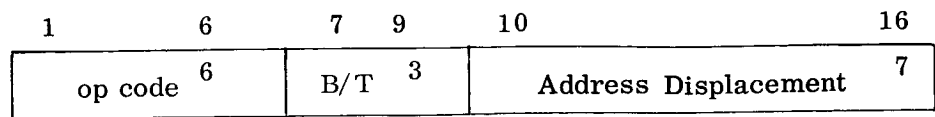
18-Bit Word

An eighteen bit instruction word with the format shown in Figure 9-6 could also be considered. However the addition of a longer bank and more op codes would not save enough memory bits to warrant the increase from 16 to 18 bits. This can be realized since the 16 bit word has very little memory restriction from either lack of op codes or bank size. The eighteen bit word would allow bigger bytes, but it provides a less flexible word size for bit manipulation (word not a power of 2). The primary advantage of this larger word would be in terms of speed increases both due to slightly less double precision operations and to a larger and more flexible instruction set. (The instruction set for the 16-bit word will not contain a number of instructions aimed at speed savings.) Therefore since storage is of primary concern in the distributed processor, it is felt that an 18 bit word is not necessary.

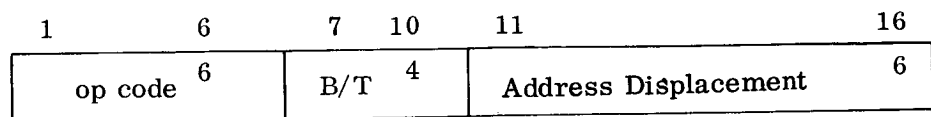
9.1.3 Control Hardware

The control hardware in the cell's processor section could be implemented with MOS gating or with a microprogrammed control unit. The primary advantages of a microprogrammed control unit are: ease of changing the instruction set by replacing the unit (e.g., replacing a diode array fixed memory wafer with another wafer), ease of design and implementation of the instruction set, and a relatively easy unit to checkout. The latter advantage may be realized since instead of complicated gating signals and combinations of signals spread throughout the processor the microprogrammed unit can be considered to be a black box with a finite fixed set of inputs giving a finite fixed set of outputs; therefore, it can be checked out by sequencing through a set of inputs that checks each memory location.

The distributed processor integrates memory and processing on a single wafer; so a primary consideration in constructing the control unit is its usage of wafer area and its affect on the yield of the processor section of the cell. A control unit constructed from MOS gating could take good advantage of redundant logic terms and could also be spread throughout the processor section of the wafer thus providing efficient gate utilization and short interconnection lines. (Note that the microprogrammed control unit would require a considerable number of long control lines.) Both of the above points would enable the gating control unit to use considerably less area than the microprogrammed unit. Therefore even though the microprogrammed control unit offers the advantages stated above a MOS gating network will be chosen for the processor section control unit in order to minimize control area. This decision can certainly be changed in the future, particularly if it is evaluated that the



5a.



Bit 7 = 0 - B₁

Bit 7 = 1 - B₂

Bit 8 - 10 = 0 0 0

0 0 1 - T₁

0 1 0 - T₂

.

.

.

1 1 1 - T₇

5b.

Figure 9-5. 16-bit Instruction Words

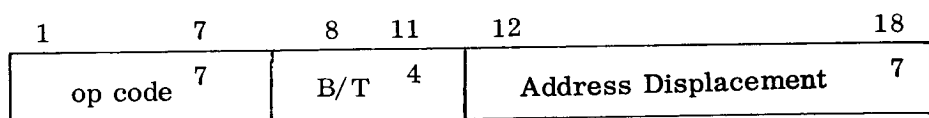


Figure 9-6. 18-bit Instruction Word

instruction set may be changed often in the distributed processor in order to apply the system to a variety of missions. (In order to change the instruction set with MOS gating the processor area of the wafer mask must be remade; however if a diode array fixed memory control unit is included in the processor a new one-zero pattern must simply be encoded into the mask.)

9.1.4 MACRO Instructions

INTRODUCTION

The DAMP system is basically an array of cells consisting primarily of storage with a small amount of each cell devoted to a processor (arithmetic and logical). An important consideration is what can be added to the processor section that can result in requiring less storage and a net reduction in total hardware required in the cell. Macro instructions (MACROS) were one such feature considered and this section will give a brief discussion of Macros in general and several specific types investigated.

A considerable amount of additional effort in this area would be necessary in order to choose a set of Macros to add to the set of common instructions (add, sub, transfer, etc.). In particular the amount of hardware necessary to implement a Macro should be traded-off against the amount of storage necessary to implement it as a subroutine using common instructions. In addition, it should be determined how much each macro would be used, since including it in the instruction repertoire requires including it in all cells. This would clearly only be worthwhile if the Macro was used often and it required a relatively small amount of hardware compared to the amount needed for common instruction storage necessary to implement the same operation. The above trade-off is biased against including most macros that may be suggested. In fact the situation is even worse when it is considered that many functions that are candidates for macros, such as sine or cosine, could be implemented as a common subroutine in a single cell devoted to receiving parameters from other cells and sending back sines, cosines, etc. as required. As a result storage for certain routines may only be required in a small number of cells.

CORDIC ALGORITHM

The Cordic Algorithm is adequately described in the literature as a useful means of generating sines, cosines, and other trigonometric or hyperbolic functions (see ref. 31 and 32). Some consideration was given to including the hardware necessary for an efficient use of the algorithm in the processor sections of the cell. The algorithm could of course be implemented by programming with a normal instruction set; however this would require more instructions than the typical series solution implemented. (The series solution for sine and cosine simultaneously requires on the order of 30 instructions depending on the machine). As a result consideration was given to making the necessary additions to the general purpose hardware in the cell. (The cordic hardware does not provide sufficient flexibility to replace the GP hardware; however it may enable some instructions to be deleted from the normal instruction set.)

The hardware used to implement the algorithm typically involves three registers capable of being shifted, two adders connected to two of the registers so that cross addition of two of the three registers can occur, a third adder for the third register, gating hardware to enable a variable pick off from two of the three registers (this enables 2^{-j} , $j = 0, 1, 2 \dots n$, times the contents of a register to be picked up for cross addition), and control circuitry. In addition to the above, for an n bit word, n angle constants must be stored either directly in fixed hardware or in the memory. The above registers can be simply made available from the normal processor registers (accumulators); however it would be necessary to add additional connections, adders, gating, control circuitry, and possibly the n constants. (If the constants are not in

fixed hardware they must be accessed from memory via stored instructions or by control circuitry.) In any case, implementation of the cordic algorithm, including the stored constants, would require a few thousand FET's in addition to that required for the general purpose hardware. Even with this hardware the algorithm still requires a number of instructions at least for initialization and storing the result. The hardware implementation of the algorithm would offer increased computation speeds for trigonometric functions, but this is not what is needed in the distributed processor system.

An accurate comparison can not be made of the above hardware that would be required in every cell with the number of instruction locations in the whole machine that would be required to execute the same functions. However, the applications for which the cordic algorithm would be useful, navigation problems involving sines, cosines, coordinate transformations etc., represent only a small percentage of the requirements for the space missions under consideration. As a result trigonometric routines would only be required in a small percentage of the cells. In fact use of separate cells for subroutine storage and execution as mentioned in the introduction would reduce to an even smaller number the percentage of cells storing trigonometric routines. From the above discussion it can then be realized that increasing the complexity of each processor by the addition of cordic hardware would bring a small return in additional available memory. As a result it is considered to be not worthwhile to implement this algorithm. (A relatively large increase in processor complexity of course makes the processor more difficult to fabricate and can result in lowered wafer yields, as a result complex macros should not be included unless they save a good amount of storage.)

DDA

The cell could be made into a DDA-GP structure. The DDA portion of the machine could be used for generation of trigonometric and hyperbolic functions as with the cordic hardware; however this DDA-GP organization would not be worthwhile for the same reasons as for the cordic algorithm. Quite a large number of FET's would be needed since at least two complete DDA's would be required including programming flexibility for the interconnections. In addition, the DDA implementation would require a number of memory words for initialization and storing the result.

GENERAL MACRO SET

Macros are being considered for the DAMP System primarily from the standpoint of saving storage. As a result, the present investigation of macros is pointed toward those that would replace a number of common instructions (decrease the number of bits associated with instructions so that less instructions can be used) and that would be used in a reasonably large number of computations. Actually, limited usage of macros would be acceptable if they did not require very much additional hardware in the processor.

The following paragraphs discuss a few basic types of macros in order to point out the type of macro that is the most fruitful to investigate for storage savings. The first type presented are basic instructions (add, etc.) that operate on non-ordered lists of data (i.e., each data word must be individually addressed). These macros save some bits, but it is felt they will not be used very often in the applications of interest here. A second type of macro would again carry out basic instructions but on

ordered lists of data. These macros are shown to be of relatively small value because they only replace loops that generally contain a very small number of instructions. The third class of macros are characterized by complex instructions on any type of data. These macros are shown to offer the best possibilities for storage savings; as a result, investigations of macros should emphasize this latter type.

It should be emphasized that when trying to save memory, the number of bits used by a macro instruction is of importance. There are a number of types of macros that can be investigated for memory bit savings. One example is multi-operand macros that individually address a number of operands (operation on non-ordered data that must be addressed from random locations) and carry out a basic logical or arithmetic operation on all of them. These macros require sufficient bits to address each operand; as a result the only memory savings is due to the saving of the op code bits that would be necessary to individually access and combine the operands. This saving could be large if enough operands could be combined at once. However, for macros that would be used sufficiently it seems that only a very few operands could be combined at once as described above and as a result the bit savings would be typically small.

Many operations with a single basic arithmetic or logical operation are typically carried out with data that is ordered into some type of a list in memory so that inherent operand addressing is possible. An example of this type of addressing is the processing of a list of information with an instruction loop that uses index registers to hold and update the operand addresses. (Note that the list does not have to be simply sequential. It could use every other memory location, etc.) The inclusion of a repeat mode in a processor enables loops, as described above, that contain a single instruction to be executed very quickly since the count of the times through the loop and the termination of the loop are automatically handled. However the use of a repeat mode or of a macro to initialize the appropriate index registers and execute an operation on a list of ordered data can save very little storage since the loops used to carry out the same operation require only a few initialization words for the appropriate index registers plus the basic loop (one operation plus index handling instructions). As a result a macro to carry out a basic operation on an ordered list would save very few bits (only the op code bits for the index initialization and handling instructions). The inclusion of such macros or even a repeat mode is then not worthwhile in the DAMP System unless an increase in the speed of execution is needed. Another type of macro that could use basic arithmetic or logical instructions and forms of inherent operand addressing to save storage bits involves using a push down stack in the processor. Data could be processed and placed in the stack, and when appropriate a macro could be issued that executes a basic arithmetic or logical operation on the top members of the stack. The number of words to be combined would be the only parameter required in addition to the op code (no addresses). This macro would then save the loop initializations and index handling that would be necessary if the same instruction was executed on the list of data by a loop. The stack may also find other uses for the programs, however, it is not clear at this time that such a stack would find any real usefulness.

The class of macros characterized by a single instruction that replaces a quantity of basic instructions offers a good opportunity for memory bit savings; however, macros of this type that will be used fairly often and do not take an unreasonably large amount of hardware are difficult to find. Useful macros of this type could operate on very few operands or could use some form of inherent operand addressing

to operate on lists of operands. In fact use of the cordic algorithm to generate trigonometric functions could be considered an example of such a complex macro. This macro could call out sine, for example, and then give the angle to a hardware unit set up to return the answer; however, this macro was shown to be impractical from a hardware standpoint since it provided high speed sine generation but saved very little storage and was not used very often. Exactly the same arguments eliminate consideration of special function generators using diode arrays, for example. One possible useful instruction of the complex macro type is the vector dot product. It would address the first element in each vector and place the addresses in index registers. The instruction would also specify the number of elements in the vector and place this value in a third index register. The elements of the vectors would be stored in the memory in sequence following the first element. The instruction would then address all elements by simply incrementing the index registers, multiplying corresponding components, and adding to the previous sum (stored in the second upper accumulator and the lower accumulator) until the third index register reaches zero and the operation is terminated. This macro would then save the bits necessary to specify the initializations and the sequence of operations within the loop that could be used for the calculation. The main usefulness of this instruction would be in executing matrix multiplies, however it may also find use as a substitute for a sum of products multiply. The difference in this latter case is that two memory locations would be multiplied instead of one of the upper accumulators and a memory location. This is acceptable except that more bits are required than for specification of a simple sum of products multiply; therefore the vector dot product instruction would be an improvement in this case only if the use of the sum of products instruction generally requires the accumulator to be loaded first. (This is probably not generally the case.) In order to get a good evaluation of the value of this macro its use in a matrix multiply needs to be compared to a loop implementation of a matrix multiply. The amount of usage of the instruction should also be evaluated. Other macros of the above complex type need to be investigated. Two possibilities are a full matrix multiply and complex operations on a stack.

9.1.5 Byte Manipulation Instructions

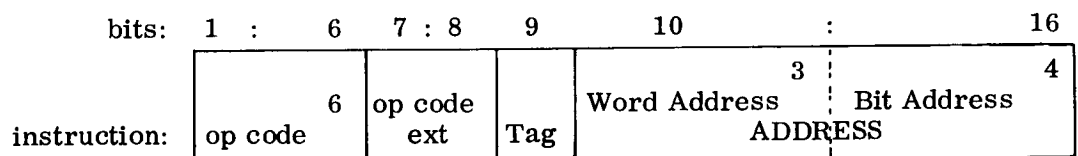
Byte manipulation instructions were investigated and several included in the instruction set that will be presented in the next section. The byte manipulation instructions were included to enable the DAMP system to flexibly manage variable length bytes for requirements such as the scientific experiments. These same instructions can be used for character instructions.

The instruction word format for byte instructions is shown in Figure 9-7. In order to set up this word a few assumptions were made about the operation on byte data. It was assumed that the bytes to be operated on are generally located in a list such that they could readily be addressed relative to the first word containing the first byte in the list. It was also assumed that one particular byte length was operated on for a number of instructions such that it would be efficient to use a separate instruction to setup the byte length in a separate register (This separate register will be referred to as the LENR register). Of course if the length was changed very often the overhead of setting up the LENR register by separate instruction would be impractically high; however if one length is used for a number of instructions, as assumed, the scheme specified above will be efficient since the length bits will not have to be included in each instruction. (The length bits are used for compare

instructions, for automatic right adjustment of bytes, for storing in memory without affecting other bits in word, etc.)

An explanation of Figure 9-7 follows. The op code bits will give a basic byte instruction by using the op code extension bits in addition to bits one to six. The address will be relative to index/bank register B_1 for all byte instructions. This register will be setup with some position in the list of bytes and it will be assumed that bit position one is the starting bit position of the first byte in this word. The address portion of the instruction word will then simply give the starting bit number in the list of bytes of the desired byte. In order to understand this addressing note that the address portion of the instruction word is divided up into a word address and a bit address. The bit address gives the starting bit position of the desired byte in its memory word. These bits are placed into a register for instruction control. The remaining address bits, the word address, are placed right adjusted into a memory address register and added to B_1 in order to obtain the memory word location to be accessed for a particular instruction. If the instruction is indexed, index/bank register T_1 and the full seven bit address given in the instruction word are added together so that the indexing can be relative to bits. This addition is carried out by first loading the full address into the accumulator then adding T_1 , and then placing the least significant four bits in a register and the remaining bits right adjusted into the memory address register. The memory address register will then of course be added to B_1 in order to get the proper word address.

The instruction word in Figure 9-7 allows for eight word lists without having to change B_1 or T_1 (the word address is three bits). If it is found that the predominate use of byte instructions is in loops it may be worthwhile to change the instruction word to allow for more indexing. For example the tag could be increased to two bits and the word address decreased to two bits. A number of byte instructions will be given in the instruction list presented in Section 9.2.



Tag: "0" - no indexing

"1" - index with T_1

Figure 9-7. Byte Instruction Word

9.2 FUNCTIONAL DESCRIPTION OF THE CELL

This section contains a functional description of the cell hardware. A block diagram of the cell will be presented first and then several areas of the cell's operation will be explained in more detail.

9.2.1 Cell Block Diagram

Figure 9-8 contains a general block diagram of the cell. As can be seen the cell contains hardware that can be broken down into six general areas:

1. Memory
2. Identification
3. Arithmetic and Control
4. Inter-Cell Bus Communications
5. I/O
6. Timing

A detailed block diagram of the cell is shown in Figure 9-9. This figure contains all the main registers in a cell. There is of course a lot of logical gating not shown. However, the drawing will serve to present some insight into the detailed operation of a cell. The registers shown also have a numeral in the right hand corner. This indicates the length (bits) of the register and gives some indication as to its complexity. Each of the blocks in the figure will be explained below:

U	Hardware Upper Accumulator: The hardware upper accumulator is used to hold one of the memory upper accumulators in any operation that uses them.
L	Lower Accumulator: The lower accumulator is used primarily in multiply, divide, and double precision operations to hold the lower half of a data word. This accumulator, and U have a one-bit extension onto their sixteen bits in order to hold the overflow carries which may be generated in the multiply operation.
U ₁ , U ₂ , U ₃ , U ₄	Memory Upper Accumulators: The memory upper accumulators are the primary arithmetic and logical registers. They are also used 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. It must therefore be connected both to the ALTU and to the memory interface lines.

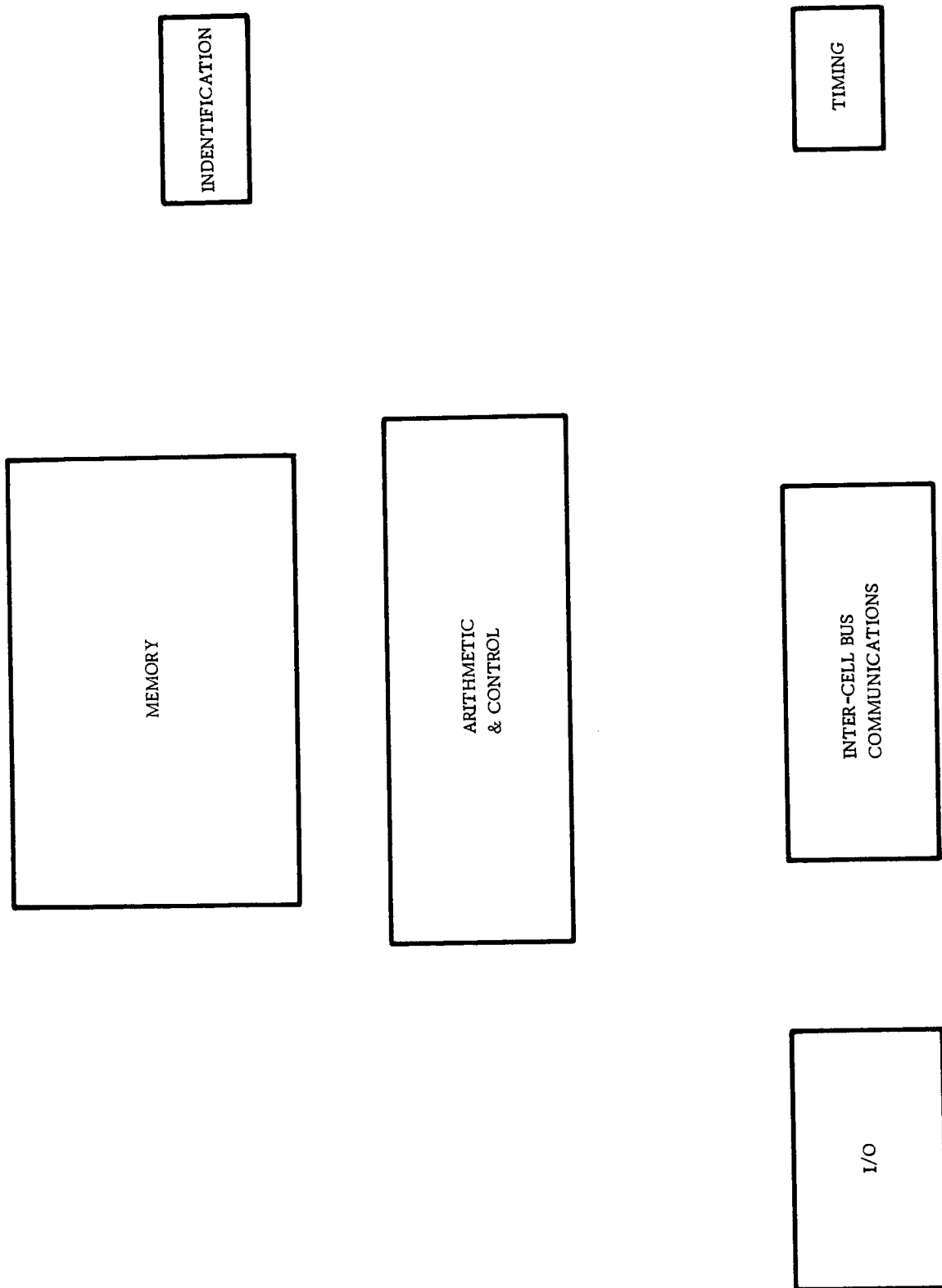


Figure 9-8. General Cell Block Diagram

MAR	Memory Address Register: This register holds the memory address for operand memory cycles. It is loaded with the address displacement from the instruction word, B ₁ or B ₂ is added to it and, if indicated, one of the index (T) registers is also added to it. This register is necessary since the B and T registers are in memory and must therefore enter the processor through MB; as a result MB cannot be used to hold the operand addresses.
MB	Memory Buffer: The memory buffer receives data and instructions from the memory, sends data to the memory, 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. 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 and one of the registers to be operated on in register operations.
B ₁ , B ₂	"B" Memory Index/Bank Registers: These registers hold both index and bank values for address calculation and looping control. One of these two registers, indicated by the B bit, is added to the address displacement for all operand address calculations.
T ₁ to T ₃	"T" Memory 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 register to B plus the address displacement. (Tag 00 specified no indexing with the T 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 hardware registers 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 T bits of the instructions. It is necessary so that B plus the address displacement can be generated, stored in MAR, and then added to T prior to an operand cycle. This register also holds one of the register addresses in register operations. It also holds a two bit op code extension for byte operations.

SCR	Shift Count Register: This register holds the shift count for shift commands, and for setting up bits in byte manipulation operations. 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. It also provides a temporary storage area for certain control information during several inter-cell bus communications operations.
LENR	Length Register: This register is used for byte instructions to specify the length of the byte that is being used. It is also used to hold the op code extension bits in shift and register instructions.
HA	Hardware Accumulator: This register identifies which of the 4 accumulators is presently in hardware. Every instruction that specifies an accumulator must compare these 2 bits to that in the instruction. If the accumulator specified in the instruction is in memory, the hardware accumulator must be stored in memory and the other accumulator brought into the hardware position (the next Section 9.2.2 will discuss this in more detail).
CFF	Comparison Flip-Flops: These flip-flops hold the results of a comparison; greater than, less than, or equal. They may be tested by an instruction for a conditional jump, skip, etc.
O	Overflow Flip-Flop: This flip-flop is set if an overflow results in the ALTU during an arithmetic operation. Like the CFF, it may be tested by an instruction.
F	Failure: This flip-flop is set by the hardware failure detection circuitry. It can be sampled by an instruction during software self test.
IMR	Interrupt Mask Register: This register is set by the programmer to mask off any interrupts that are not to be allowed.
CCM	Controller Cell Mode: This register holds the mode of a cell in the controller cell state. It has no meaning in cells in any other state. It will be set to one of the four transmit/execute modes and will control the execution of instructions in the controller cell.
Cell ID	This register holds the cell address or identification. It is used in the Bus I/O control section to decode commands over the Bus.

State	This register holds the current state of a cell. It will be in one of 5 states: <pre> 000 INDEPENDENT 001 DEPENDENT GLOBAL 010 DEPENDENT LOCAL 011 DEPENDENT WAIT 100 CONTROLLER CELL </pre>
Level	This register specifies one of eight levels that the cell may use as a means of identification during dependent state operation.
BCR	Bus Communication Register: This register receives the present byte on the inter-cell bus. It is used by the Bus I/O Control to decode commands and control the bus communications.
Bus I/O	This register holds the identification of the present operation being carried out on the inter-cell bus (input, output, etc).
IC Busy	INTER-CELL BUS BUSY: This flip-flop is used to determine if the bus is presently being used.
Timer 1, 2	These two timers are used to time certain operations over the inter-cell bus. Expiration of the timers during these operations results in an interrupt being issued.
ION R	I/O and Neighbor Register: This is a serial/parallel in and out buffer register used for communications between I/O devices and one of the four neighbor cells. It is 18 bits long due to the need for sync and control bits in addition to the data bits (16 bits).
Timer 3	This timer is set during certain I/O operations such as requesting data from a neighboring cell. Its expiration will cause an interrupt.
Counter	This is a counter to count the serial in/out of the ION R. It will issue interrupts during I/O operations.
I/O	This flip-flop is set to either the input or output state during an I/O instruction execution. It controls the type of interrupt generated by the COUNTER mentioned directly above.
FU1, 2, 3, 4	ACCUMULATOR FLAGS: These flags are set to one of five states: N, E, S, W, or X. The first four indicate the accumulator holds data for a neighboring cell and the last one indicates no neighbor data. There is one flag register for each accumulator.

RTC EXT	Real Time Clock Extension: This is a simple counter driven by the clock.
RTC	Real Time Clock: This register is a simple counter driven by the RTC EXT. It is set and read by instructions and counts down to zero.
BTC	Bit Time Counter: This counter is driven by the clock and generates two pulses (counts to two). It is used to control the instruction execution and drive the instruction decoding and control generation logic.
MC	Mode Counter: This counter is driven by the BTC and generates eight control signals. It drives the instruction decoding and control generation logic.

It should be mentioned here that the length of the registers indicated in Figure 9-9 does not include any bits for fault detection. At least one bit for parity will be added on to several registers such as U, L, IONR, and BCR. Other registers such as HA, CFF, O, etc. may be combined for generating parity and, therefore, will not necessarily have an extra bit for each of these registers.

9.2.2 Accumulator Mechanization

A somewhat more detailed discussion of the accumulators is necessary in order to understand their use. There are a number of methods of handling the four accumulators in each cell. One method would assign tag bit combinations to each accumulator such that U_1 is the hardware accumulator and U_2 , U_3 and U_4 are the memory accumulators. Each instruction would then specify one of these accumulators and cause its contents to be exchanged with those of the hardware accumulator, U_1 , for execution of the operation. At the conclusion of the operation the hardware accumulator would keep its current value so that any further operations of this value would be carried out by specifying U_1 . This scheme uses the minimum amount of processor hardware for handling the accumulators, but it makes it difficult for the programmer to keep track of information that was initially or subsequently stored in one of the accumulators. It would also be necessary at the ends of a branch to restore the accumulators to some specified ordering of information that is consistent between the branches. Because of the above two disadvantages, this scheme was not chosen.

A second scheme would replace the named accumulator into its original location at the completion of each operation. This means that unless the hardware accumulator, U_1 , was in operation, a final exchange of the present value of U_1 and the original value of U_1 in one of the memory accumulators would have to be made. The main disadvantage of this approach is that it requires two extra memory cycles for any accumulator operation that does not use U_1 . This extra time could be considerable since when an accumulator is brought into execution it is generally used for a few instructions; as a result each operation would require two extra memory cycles. For this reason this approach was not chosen.

A third approach as follows is also possible. Four sets of tag bits are held in the processor to specify the tag associated with the hardware or memory accumulator positions. (In this scheme U_1 , U_2 , U_3 , and U_4 can be in any of three memory positions or the hardware position.) Each instruction operating with an accumulator simply specifies the tag bits of the accumulator that it would like to use. This accumulator is then found by an automatic comparison to the accumulator tag bits, HA1, MA2, MA3, and MA4. The specified accumulator is then loaded into the processor accumulator position (if it is not already there) and the accumulator tag bits are updated to reflect the present locations of U_1 , U_2 , U_3 , and U_4 . (When the machine is started, these tag bits must be loaded into HA1, MA2, MA3, and MA4 in any order.) This scheme requires slightly more processor hardware than the other schemes mentioned, but it has the advantage of leaving the last accumulator referenced in the hardware accumulator position. Therefore only the first reference to a new accumulator requires an extra memory cycle (the exchange of accumulators can be carried out in one memory cycle). It should be noted that the accumulator tag bits in the processor must be stored after an interrupt in order to enable proper restarting of an interrupted program.

A fourth approach similar to the last scheme is described below. Four locations in memory, as shown in Figure 9-9, are used to hold U_1 to U_4 . Whatever accumulator is referenced by the instruction word tag bits, is placed in the hardware accumulator position and the present contents of the hardware accumulator are returned to their proper memory position. Two control bits HA1 are necessary so that the accumulator presently in hardware can be specified and compared to the tag bits in the instruction. Clearly, if the present hardware accumulator is specified no memory access is required. This scheme then accomplishes the same operation as the last scheme (it leaves the last referenced accumulator in the hardware position) but uses slightly less processor hardware for control and one more memory location. It also requires only one additional memory cycle when an accumulator from memory is specified since the present accumulator value should be able to be replaced in its proper memory position and the new accumulator picked up all in one memory cycle. At the same time the HA bits will be updated to the new accumulator tag. This scheme was selected over the third approach described above because it should actually require less total hardware usage. It requires less control and register hardware in the processing section due to requiring only the HA tag bits. More important though is that less bits will have to be stored upon an interrupt which may actually make up for the extra memory location used for the accumulator.

9.2.3 Instruction Set

This section gives the instruction set selected for the distributed processor. Since 6 bits are used for the op code in the instruction word, there are a total of 64 op codes that may be used for the instructions. However, this does not limit the set to 64 instructions since many op codes will use op code extension thereby mechanizing many instructions from one basic op code.

There are four accumulators used and referencing each accumulator for operations such as add, subtract, multiply, etc requires one op code for each reference. Therefore, four op codes are required to provide an add instruction for each accumulator. There are not enough op codes to provide full accumulator specification for

every instruction that specifies an accumulator. Therefore, the approach taken was to provide full accumulator capability for the more important instructions only such as add, subtract. Other instructions will be limited to less accumulators or to the hardware accumulator only.

The operation times listed below with the instruction set are given in terms of memory cycles. This leaves open the specification of the memory cycle time and the processor bit time. However, if one wishes to assign values for preliminary performance estimates, a value of 1 μ second for memory cycle time and 0.5 μ sec for processor bit time may be used.

The following is a list of abbreviations used in the specification of the instruction set:

m:	Address Displacement
M:	Address After Banking and Indexing (if specified)
(M):	Contents of Addressed Memory Position
$\rightarrow$:	Replaces
U:	Any of the Upper Accumulators
U_1, U_2, U_3, U_4 :	Upper Accumulators
U_{123} :	Upper Accumulator U_1, U_2 or U_3
U_{12} :	Upper Accumulator U_1 or U_2
$U_{12,34}$:	Upper Accumulators U_1 and U_2 or U_3 and U_4
U_h :	Hardware Upper Accumulator
U_m :	Any of the Memory Upper Accumulators
L:	Lower Accumulator
T_n :	Any of Bank/Index Registers Specified by the T_n Bits
B:	Bank/Index Register One or Two
R:	Any of the Bank/Index Registers or Accumulators
MB:	Memory Buffer Register
MAR:	Memory Address Register
RTC:	Real Time Clock

P:	Program Counter
LU^n :	n Position Left Shift of U
RU^n :	n Position Right Shift of U
LU_c^n :	n Position Left Cycle of U
RU_c^n :	n Position Right Cycle of U

Before giving the instruction list an example of the execution of an add instruction will be given to clarify the use of memory cycles.

INSTRUCTION: ADD

$$(M) + U \rightarrow U$$

This is executed as follows with no indexing and with U located in the hardware location:

Instruction access: $m \rightarrow \text{MAR}, (P) \rightarrow \text{MB}$	1 memory cycle
Bank access: $B + \text{MAR} \rightarrow \text{MAR}$	1 memory cycle
Operand access and execution: $(M) + U \rightarrow U$	<u>1 memory cycle</u>
	3 memory cycles

It is executed as follows with indexing and with U located in one of the memory accumulator positions:

Instruction access: $m \rightarrow \text{MAR}, (P) \rightarrow \text{MB}$	1 memory cycle
Bank access: $B + \text{MAR} \rightarrow \text{MAR}$	1 memory cycle
Index Access: $T_n + \text{MAR} \rightarrow \text{MAR}$	1 memory cycle
Accumulator access: $U_h \rightarrow U_m$ (old accumulator put in its location)	1 memory cycle
$U_m \rightarrow U_h$ (new accumulator picked up)	
Operand access and execution: $(M) + U \rightarrow U$	<u>1 memory cycle</u>
	5 memory cycles

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
1-4	ADD 3 (4) mc (memory cycles) ⁽¹⁾	$(M) + U \rightarrow U$	Since the memory contents are loaded into the memory buffer register, the actual add is between the MB and the accumulator. Note that any of the 4 accumulators can be used.
5-8	SUBTRACT 3 (4) mc	$(M) - U \rightarrow U$	
9-12	MULTIPLY ~7 (8) mc	$(M) \times U \rightarrow U, L$	A one bit at a time multiply is used since speed is not critical.
13-15	DIVIDE ~7 (8) mc	$U_{123}, L \div (M) \rightarrow U_{123}(\text{quotient})$ $L(\text{remainder})$	This time is for a straight one bit at a time divide. Note that this instruction is limited to 3 accumulators. It also uses a U, L dividend so that double precision divide software is possible to implement. For single precision divides the L register can be zeroed, if desired.
16-17	AND 3 (4) mc	$(M) \cdot U_{12} \rightarrow U_{12}$	Note that only two of the accumulators can be used for the logical and.

⁽¹⁾One additional memory cycle should be added to this and any other instructions that are to be indexed (specified by a T bit). The execution time in parenthesis represents the execution time if the accumulator is in memory and not in the hardware location.

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
18-19	OR 3 (4) mc	$(M) + U_{12} \rightarrow U_{12}$	
20	EXCLUSIVE OR 3 (4) mc	$(M) \oplus U_h \rightarrow U_h$	This instruction is limited to the contents of the hardware accumulator. Note that this type of limitation enables any accumulator to be used. It is simply necessary to load the hardware accumulator in the desired accumulator by an earlier instruction.
21	SUM OF PRODUCTS MULTIPLY ~ 7 mc	$(M) \times U_h + U_1, L \rightarrow U_h, L$	Note that this is a full length sum of products multiply. This is not possible with only one accumulator.
22	SHIFT SINGLE PRECISION		

NOTE: The shift instructions, op code 22 and 23, use an op code extension format shown below:

bits: 1 --- 6 7--8 9 10--11 12 --- 16

OP CODE	ACC	T	OP EXT	COUNT

where ACC: One of the 4 accumulators, U

T: A zero specifies no indexing, a one specifies indexing the shift with index register T_1

OP EXT: Two bits to provide 4 instructions, they are held in the LENR register.

COUNT: The amount of the shift up to 32 positions (32 are needed to handle double precision instructions given in op code 23).

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
22a	Short Right Shift 1 (2) mc + n -1 bit times ⁽¹⁾	$RU^n \longrightarrow U$	It should be noted that on all noncyclic right shifts the sign is spread whereas noncyclic left shifts insert zeros.
22b	Short Left Shift	$LU^n \longrightarrow U$	
22c	Short Logical Right Shift	$RU^n \longrightarrow U$	'0' 's are inserted at the left.
22d	Short Logical Right Cycle	$RU_c^n \longrightarrow U$	
23	SHIFT DOUBLE PRECISION		
23a	Long Right Shift	$RU, L^n \longrightarrow U, L$	One of the four accumulators and the lower accumulator are shifted n positions.
23b	Long Left Shift	$LU, L^n \longrightarrow U, L$	
23c	Long Logical Right Shift	$RU, L^n \longrightarrow U, L$	With '0' 's inserted on the left.
23d	Long Logical Right Cycle	$RU, L_c^n \longrightarrow U, L$	
24-25	DOUBLE PRECISION ADD 5 (6) mc	$(M) + U_{12,34} \longrightarrow U_{12,34}$	Two sequential memory locations are added to accumulators 1 and 2 or 3 and 4
26-27	DOUBLE PRECISION SUBTRACT 5 (6) mc	$(M) - U_{12,34} \longrightarrow U_{12,34}$	
28-31	LOAD ACCUMULATOR 3 (4) mc	$(M) \longrightarrow U$	

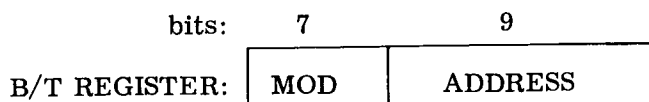
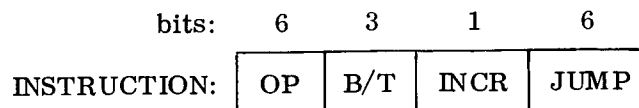
⁽¹⁾ Same execution time for all shift instructions given by OP Codes 22 and 23

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
32-35	STORE ACCUMULATOR 3 (4) mc	$U \longrightarrow (M)$	
36-37	DOUBLE PRECISION LOAD ACCUMULATORS 5 (6) mc	$(M) \longrightarrow U_{12, 34}$	Two sequential memory locations are loaded into accumulators 1 and 2 or 3 and 4.
38-39	DOUBLE PRECISION STORE ACCUMULATORS 5 (6) mc	$U_{12, 34} \longrightarrow (M)$	
40-43	COMPARE 3 (4) mc	$U > (M), \text{CFF} = G$ $U = (M), \text{CFF} = E$ $U < (M), \text{CFF} = L$	This instruction compares any-one of the accumulators to memory; it results in the comparison flip-flops (CFF) being set to one of three states.
44	INCREMENT AND REPLACE 3 mc	$(M) + U_h \longrightarrow M$	
45	EXECUTE	$(M) \longrightarrow \text{MAR}$	This instruction sets up the address of the next instruction in the memory address register. This register is then used to pick up the next instruction. 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. However, it should be noted that it is not considered completed until the next instruction is completed. (It is not interruptable until this time.)
46	LOAD ADDRESS 3 mc	$m + B + T \longrightarrow U_h$	

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
47	JUMP 2 mc	$M \longrightarrow P$	
48	JUMP AND SET INDEX 3 mc	$P \longrightarrow T_n$ $(m+B) \longrightarrow P$	This instruction is for sub-routine linkage. Note that the loading of P is indirected. The instruction must be preceded by a load bank, if the subroutine starting location is not contained in the program's working storage. Note that the command also stores the return address in an index-bank register. A JUMP instruction indexed by the same index-bank register will return back to the original program exit plus the increment (m). This instruction is also convenient if it is desired to have the calling sequence (subroutine parameter values) in the program bank after the instruction. (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.
49	JUMP ON COMPARE FF 2 mc	If any condition TRUE then: $B + m \longrightarrow P$	The T bits of the instruction hold the conditions (greater than, less than, Equal, or Overflow) that are to be compared to the CFF.
50	LOAD B/T INDEX/ BANK REGISTERS		Two instructions are derived from this Op Code.
50a	LOAD B REGISTERS 4 mc	$(M) \longrightarrow B$	This instruction is executed if the T bits = 0. M is given by B+m, where B is either B ₁ or B ₂ .
50b	LOAD T REGISTERS 4 mc	$(M) \longrightarrow T$	This instruction is executed if T $\neq$ 0. M is given by B+m and T is specified by the T bits.

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
51	STORE B/T INDEX/BANK REGISTERS		Two instructions are derived from this Op Code.
51a	STORE B REGISTERS 3 mc	$B \longrightarrow (M)$	This instruction is executed if $T = 0$. M is given by $B+m$.
51b	STORE T REGISTERS 4 mc	$T \longrightarrow (M)$	This instruction is executed when $T \neq 0$. M is given by $B+m$.
52	LOAD/ INCREMENT B/T REGISTERS		Three instructions are derived from this Op Code.
52a	LOAD T IMMEDIATE 2 mc	$m \longrightarrow T$	This instruction is executed if $T \neq 0$. Also $B = 0$, otherwise 52b will be executed.
52b	INCREMENT T IMMEDIATE 3 mc	$m + T \longrightarrow T$	This instruction is executed if $T \neq 0$ and $B = 1$.
52c	INCREMENT B IMMEDIATE 3 mc	$m + B \longrightarrow B$	This instruction is executed if $T = 0$.
53			This Op Code is used to derive two instructions.
53a	LOAD B IMMEDIATE 2 mc	$m \longrightarrow B$	This instruction is executed if bit 8 (one of the T bits) is 0. This leaves 8 bits for m.
53b	LOAD STATUS 3 mc	$(M) \longrightarrow L, P, HA, CFF, O, CCM$	This instruction is executed if Bit 8 = 1. A direct address given by bit 7 and bit 9 through 16 (9 bits - full address) is used to form M and access the first of two sequential locations that contain the initializing information for the L, P, HA, CFF, O, CCM registers. These two locations will be the place an interrupting program stores this information that was automatically stored upon an interrupt.

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
54	INCREMENT B/T TEST AND JUMP 2 mc		This instruction is explained by showing the contents of the instruction and B/T register.



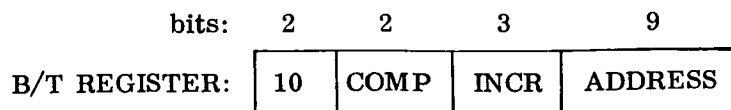
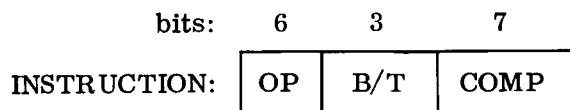
It should be recalled that the B/T registers are in memory (16 bits and only 9 bits are required for the full address. Therefore, the excess 7 bits are used in mechanizing this instruction.

This instruction fetches the designated B or T register (B if T = 0) as indicated by the B/T bits in the instruction. It increments it (± 1) as indicated by the increment bit in the instruction (both the Mod and the ADDRESS in the B/T register are incremented, but the MOD is always counted down by 1). The amount of increments is specified by MOD in the B/T register (up to 128). If the MOD equals 0 then the JUMP takes place. The JUMP (± 32) is with respect to the P register.

Note that if a T register is being used, the B bit in the instruction can extend the increment value to ± 2 .

55		This Op Code is used to derive 4 B/T register instructions.

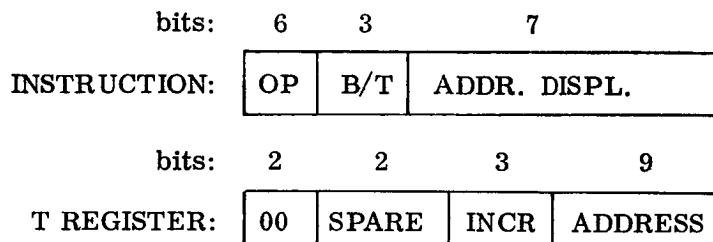
55a INCREMENT B/T
COMPARE
IMMEDIATE AND
SKIP
2 mc



The B/T bits in the instruction specify which B/T register is to be fetched. The first two bits in the B/T register are then decoded to derive which one of the four instructions given by OP Code 55 are to be performed. The B/T register (the address) is incremented ($\pm 1, 2, 4, 8$) as specified by the INCR bits in the B/T register. The COMP bits in the instruction and the B/T register are combined to get a full 9 bit comparison value that is compared to the address in the B/T register. If the comparison is not equal the program counter is incremented by 2 rather than 1 if they compare.

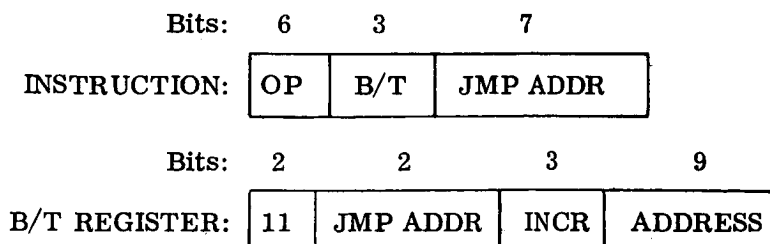
<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>

55b	INCREMENT T, COMPARE TO A VARIABLE AND SKIP 4 mc		



The B/T bits are used as follows. The T bits cannot = 0 in this instruction, they specify which T register is to be fetched. The first two bits in the T register identify this particular instruction (55b). Then the B bit in the instruction is used to fetch the variable (located by B + m) using the address displacement field in the instruction as done in conventional instructions. The address in the T register is incremented ($\pm 1, 2, 4, 8$) as specified by the INCR bits and compared to the variable. If they do not compare the program counter is incremented by 2 rather than 1 if they compare.

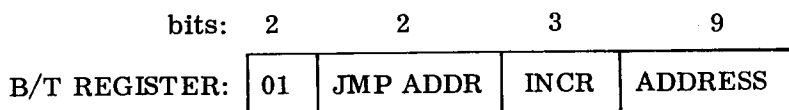
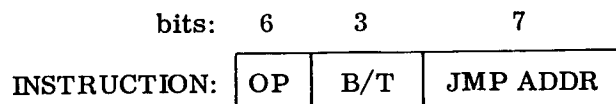
55c	INCREMENT B/T AND JUMP ON 0 2 mc		



The B/T bits specify which register to fetch. The first two bits identify this Op Code (55c). The B/T register (address) is incremented ($\pm 1, 2, 4, 8$) as specified by the INCR in the B/T register. The address is then tested against 0; if it is 0, then a full address is generated from the 7 bits in the instruction and the 2 bits in the B/T register to be used as a jump (replaces the program counter). If it is not 0 then the program counter is simply incremented by 1.

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>

55d	INCREMENT B/T AND JUMP ON + 2 mc		



This is identical to 55c except the jump takes place if the address is +.

56	LOAD BYTE		

This op code is used to form 4 different types of instructions for loading bytes. See Section 9.1.5 for a discussion of the byte instructions and the byte instruction word format.

56a	CLEAR U AND PLACE BYTE IN HIGH ORDER END	$(M) \rightarrow U_h$ $U_{hBL} \rightarrow U_h^{(1)}$ 3 mc + ($N_R + N_L + 1$) bit times ⁽²⁾

(1) This transfer and all others involving U_{hB} contain at least the following operations. (U_{hB} represents the desired byte in its unshifted position in the hardware accumulator with all other bits zero. U_{hBL} (U_{hBR}) is the same accumulator only left adjusted (right adjusted).

1. The bits including the leftmost of U_{hB} are right shifted into L
2. The accumulator is zeroed.
3. Only the byte is shifted back into U. (The number of bits to be shifted is controlled by length in LENR.) The byte will be right adjusted at this point.
4. The byte will now be shifted to its original position or right or left adjusted.

(2) This same execution time applies to all the memory addressing byte instructions. N_R represents the number of bit positions shifted right into L. N_L represents the number of bit positions shifted left back into U_h . The one is added for the zeroing of U_h .

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
56b	CLEAR U AND PLACE BYTE IN LOW ORDER END	$(M) \longrightarrow U_h$ $U_{hBR} \longrightarrow U_h$	
56c	CLEAR U AND PLACE BYTE IN UNSHIFTED	$(M) \longrightarrow U_h$ $U_{hB} \longrightarrow U_h$	
56d	PLACE BYTE IN UNSHIFTED	$U_{hS} \longrightarrow U_1^{(3)}$ $(M) \longrightarrow U_h$ $U_{hB} \longrightarrow U_h$ $U_h \vee U_1 \longrightarrow U_h$ (or)	Remaining Bits are left in accumulator.
57	STORE BYTE		4 store byte instructions are derived from this Op Code.
57a	STORE BYTE IN HIGH ORDER END	$U_{hBH} \longrightarrow U_1$ $(M) \longrightarrow U_h$ $U_{hSH} \longrightarrow U_h$ $U_h \vee U_1 \longrightarrow U_h$ $U_h \longrightarrow (M)$	The rest of the memory location remains the same. U_{hSH} is the same as U_{hS} except the byte is left in the higher order end of the accumulator.

⁽³⁾ This transfer is executed as follows:

1. The bits including the leftmost of the desired byte are right shifted into L.
2. L is then left shifted the byte length, but zeros are inserted into the lower end of U. The remaining bits are shifted back into U. (This has simply zeroed the desired byte position in U_h).
3. U_h then replaces U_1 .

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
57b	STORE BYTE IN LOW ORDER END	$U_{hBL} \longrightarrow U_1$ $(M) \longrightarrow U_h$ $U_{hSL} \longrightarrow U_h$ $U_h \vee U_1 \longrightarrow U_h$ $U_h \longrightarrow (M)$	
57c	STORE BYTE UNSHIFTED	$U_{hB} \longrightarrow U_1$ $(M) \longrightarrow U_h$ $U_{hS} \longrightarrow U_h$ $U_h \vee U_1 \longrightarrow U_h$	$U_h \longrightarrow (M)$
57d	LOAD LENGTH REGISTER	$M_{13:16} \longrightarrow LENR$	The four bits are used from bits 13 through 16 of the instruction.
58	COMPARE BYTE		This OP Code can be used to derive 4 compare instructions. However only 3 have been included thus far.
58a	ALGEBRAIC BYTE COMPARISON	$(M_B) > U_{hB},$ $CCF = G$ $(M_B) = U_{hB},$ $CCF = E$ $(M_B) < U_{hB},$ $CCF = L$	Note the comparison will assume that the same position in the memory word and the accumulator are to be compared. M_B represents the proper memory byte. For this comparison the first bit is considered to be a sign bit.
58b	LOGICAL BYTE COMPARISON		Same as above except no sign bit is considered.

<u>OP Code</u>	<u>Instruction</u>	<u>Operation</u>	<u>Comments</u>
58c	COMPARE BYTE ABSOLUTE	$\left (M_B) \right > \left (U_{hB}) \right , \text{CCF}=\text{G}$ $\left (M_B) \right = \left (U_{hB}) \right , \text{CCF}=\text{E}$ $\left (M_B) \right < \left (U_{hB}) \right , \text{CCF}=\text{L}$	
59	ACCUMULATOR- ACCUMULATOR AND NEIGHBOR INSTRUCTIONS		Many instructions are derived from this Op code. They are listed at the end of the instruction list.
60	REGISTER-REGISTER INSTRUCTIONS		This Op code is used to derive many instructions for operating on the registers. They are listed at the end of the instruction list.
61	SKIP INSTRUCTIONS - ACCUMULATORS		Many conditional skips are derived from this Op code. They are listed at the end of the instruction list.
62	CC INSTRUCTIONS		A variety of CC instructions are derived from this Op code and are listed at the end of the instruction list.
63	LONG INSTRUCTION FORMAT (32 bits)-1 (Input/Output)		This instruction and the one that follows use two words for their mechanization. The format of this long instruction is shown for Op Code 63. This particular long instruction is used for I/O and is explained in Section 7.2.

BITS: 6 1 9 7 9

INSTRUCTION:	OP	I/O	Address	COND/DEVICE	WORD COUNT

64	LONG INSTRUCTION -2		The instructions derived from this format are given at the end of the instruction list.

It was seen that some of the instructions used OP Code extension to derive many instructions from one OP Code. Instructions 59, 60, 61, 62, and 64 will be presented separately below.

OP CODE 59

Accumulator - Accumulator and Neighbor Instructions

CASE 1

	BITS:	6	1	5	2	2
INSTRUCTION:		OP	0	OP2	R ₂	R ₁

R₂ is always an accumulator in this cell

R₁ is either an accumulator in this cell or is one of the 4 neighbor cells. If OP2 is even, then this cell accumulator is used, if OP2 is odd, then a neighbor is used, except for some cases as shown below.

OP2 is

- 0-1 Load
- 2 Exchange
- 3 Double Precision Exchange
- 4-5 Add
- 6-7 Subtract
- 8-9 Multiply, 32 bit result
- 10-11 Divide
- 12-13 AND
- 14-15 OR
- 16-17 XOR
- 18-19 Logical Add
- 20-21 Compare, Arithmetic
- 22-23 Compare, Logical
- 24-25 Load Byte

- 26-27 Multiply, 16 Bit Result
- 28-29 Load Double Precision
- 30 Set Flag, Accumulator R_2 to neighbor R_1
- 31 Double Precision Add

From CASE 2 Format (below)

- 1-2 Double Precision Subtract
- 3 Double Precision Compare

CASE 2 Single Register (see also OP Code 60)

bits: 6 1 5 4

INSTRUCTION:	OP	1	OP2	R_2

R_2 is a register in this cell

- 4 Accumulators
- 3 Index Registers
- 2 Bank Registers
- 1 Level Register
- 2 Pairs of Double Precision Accumulators
- 1 Program Counter
- 1 State and Level Register
- 1 L Register
- 1 Real Time Clock

OP2 is

0	1's Complement	16	Skip if $R_2 = N$
1	2's Complement	17	$R_2 \neq N$
2	Absolute Value	18	$R_2 \geq N$
3	Negative Value	19	$R_2 > N$
4	Clear to Zero	20	$R_2 < N$
5	Set to all Ones	21	$R_2 \leq N$
6	Jump to Address	22	R_2 all ones
7	Store P	23	R_2 even
8	Spare	24	R_2 odd
9		25	Spare
10		26	
11		27	
12		28	
13			
14			
15			

OP CODE 60

Register - Register Instructions

	6	2	4	4
INSTRUCTION:	OP	OP2	R_2	R_1

R_2 is any Register

R_1 is any Register

- 4 Accumulators
- 3 Index Registers
- 2 Bank Registers
- 1 Level Register
- 1 Program Counter
- 1 State and Level Register
- 2 Pairs of DP Accumulators
- 1 L Register
- 1 Real Time Clock

OP2 is

- 0 Load
- 1 Exchange
- 2 Add
- 3 Compare

OP CODE 61

Skip Instructions - Accumulators

BITS: 6 3 3 4

INSTRUCTION:

OP	T	R ₁	R ₂

R₂ is one of the following

- 3 Index Registers
- 4 Accumulators
- 2 Bank Registers
- 1 Level Register

R₁ is one of the following

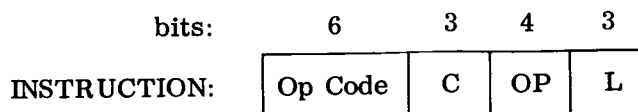
- 4 Accumulators
- 3 Index Registers

T is type of Skip. Skip if

- 0 $R_1 = R_2$
- 1 $R_1 \neq R_2$
- 2 $R_1 \geq R_2$
- 3 $R_1 < R_2$
- 4 $R_1 > R_2$
- 5 $R_1 \leq R_2$
- 6 Overflow (accumulators), I/O tests, etc.
- 7 Spare

OP CODE 62

CC INSTRUCTIONS



This instruction is used to control the modes of the controller cell and to generate the commands for the inter-cell bus operation. For certain instructions it uses an extended format. It was explained in Section 6.

C - Controller Cell Field

= 000 Do not change mode

= 001 Control bus I/O logic

= 010 }
 = 011 } Bus I/O (Generate GC Commands) (Sent over inter-cell Bus)

= 100 Transmit mode, single

= 101 Transmit mode, all

= 110 Execute mode, single

= 111 Execute mode, all

If C = 000, Do not change mode, then the following instructions may be formed

IF OP = 000

No Operation

IF OP = 001

Format Instructions:

<u>L</u>	<u>Instruction</u>
000	A
001	D16
010	D32
011	A, D16
100	A, D32
101	I
110	DS

IF OP = 010

State and level control instructions:

<u>L</u>	<u>Instruction</u>
000	Go to Controller Cell State, Set Level
001	Go to Independent State, Set Level
010	Go to Independent State, No Level Change
011	Go to Dependent Wait State, Set Level
100	Go to Dependent Wait State, No Level Change

IF C = 001, Control Bus I/O Logic

There is room for many instructions here. Thus far, two have been defined.

- a. Prepare to input over the bus
- b. Prepare to output over the bus

IF C = 010 or 011, Generate commands to be sent over the inter-cell bus

There are eight basic commands $x_0 - x_7$ generated here. They will be defined below. Section 10 discusses how they are formed from the CC instruction in great detail. All these commands are sent out over the bus and Section 10 describes their functional usage.

<u>L</u>	<u>Command</u>
000	x_0 - Global Mode Command
	a) Format Instructions
	$x_0 - 1$ A Format
	$x_0 - 2$ D16 Format
	$x_0 - 3$ D32 Format
	$x_0 - 4$ A, D16 Format
	$x_0 - 5$ A, D32 Format
	$x_0 - 6$ I Format
	$x_0 - 7$ DS Format
	$x_0 - 8$ End DS Format

<u>L</u>	<u>Command</u>
000	x_0 - Global Mode Command (Cont) b) State Control via Levels x_0 - 9 Go to Global State x_0 - 10 Go to Dependent Local State
001	x_1 - Global Mode Command x_1 - 1 Go to Independent State x_1 - 2 All Dependent Cells at this Level Reply x_1 - 3 All Dependent Global Cells at this Level Reply x_1 - 4 Go to Dependent Wait State
010	x_2 - Report Communication Request Status
011	x_3 - Input
100	x_4 - Output
101	x_5 - Report Status Word
110	x_6 - Reconfigure
111	x_7 - Extended Format

a) State and Level Control of Cells

- x_7 - 1 Go to Independent State, Set Level
- x_7 - 2 Go to Independent State, No Level Change
- x_7 - 3 Go to Dependent Global State, Set Level
- x_7 - 4 Go to Dependent Global State, No Level Change
- x_7 - 5 Go to Dependent Wait State, Set Level
- x_7 - 6 Go to Dependent Wait State, No Level Change
- x_7 - 7 Go to Dependent Local State, Set Level
- x_7 - 8 Go to Dependent Local State, No Level Change
- x_7 - 9 Go to Controller Cell State

<u>L</u>	<u>Command</u>
111	x_7 - Extended Format (Cont)
	b) Output
	x_7 - 10 Output

IF C = 100 through 111 then transmit or execute mode is entered as indicated.

OP CODE 64 - Long Instruction - 2

This instruction uses 2 words (32 bits) for its mechanization. Several possible instructions that will be included are given below. There is room for several more and future study should investigate other useful instructions. This instruction was primarily designed for the bank/index instruction given below. However, this did not quite use up all the bits. Therefore, it was possible to include many instructions under this one OP code.

1. INCREMENT Bor T TEST AND JUMP

Bits:	6	3	9	2	9	3
Instruction:	OP 64	B/T	Jump Address	INCR	Compare	OP EXT

B/T: Specifies the B or T index/bank register to be operated upon.

Jump Address: A full length address that replaces P if the comparison is equal after incrementing.

INCR: The amount the register is to be incremented (± 1 , 2).

Compare: A full 9 bits for comparison with the B or T register after it is incremented.

OP EXT: Indicates this particular instruction.

2. TRANSFER MEMORY - MEMORY

Bits:	6	3	7	3	7	3	3
Instruction:	OP 64	B/T ₁	m ₁	B/T ₂	m ₂	Spare	OP EXT

Operation: $(M_1) \longrightarrow (M_2)$

B/T₁: B/T register specification to form M₁.

m₁: Address displacement for M₁.

Spare: 3 spare bits are left for further OP code ext.

3. EXCHANGE MEMORY - MEMORY

OPERATION: $(M_1) \longleftrightarrow (M_2)$

This instruction has the same format as (2) above, it can use one of the codes from the 3 spare bits for its specification.

4. COMPARE TOLERANCE AND SKIP

OPERATION: $U + L \geq (M) \geq U - L$

This instruction has the same form as (2) above. It can use one of the codes from the 3 spare bits for its specification. M will be formed from B/T₁ and m₁ while L is located by B/T₂ and m₂. If (M) is out of tolerance then the program counter is incremented by 2 rather than 1.

5. COMPARE TOLERANCE AND JUMP

Same as (4) above except B/T₂ and m₂ are used to form a jump address. L will have to be loaded with the tolerance previously. If (M) is out of tolerance then the jump address replaces the program counter.

6. SET REAL TIME CLOCK

Bits:	6	3	1	9	10	3
Instruction:	OP 64	B/T	I	m	Spare	OP EXT

I: Specifies whether the B/T bits should be used or ignored and m used directly to load the real time clock.

This instruction loads the real time clock with a value located by using m and B/T to form an address and fetch the value to be loaded or by using m as a direct load.

9.2.4 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 to the memory.

The comparison flip-flops represent conditions, greater than (G), less than (L), and equal (E), 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 compare FF) 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.

When an interrupt occurs, the control unit generates a store status sequence after the present instruction is completed. This sequence will be explained below under interrupts (Section 9.2.6). Any of the interrupts can be masked by the interrupt mask register (IMR).

The section of the control unit labeled 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, SCR, and LENR register contents. The signals are then sent out on a number of lines to the remainder of the processor and to the memory.

The state register and the CC mode register play an important role in the control unit's operation. The state register will be used to interpret instructions in ways dependent on the state of the cell. The general functions here were described in Section 6 under group architecture when the instruction execution was discussed. For example, jump instructions may modify the program counter in independent cells and the level register in dependent cells. If the cell is in the dependent global state, the program counter is not used and the control unit will use the instructions received over the bus. The format instructions (Given Address, etc.) modify some of the normal sequences of control that are generated in the IDCG.

If the cell is in the controller cell state, the CC mode flip-flops will be sampled by the IDCG for every instruction. These flip-flops determine whether the instruction is to be executed internally or sent out over the inter-cell bus. Several hard wired addresses are also contained in the IDCG unit. These are used for the interrupts and for certain types of interrupts due to inter-call bus commands decoded for this cell. It should again be noted that Section 6 contains a general discussion on many of the functions that the IDCG unit will control or generate.

9.2.5 Timing

As shown in Figure 9-9, each cell contains its own clock. For preliminary design purposes a clock frequency of 2 MHz has been assumed. This number may certainly be varied depending on the technology without affecting the cell design. The clock drives two counters as shown in Figure 9-9, the RTC Ext, and the bit time counter (BTC).

A real time clock as shown in Figure 9-10 will be used in each cell. The lower 23 bits of this clock are hardware registers, while the upper 16 bits, or more if desired, are in the memory. The hardware portion is divided into two sections, a 16 bit clock register (RTC) and a 7 bit clock extension register (EXT). The RTC register can be set and read by instructions. Once it is set it will count down to zero and send out an interrupt. The EXT register cannot be read or set, but it can

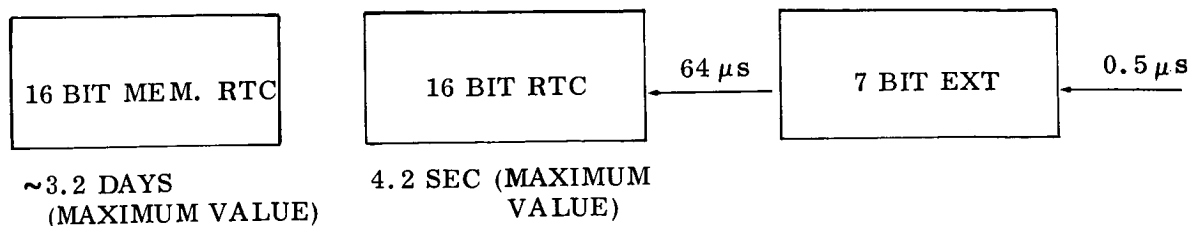


Figure 9-10. Real Time Clock

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.

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 9-10. This weighting was chosen to allow for fairly low frequency periodic programs, up to 4.2 sec, while not being forced to waste too much time in order to handle higher frequency programs. A maximum of 64 μsec may have to be wasted, but the majority of this time is actually well spent in storing the registers of the interrupted program. Further detailed discussion on using the RTC to interrupt and schedule periodic programs may be found in Ref. 1.

The bit time counter (BTC) will be two bits and is incremented by the clock. (This assumes a memory cycle will take 2 processor clock times). It will in turn increment the mode counter (MC) that is used to keep track of the phases of instructions longer than a memory cycle. The mode counter has been set at 4 bits so that it will provide at least 8 counts for long instructions, such as divide and provide room for growth up to 16 if complicated instructions are later added to the list. Therefore, it will be assumed that the BTC sends two lines to the control section of the processor and the MC eight lines to the control section.

9.2.6 Interrupts

There are several types of interrupts that may occur in the system:

1. Real time clock counts down to 0
2. Software or machine error
3. I/O Buffer register operation complete
4. Inter-cell bus I/O complete

5. Timer interrupt due to no neighbor response
6. Timer interrupt due to no response to a GC X_2 command sent out
7. Timer interrupt due to time interval between words received over the inter-cell bus exceeding some value.

Any of the above interrupts may be masked out by the Interrupt Mask Register (IMR). This register may be set by the programmer. Some of these interrupts were discussed in previous sections of the report. The remaining ones will be discussed in Section 10 and 11.

The general actions that occur upon the occurrence of an interrupt will be presented below.

The present instruction will be completed and certain processor registers saved before giving control to the interrupting routine.

1. Complete present instruction
2. Pick-up hard wired interrupt address
3. Store L register
4. Store processor status (see status word below)
5. Store U back in its memory location
6. Store B_0 register

The address of where the processor registers will be saved is given by a hard wired address in the processor. This will be picked up and placed in the MAR. The L register will be saved in the first location. Then the MAR will be incremented and the status word stored. The status word is shown below:

bits:	9	2	1	2	2
Status Word:	P	HA	O	CFF	CC MODE

- P: Program Counter
- HA: Hardware Accumulator ID bits
- O: Overflow Flip-Flop
- CFF: Comparison Flip-Flops
- CC MODE: The CC Mode Register

The MAR is incremented again and the B_0 index/bank register is moved from its regular memory location to the location specified by MAR. The MAR is incremented again and placed in P and the memory location normally used for B_0 .

This final location provides a jump to the executive routine for processing the interrupt. It also loads the B_0 register with a bank address for the executive program to use to address operands. Depending on what action is to follow the interrupt, the executive may have to store the remainder of the processor registers; (U_1 , U_2 , U_3 , etc.) however, it can easily do this by instruction since the program counter and B_0 have been saved and B_0 is then able to be used for addressing storage.

A different executive interrupt routine will be entered depending on the particular interrupt. This is accomplished by incrementing the program counter an appropriate number of times before using the final location as a "jump to" address. Thus in the above example the P register was not incremented before the jump, this could have been a RTC interrupt. For a machine error interrupt the P register might be incremented once before jumping, etc. for the other interrupts.

The priority scheme among the interrupts has not been defined yet. Further study would dictate the relative priority among the interrupts.

To restart the interrupted program, the reverse procedure is essentially followed. The registers that were saved by the interrupting program are restored and then a LOAD STATUS instruction is executed (OP Code 53b). This instruction takes the full 9 bit address from the instruction word and places it in the MAR. This address will specify where L and the status word were saved by the interrupting program. First L will be loaded, then MAR will be incremented by 1 and the status word picked up. The status word will reload P and the other registers described previously. The original interrupted program is then ready to continue. It should be noted that it is not necessary to restore B since this can be done with a regular load B_0 instruction. This is possible since the LOAD STATUS instruction does not use any B/T registers.

9.2.7 Neighbor - Neighbor Communications

Some of the hardware associated with implementing neighbor communications was discussed previously.

The functional operation is described here. One cell will pass a word of data to a neighbor cell. The sending cell, called here CO, will be executing a program PO. The receiving cell (C1) will be executing a program (P1) that requires the word of data contained in CO.

The program PO will place the word in one of its accumulator registers and set a flag associated with that register. This flag will indicate to whom the data is to be sent. Thus the flag can be set to 5 states: N, S, E, W, or X. The X setting is the normal state of the flag and indicates that the register contents are for the use of cell O only. Unless an instruction is executed to change the flag state, the registers can only be used by cell O; the flag will always be X. If the flag is set by an instruction to N, the North cell is to receive the register contents. By setting the flag to S, E, or W, the South, East, or West neighbor can receive the data.

Subsequently, the program in cell 1 needs the data from cell 0. Program P1 will execute an instruction with the following format.

OP CODE 59	O	OP 2	R ₂	R ₁

- OP 2 - Is the operation to be performed, and is any of the usual register-to-register operations (add, subtract, load, etc.)
- R₂ - Is the accumulator register into which the results will be loaded. This accumulator may also furnish an operand for some instructions.
- R₁ - Is the relative location of the neighbor cell (N, S, E, W) which will have a register with the proper flag.

The execution of this instruction will cause cell C1 to request from C0 a word of data. If C0 has a register with the proper flag, C0 will send C1 the data word. Each cell contains a buffer register, ION (Serial line used between cells); thus the transfer will not destroy any accumulator contents. After the transfer is complete, cell C1 will perform the operation specified by the OP Code, using the transferred word as one of the operands.

After a data word has been transferred the register flag in C0 is reset to X. By executing an instruction, the cell can test its flags. In this way, a cell can verify that the neighbor cell called for and was sent the data. This flag test may provide a means of detecting a malfunctioning cell.

Several special cases are outlined below. The descriptions below may be changed as the system is studied further. Actual programming of some sample problems may show ways of improving the system.

If two accumulator registers are set to the same flag value, the lower numbered accumulator will be transferred first. Only one datum will be transferred at any one time for any single instruction execution.

C0 may attempt to alter the accumulator contents before a neighbor cell has requested and received its data word from cell C0. The control circuits in C0 can be set to: (1) wait until C1 receives the data, (2) interrupt to some special error routine, or (3) ignore the flag and use the accumulator anyway. Which options are implemented requires further study and programming examples.

The hardware needed to implement this neighbor-to-neighbor communication scheme consists of a buffer register, accumulator flag registers, and control circuitry. Figure 9-11 shows a block diagram of the hardware. Upon execution of an instruction flagging an accumulator the hardware will set the flag flip-flops associated with the accumulator to the proper state. The control circuitry will be set to a state to expect a request.

Upon execution of an instruction requesting a word from a neighbor, the control in cell C1 will select the proper line and send a request to that cell. If cell CO does not have any flag set for this neighbor, the request will be rejected. (Actually no response will be received. When C1 sends out the request it also sets a timer which allows for a maximum delay in receiving the data. If the timer runs out, an interrupt occurs in C1 and is handled by software.) If cell CO has an accumulator flagged, the accumulator contents will be transferred in parallel to the buffer register in CO. The accumulator flag is then set to X. A bit-by-bit serial transfer will move the bits from CO to the buffer register in C1. The filled buffer register in C1 will be used as an operand.

One aspect of the control circuit should be mentioned here. A single line between any pair of cells gives the highest reliability because of the least number of connections. The only problem comes about when two cells attempt to request data from each other at the same time. One method of control is to have the cells with an even address make requests at one time, and cells with an odd address make requests during alternate times. The address assignment is done by the controller cell via the inter-cell bus. By assigning cell addresses in a checkerboard pattern, every evenly addressed cell will have four odd addressed neighbors.

The scheme described here requires one instruction to send out a word, and one instruction in the receiving cell to pick up the data and operate on it. A total of only one additional instruction compared to a sequential program is therefore required.

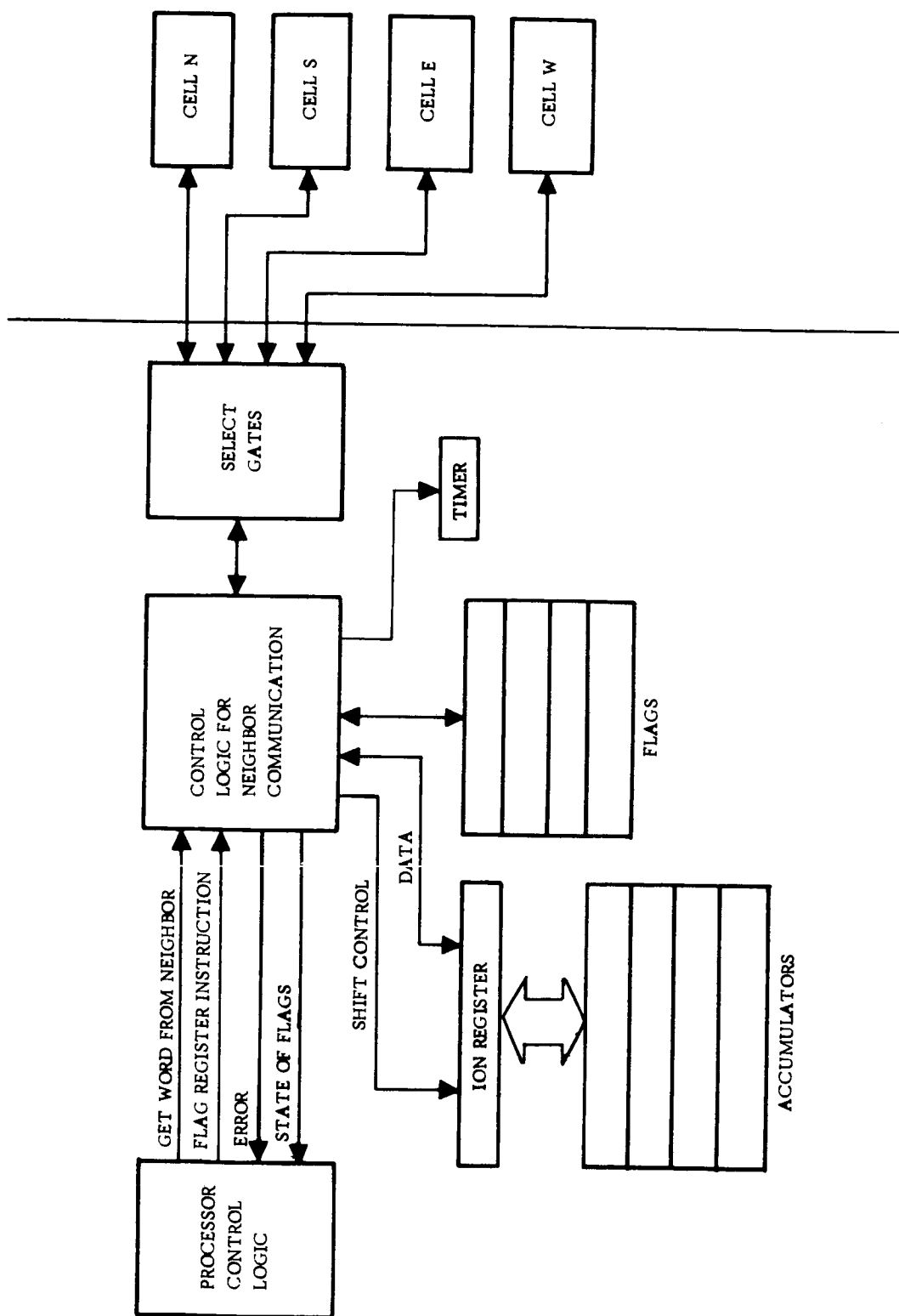


Figure 9-11. Neighbor-to-Neighbor Communication

9.2.8 Inter-Cell Bus Communications

Basically nine lines enter the bus communications section via the BCR. One is used for control and the remainder for 1/2 word data (8 bits). The Bus I/O Control Section will examine every byte on the bus that has the control line set to the one state. It will then determine if this byte is for this cell. This will be accomplished by examining the bits in the byte and determining what type of command this is, and if the cell address (ID) or level register match it (which one is used depends on the command). The Bus I/O Control Section will then issue the proper control signals to execute commands for the cell from the bus. Section 10 will present a detailed discussion of the bus operation, the bus commands, and detailed timing and logical operations to mechanize the bus commands. However, a general discussion of the bus section is in order here to place it in proper perspective to the remainder of the cell hardware.

Some of the commands from the bus result in a pseudo interrupt. That is, they halt the execution of the present instruction (essentially freeze the IDCG and the arithmetic and control units) and provide access to the memory to carry out data transfers. Once the data transfers are complete, the cell is allowed to complete the instruction. This is relatively simple to accomplish since the arithmetic and control registers are not altered during such an interrupt.

Other commands result in: the state or level registers being changed, certain global control instructions (such as the format) etc. Certain commands will also tell the cell that global instructions will be arriving over the bus. The Bus I/O Control then has the responsibility of generating the proper control signals to the IDCG unit and sending it the received instructions.

This section of the cell also contains a Bus I/O register that essentially holds the I/O commands for bus operation so that the Bus I/O control section may generate the proper control signals, such as the memory write signals for inputting a string of bytes from the bus into the memory.

Two timers are also shown in this section of the cell. One timer is used by a cell in the controller cell state to time the response from other cells to certain commands. This prevents the controller cell from being hung up waiting for a response from a faulty cell. If the timer exceeds its maximum value, an interrupt is generated. Similarly another timer is used to time the interval between bytes received over the bus. If the interval exceeds some value an interrupt will be issued if the bus I/O operation has not yet been completed. This has several functions, one of which is to prevent a cell from being hung up waiting for a cell that fails during the middle of a communications between the two cells. The above points will be clarified when Section 10 is referred to. The important point to note here is that the bus I/O control has a very high priority. This approach was taken to provide fast communications over the bus and to not make the controller cell have to wait for cells to complete instructions or programs they are executing before it gets communications control of the cell.

9.2.9 Memory

Each cell contains a 512 word (16 bits per word) memory as shown in Figure 9-9. Part of the Memory (9 locations) are used for processor registers namely the accumulators and B/T registers. The memory receives addresses and data and read/write control signals from the processor. All information read out of the memory is sent to the MB register.

A description of a basic memory cell for storage of a bit of information and how the basic memory cells are used to form the memory will be given below. Figure 9-12 shows the schematic of a conventional complementary MOS bistable circuit without any provisions for reading or writing. Referring to the schematic of Figure 9-12 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_2 . Addition of read and write circuitry will add at least one more leakage current path, making a minimum of three per memory cell.

A block diagram of a possible memory cell design is shown in Figure 9-13 along with a truth table.

The memory cells may be combined into a bit plane array as shown in Figure 9-14. Each bit plane array consists of 512 bits of memory cells shown in Figure 9-13. The bit plane arrays may now be combined into a 512 word -16 bit/word memory as shown in Figure 9-15. This is accomplished by stacking the arrays in parallel. Each array is inputted common row select, column select, read, and write signals. One bit of the data word is fed into each array. It should be noted here that this is the place where yields on the wafers may be increased by manufacturing more

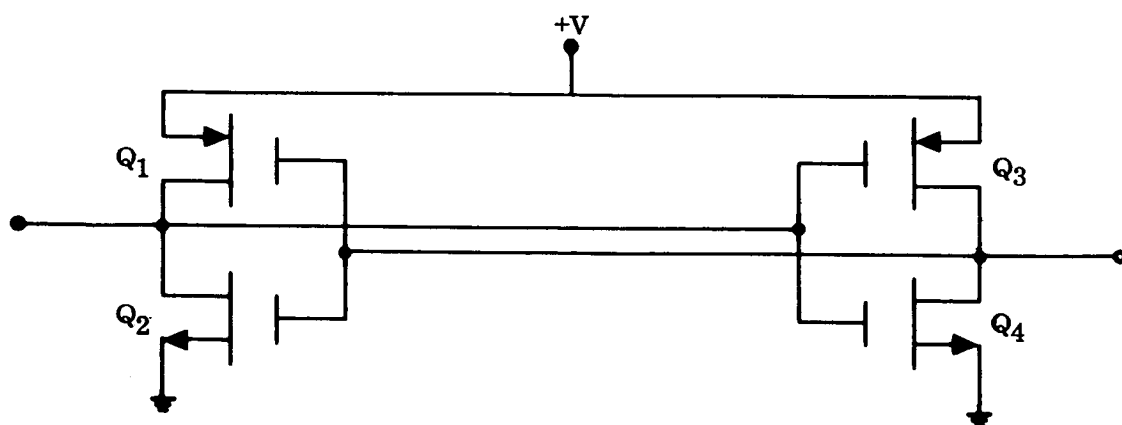
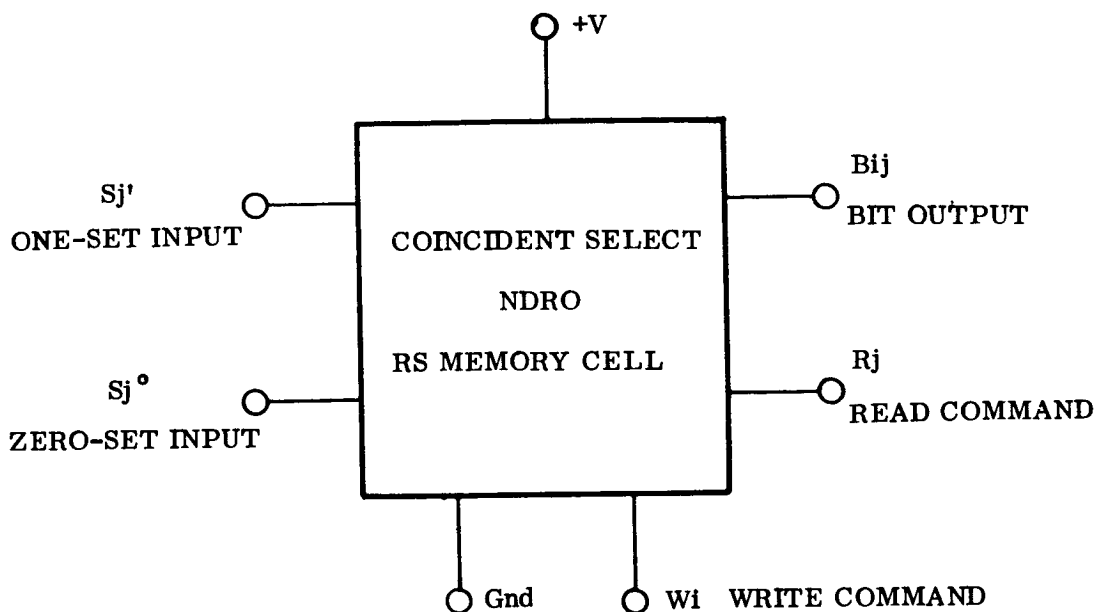


Figure 9-12. Basic Memory Cell Utilizing Complementary MOS Transistors Without Selection or Readout Provisions



<u>FUNCTION OF ij CELL</u>	<u>Rj</u>	<u>W1</u>	<u>Sj'</u>	<u>Sj°</u>
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 "0"	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 9-13. Logical Operation of a Coincident Select Memory Cell

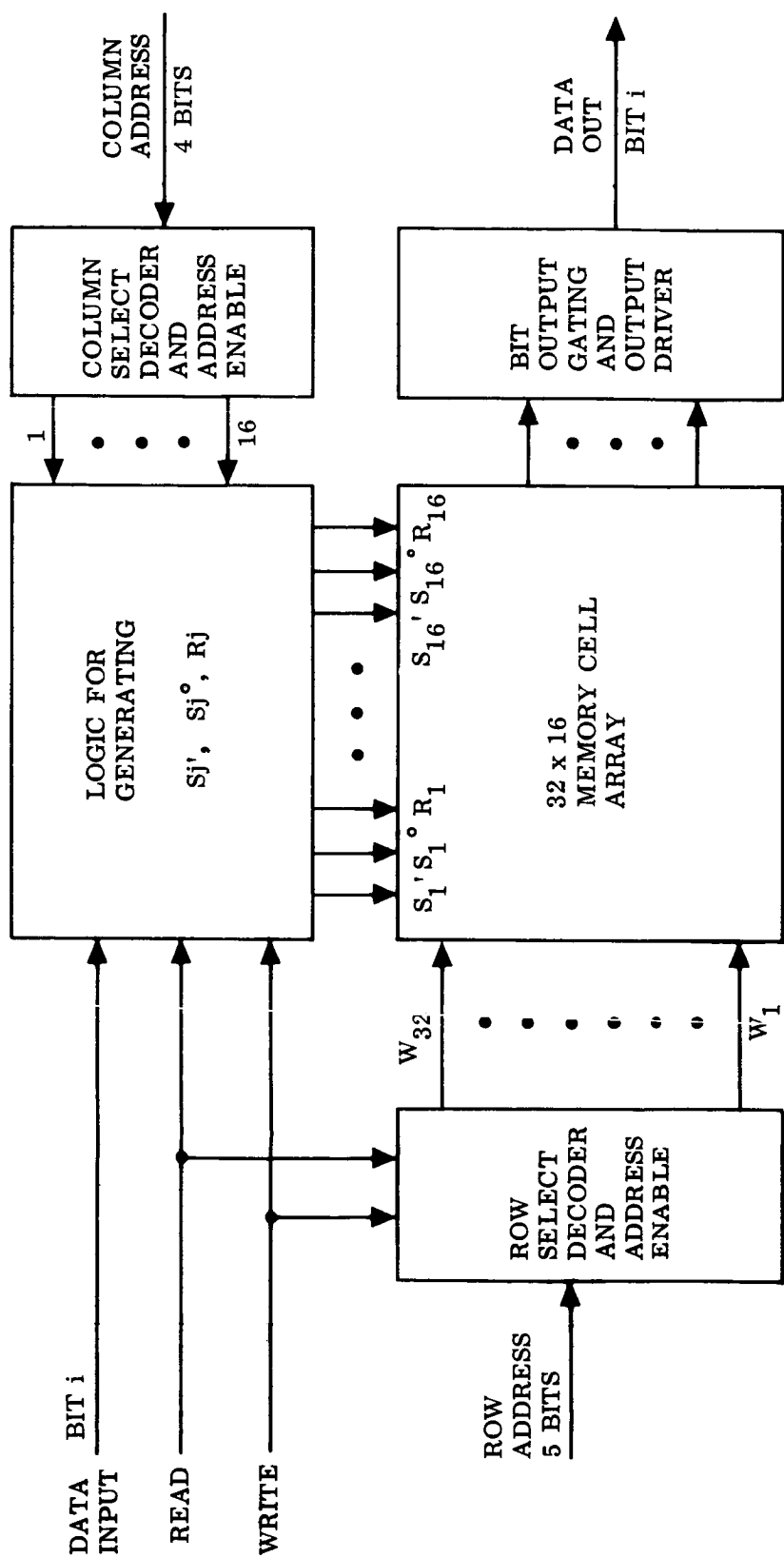


Figure 9-14. Organization of a Coincident Select Memory Cell Bit Plane Array

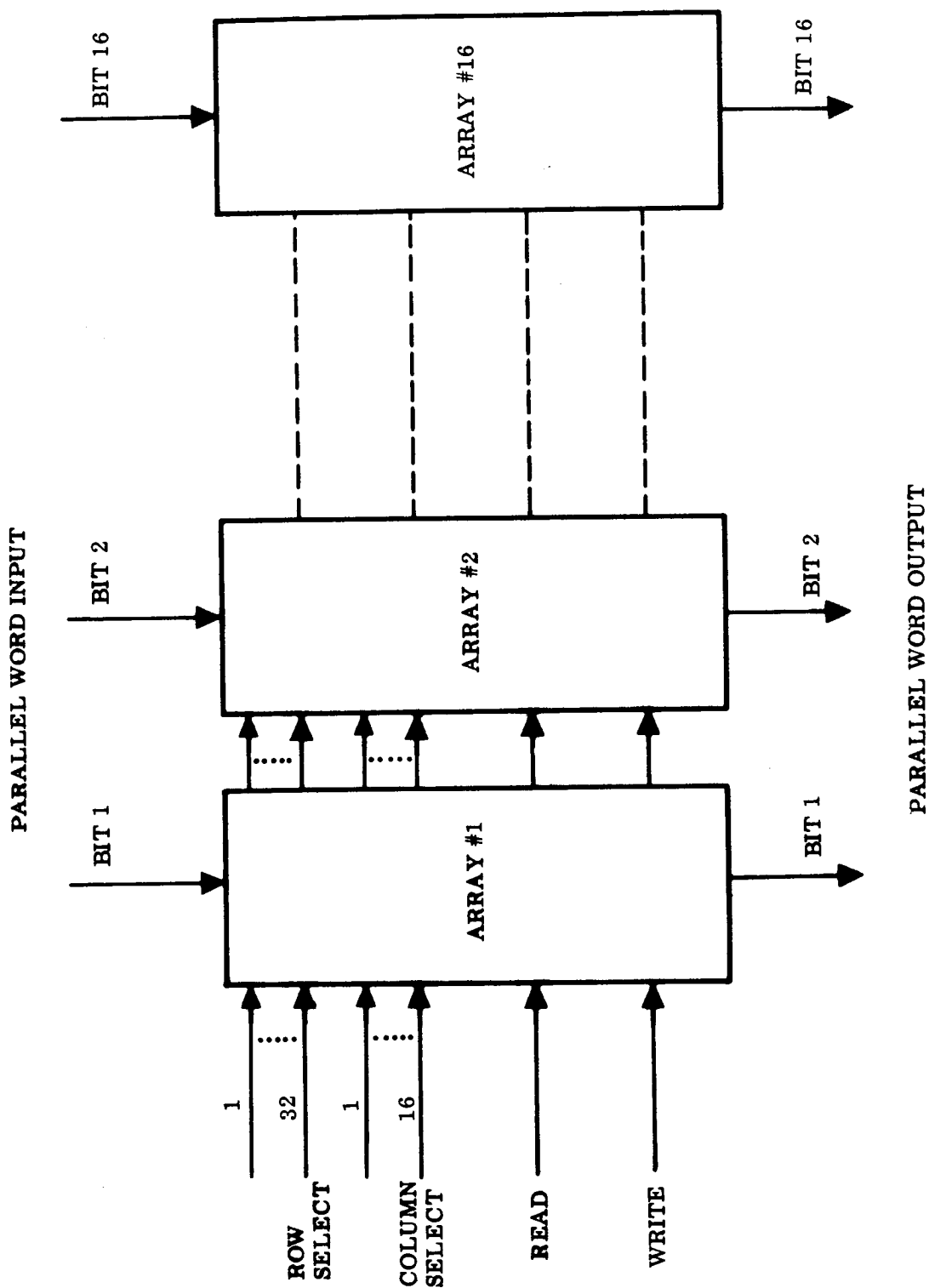


Figure 9-15. Connection of 16 Arrays to Form a 512 Word, 16 Bit, Memory

than 16 arrays on the wafer and selecting which of the arrays to wire together to form the memory shown in Figure 9-15. Finally, to form the entire block labeled "memory" in Figure 9-9 a 9 bit address register and a small amount of control logic would be added to Figure 9-15.

The memory will read and write in two bit times. The first bit time will be used to set up the address decoding and selection circuitry. The second bit time will be used to either write in the data or read out data from the selection cells. It should be noted that the data will be actually written into the selected cells somewhat before the end of bit time 2. However, the data read out of the memory will only be strobed into the processor section at the end of bit time 2. Therefore, the actual write cycle takes somewhat less time to complete than the read cycle. The numbers used in the processor section for bit times were 500 nano seconds resulting in a memory cycle of 1 μ sec. These are conservative numbers and should be readily attainable. The actual figures in the time period of application of the DAMP computer may result in typical memory cycle times of 250-400 nano seconds.

This section has presented only a preliminary design of the memory. Further study may dictate better ways of organizing the basic memory cells. In particular, discretionary wiring techniques to achieve reasonable yields will play an important role in determining how to organize the basic memory cells. It should be noted that Ref 1 contains further detailed analysis of a complementary MOS memory similar to the one presented here.

9.3 GROUP SWITCH

This section will present some preliminary design considerations of the group switches. Referring to Figure 1-1, one can see the relation of the group switches to the cells and the busses. The group switch controls the information transfer between the inter-cell busses and the inter-group busses. Essentially the group switches look like another cell to the group that is the system executive (also referred to as the executive group). This is due to the fact that the system executive group will control the communications over the inter-group bus. This is quite similar to the controller cell in the group controlling the communications over the inter-cell bus.

The main difference between the control of the two busses is that the controller cell receives an instantaneous response to its commands that it sends out to the other cells in the group whereas the executive group does not receive an instantaneous response to its commands sent over the inter-group bus to other groups. This results in different communication system mechanizations for the two busses. This subject will not be gone into in this section but will be treated in detail in Section 10 where both busses and their mechanizations will be discussed. However, a block diagram of the group switch and a functional description of its registers and circuitry will be given below. It will be seen that the group switch is relatively simple in terms of hardware complexity. This should offer the possibility of employing redundancy techniques to design a highly reliable group switch. A block diagram of the group switch is shown in Figure 9-16.

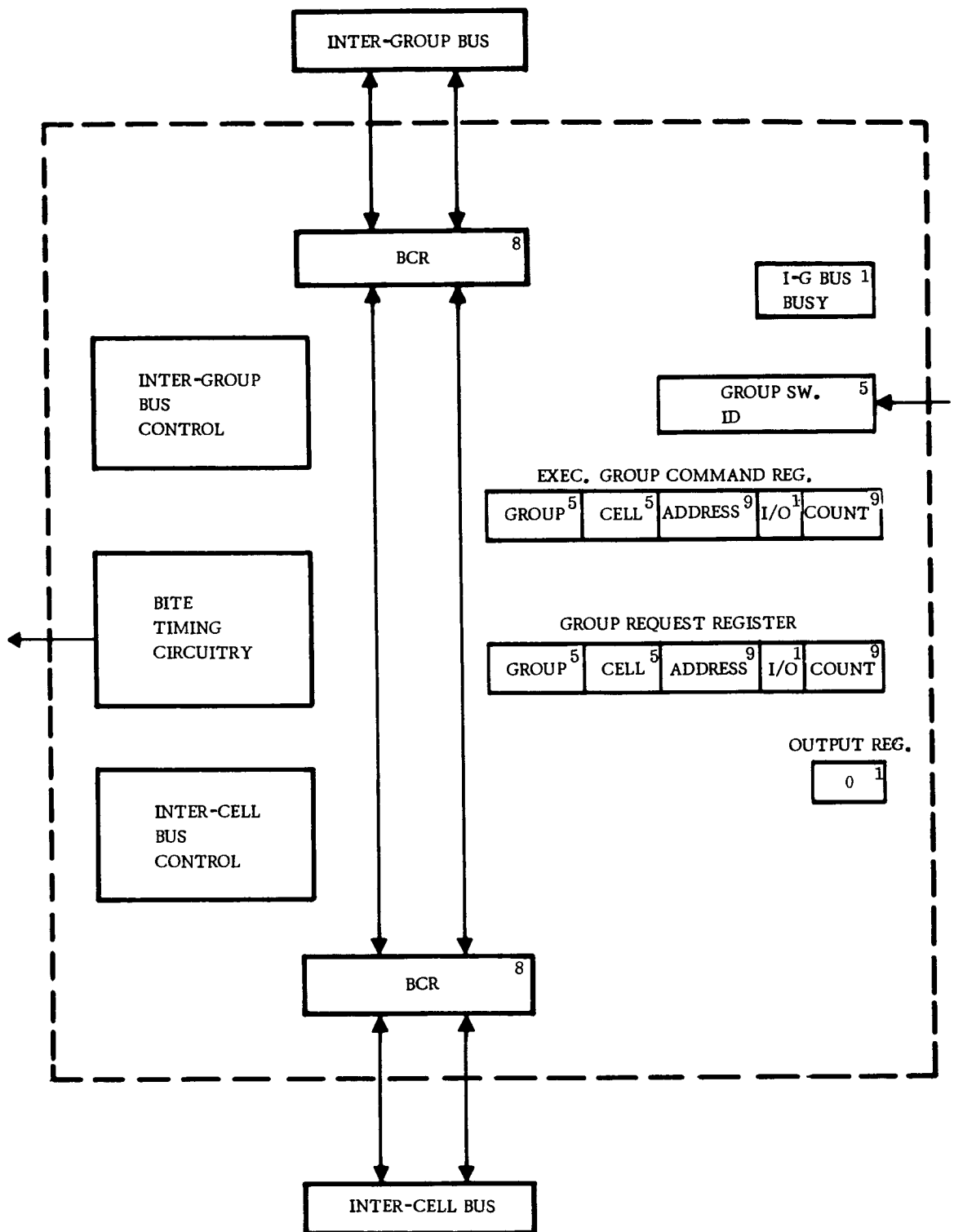


Figure 9-16. Group Switch Block Diagram

A functional description of the hardware in the group switch will be given below:

BCR:	Buffer Communication Register - Two half word long registers that receive the present byte over the bus they are connected to.
GROUP SWITCH ID	A 5 bit register to hold the identification or address of the group switch. This register is used to decide which commands are addressed to the group switch. It has a serial load line for initialization purposes. Note that this provides the capability for having up to 32 group switches or 16 groups (2 group switches used per group) in the system.
I-G BUSY:	Inter-Group Bus Busy Flip-Flop - flip-flop is set when the EXECUTIVE GROUP COMMAND REGISTER is loaded in a group. It is reset when an "end of transmission" command is sent over the inter-group bus. It may be sampled by command over the inter-cell bus.
EXECUTIVE GROUP COMMAND REGISTER	This register holds a command received over the inter-group bus that was addressed to this group switch. It may be sampled by command over the inter-cell bus. It will be loaded with a command for this group identifying the cell, address and number of words to be input or output over the inter-group bus. This register will automatically be reset when it is sampled over the inter-cell bus.
GROUP REQUEST REGISTER	This register holds a request received over the inter-cell bus by a command addressed to the group switch to load this register. It may be sampled by command over the inter-group bus. It will hold the requested I/O over the inter-group bus.
O:	Output Register - This register is set by a command received over the inter-group bus. It may be sampled by a command over the inter-cell bus. It is used to synchronize the start of communications between two groups. This register is automatically reset when it is sampled by command over the inter-cell bus.
BITE TIMING CIRCUITRY	This circuitry consists of a flip-flop that is alternately set-reset by a command from the inter-cell bus. The rate at which it is set will determine the voltage generated across a circuit. Two out of tolerance level detectors across this circuit will check for a high and low value of voltage. If the voltage is out of tolerance a failure signal is issued.

**INTER-CELL AND
INTER-GROUP
BUS CONTROL**

These blocks contain all the control circuitry to decode information received over the busses and execute commands addressed to the group switch. This control circuitry will also generate inter-group bus commands from certain inter-cell bus commands. It is responsible for opening up the communication path between the two busses.

10. COMMUNICATION BUS

The inter-cell and inter-group bus communications will be described and mechanized in this chapter.

10.1 INTER-CELL BUS

This section will discuss the operation of the inter-cell bus. A functional description of the bus will first be given followed by a detailed mechanization of the operations that take place on the bus. The bus is under complete control of the cell in the group designated as the controller cell. The number of lines required in the bus depends heavily on the communication rates required. Determination of this number is not possible at this stage of the machine design. However, for preliminary design purposes an 8-bit parallel bus shall be assumed. This then provides for 1/2 word bytes to be transmitted over the bus (16-bit word length in the cells).

The inter-cell bus communications may be divided into two classes: (1) communication between cells in a group and (2) communications between the controller cell and the group switches of the group. The latter type of inter-cell bus use is primarily concerned with accomplishing intergroup communications. Therefore, it will be treated in the following section (10.2) where the intergroup bus is discussed. This section will treat the inter-cell bus operations that take place in the group between the cells only. When the "bus" is referred to in this section, it will imply the inter-cell bus.

Communication on the bus takes place in basically two types of modes, local and global. Local use is basically communication between two cells with control set up on the basis of cell address identification and not dealing with the control of levels or states of cells. Global use implies communication of the controller cell with one or more of the other cells with control set up on the basis of cell address identification, levels, or states for the purpose of global control of the cells.

The bus may be used for both instructions and data. Local use of the bus is basically for passing data amongst cells and amongst cells and I/O devices connected to the bus (see I/O section for a discussion of the I/O operation). Global use of the bus may be for instructions and/or data. In any case the controller cell sets up and controls all information flow over the bus. Software routines are set up in the controller cell to control the operation of the bus. There are basically two types of operations in these routines: fixed periodic and background. Fixed periodic operations on the bus are those that must take place at predetermined intervals. These may be either local or global type of communications. Examples of such operations are a cell requiring updating of navigation and guidance parameters computed in another cell every second, a set of global operations comprising a periodic program that must be computed ten times a second, etc. Background type of operations on the bus are those that take place in the absence of any fixed periodic operations; in other words operations fitted in between the fixed periodic operations. The executive section (Section 11) should be referred to for a discussion of the software routines for bus operation.

A total of ten lines are used for the inter-cell bus. One line is used to denote control or data and is designated the control line. Another line is used for parity; future studies may dictate the need for more lines for error control, however for the present only one line shall be assumed. The remaining eight lines are used for control or data words. The lines are bidirectional so that a cell may receive or transmit over the same line; this is accomplished by the use of driver/receiver circuits at the interface with the lines in each cell (see Ref. 33 for a discussion of these circuits).

The use of the eight lines used for control/data words, in particular for control purposes, will be described below. There are basically two types of control words used: Local and Global. Local words are distinguished by requiring a particular cell identification or address to be specified while global words require the specification of an identification address or certain global levels or modes. The control words are decoded by all the cells and only the appropriate cells partake in or accomplish the desired communication on the inter-cell bus.

All words over the bus are composed of eight-bit bytes whether they are control or data words. The control words are identified as such by use of the control line: the first byte of any control word is identified by the control line set to a one (control) state. The control line will return to a zero after the first byte of a control word. Some control words require more than one byte, this is accomplished by use of special control word formats as will be explained below. The format of the first byte of a control word is shown below:

lines:	C	1	2	3	4	5	6	7	8	P
	Cont/ Data	Command			Cell Address or Global Control				Parity	

Three lines are used for the command thereby providing eight possible command states and five lines for the cell address. This provides for addressing up to 32 devices or cells. The group may consist of typically 20 cells for the manned Mars mission considered, this would then leave room for 10 I/O devices to be connected to the bus (2 addresses are required for the two group switches connected to each intercell bus). It also provides for expansion so that more cells may also be added. If 32 addresses are not sufficient for addressing cells and I/O devices on the bus, it is relatively simple to provide for expansion beyond 32 by using an address extension scheme. This involves using some address, say address 32, to signify that the next byte contains more address bits. This of course has to be designed into the hardware, however it is relatively simple to implement and can provide for a high degree of expandibility both in terms of cells and I/O devices connected to the intercell bus. Of course it is possible to provide this expandibility in the cells or the I/O devices only, for example providing up to 31 cells and group switches and using address 32 for an address extension scheme in the I/O devices only, etc.

Table 10-1 lists the types of communication required to be carried out on the inter-cell bus: (this list includes both local and global operations). It should be noted that I/O devices are also included in the term cell used in Table 10-1.

Table 10-1. Communication Bus Operations

1. Controller cell to send words directly to a cell under controller cell's command.
2. Controller cell to receive words directly from a cell under controller cell's command.
3. Controller cell to scan bus usage requests from individual cells and establish communication between two cells based upon requests.
4. Controller cell commanding communication between two cells.
5. Controller cell issuing a command to a cell specifying some changes to the internal control state.
6. Controller cell issuing global commands to one or more cells or a cell issuing an end of transmission command.

The operations listed above are mechanized by various control words. Each control word is composed of one or more bytes, the first byte having the format given above. The first byte will be identified as a command; a list of commands is given below in Table 10-2.

Table 10-2. Communication Bus Commands

Command No.	Code	Description
X_0	000	Global mode command
X_1	001	Global mode command
X_2	010	Report communication request status
X_3	011	Input
X_4	100	Output
X_5	101	Report Status Word
X_6	110	Control Reconfiguration
X_7	111	Extended command format

Commands X_0 , X_1 :	These commands are used for global operations, the remaining 5 lines are not used for cell address purposes, but will be used for various global control functions.
Command X_2 :	This command requests a response from a given cell as to the status of its requests for use of the communication bus.
Command X_3 :	This command tells the cell to input the next set of data words on the bus. (A set of data words is defined as the words on the bus in between control words)
Command X_4 :	This command tells a cell to output a set of data words on the bus.
Command X_5 :	This command requests a cell to send to the controller a status word representing certain control states in the cell.
Command X_6 :	This command forces a cell to perform some change to the internal control state (e.g., turn on/off, etc.).
Command X_7 :	This command uses an extended format. It requires the second control word byte to identify what the command consists of; this then provides for more than the eight basic commands listed here. This command will also be used to change cells from local control modes to global control modes as will be explained later in this section.

10.1.1 Local Operations

The mechanization of the required communication operations on the bus will now be discussed. A description of the control words will be given for each operation. The first five operations listed in Table 10-1 may be identified as local operations; these will be given below.

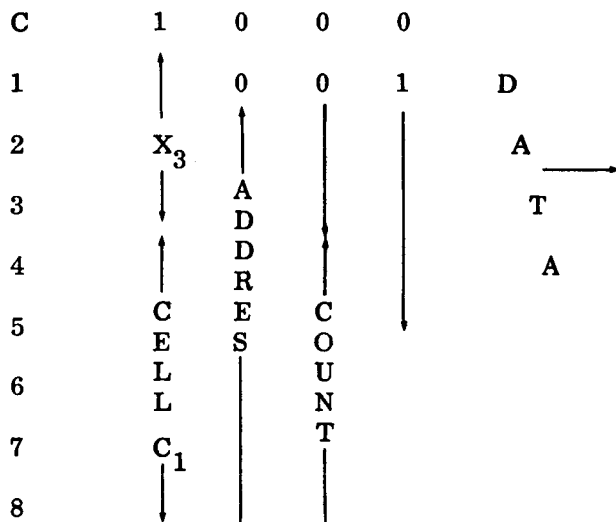
1. Controller to send words to a cell:

Sender: C_0 C_0 C_0 C_0 $\longrightarrow$

Receiver: ALL C_1 C_1 C_1 $\longrightarrow$

BYTE: 1 2 3 4

Lines

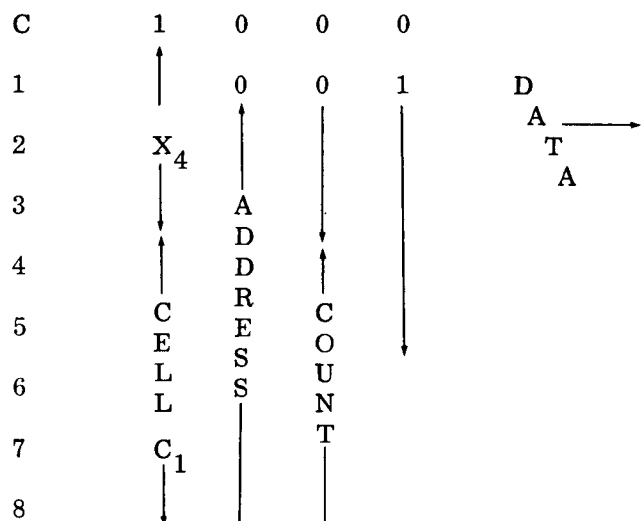


The first byte of the control word is identified by a one on the control line. As mentioned previously the first byte contains the command, input, and the address of the cell, C_1 . A 9-bit address is sent in the second byte and part of the 3rd byte to identify the first location in the cell to be input to. The number of words to be input to the cell are specified by the count which is sent in the remaining part of the third byte and part of the fourth byte. The use of only three bytes in the control word provides for a count of up to 32 words while four bytes provide for a count of up to 512 words. The cell determines when the control word is complete by the 0 to 1 transition on line number 1 as shown above; this scheme provides for the capability of a variable length control word format and saves the transmission of one byte when less than 32 words are to be input. The following definitions will be used to identify the cells: C_0 , the controller cell, C_1 and C_2 , the cells used in the communication process.

2. Controller to request words from a cell

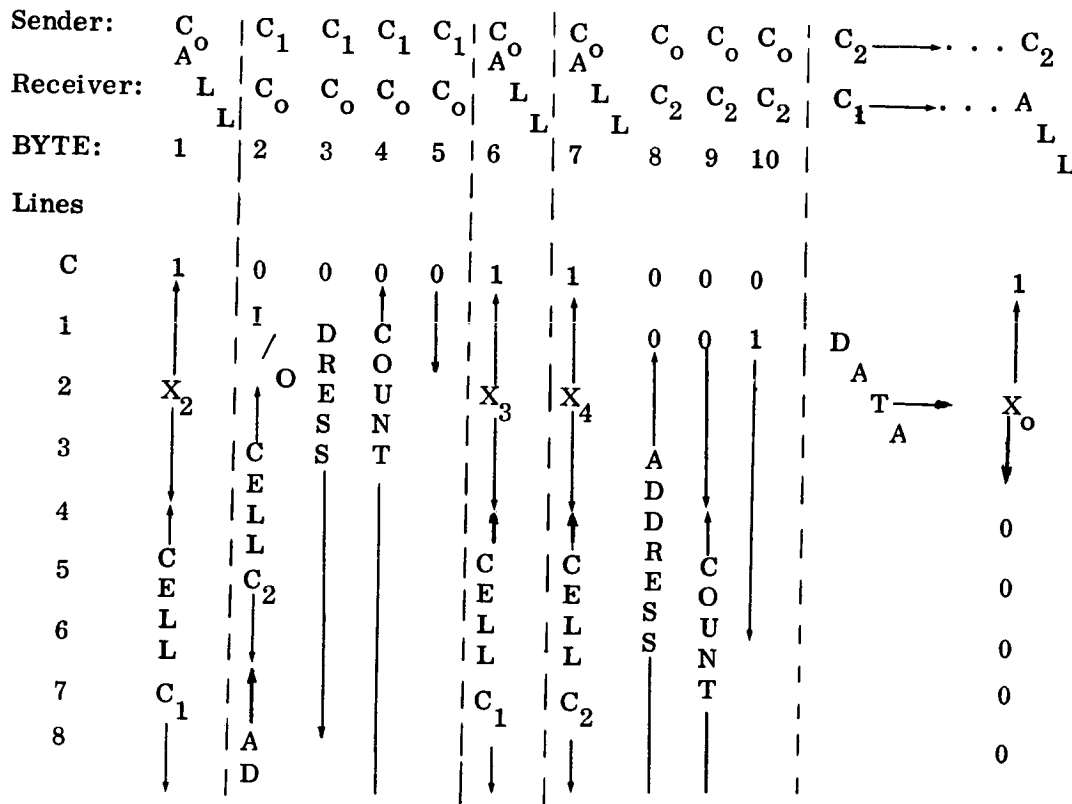
Sender:	C_o	C_o	C_o	C_o	C_1	→
Receiver:	A_{L_L}	C_1	C_1	C_1	C_o	→
BYTE:	1	2	3	4		

Lines



The same discussion as for 1 above applies here except that the control word applies to outputting data from cell C_1 now.

3. Controller to scan a cell for communication requests - Cell, C_1 , to request words from cell C_2 .



The controller cell outputs the first byte of the control word sequence, X2, which asks a particular cell, C₁, if it has a request for service on the bus. Cell, C₁ responds by outputting a response word as shown above. In this particular case the first byte of the response identifies the desired operation (input for this case), the Cell, C₂, communications is desired with, and part of the address in the cell C₂. The address and number of words are completed in the remaining three bytes. The address in this case specifies the location in cell C₂ from which the words are desired. If a cell has no request the response will consist of two bytes all zeroes. It should be noted that cell C₂ could also be the controller cell with whom a request is made for communications with. The control word response could actually be formed in three bytes, however using four bytes results in an easier mechanization within C₁ and actually does not result in requiring more time on the bus as will be seen later in this section.

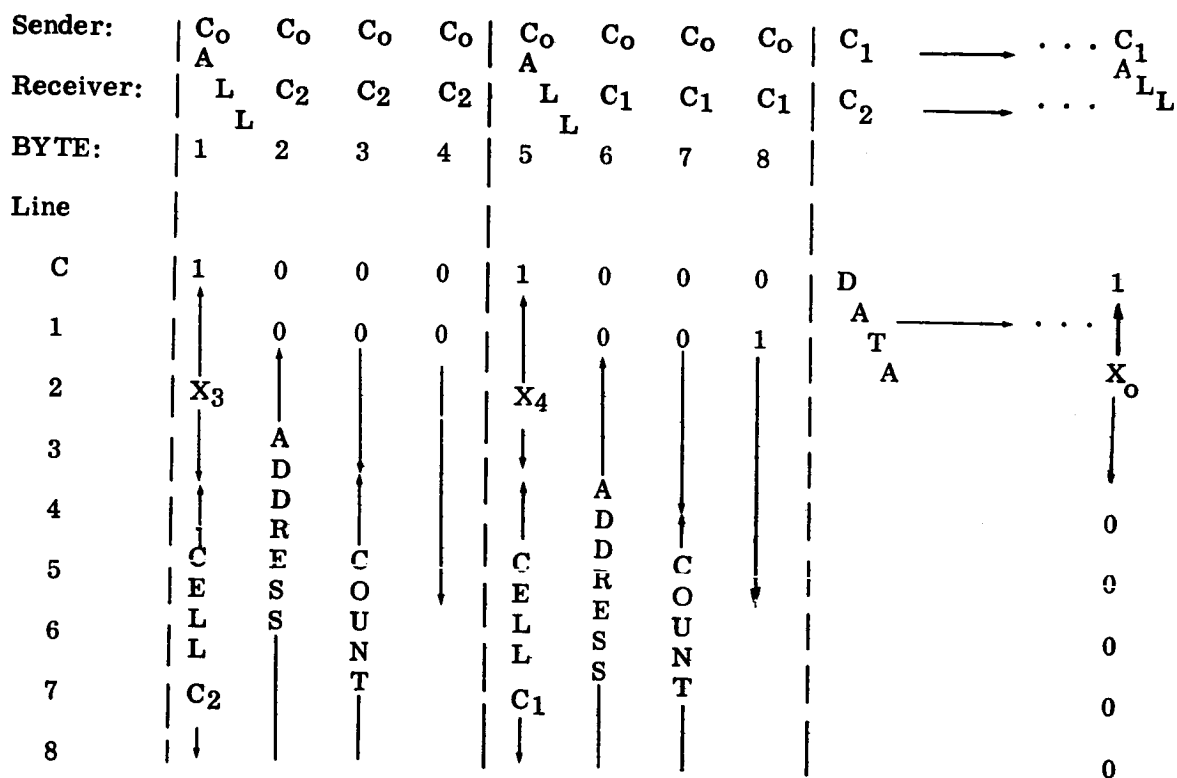
The controller cell accepts the response from the cell and examines the request. It will determine whether the full word count requested can be accommodated on the communication bus. The controller cell then outputs the sixth byte of the control word which is an input command to the cell, C_1 , that requested the words. Next the controller cell outputs the remaining control word bytes telling Cell C_2 to output a certain number of words (may be reduced below that requested by C_1) starting at a location specified by the address. The same comments apply as before with regards to varying the length of the X_4 command should less than 32 words be desired to be communicated.

4. Controller to scan a cell for communication requests - Cell, C_1 , to request to send words to cell C_2



The mechanization of this operation is very similar to that described above for the input request by a cell. In fact the first five bytes are identical except that an output is indicated in the second byte. The cell, C_2 , to whom words are to be sent is told to input by bytes 6 through 9; note that byte 9 does not have line 1 in a 1 state. This prevents cell C_2 from picking up the next three bytes as data words. The requesting cell, C_1 , is told to input by means of an X_7 extension command (output - word count), this is used since all the controller can tell cell C_1 is how many words it may output since it does not know the address they will come from (if it did it could use an X_4 command). The X_7 command is a variable length command and therefore uses a 0 to 1 transition on line 1 after the first byte to signify its completion; note that this transition is also used to complete the X_3 command given previously to cell C_2 . In addition, cell C_1 will send an end of transmission command as the last byte.

5. Controller Cell to tell one cell, C_1 , to output to another cell, C_2 .



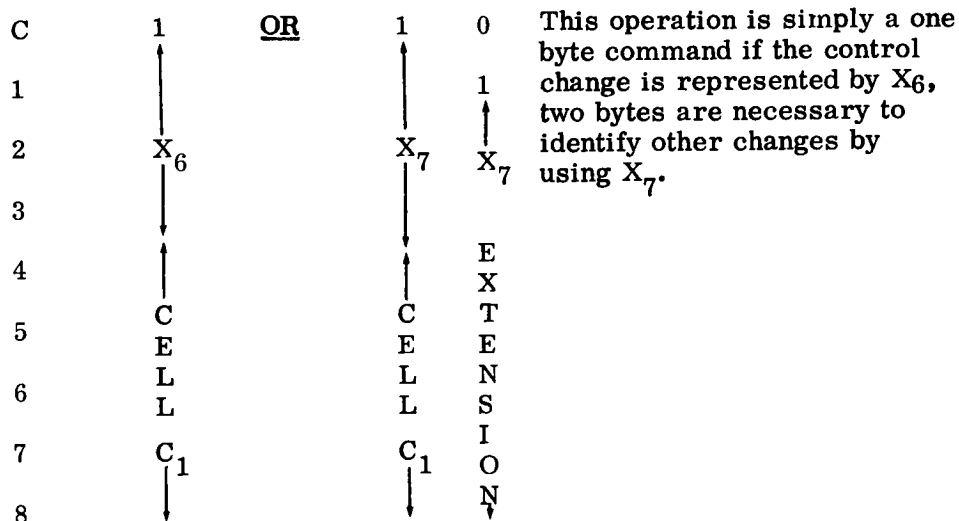
This operation requires an output command as described previously to be given to the cell C_1 and an input command as described previously to cell C_2 . The only difference is that a 0-1 transition on line 1 is inhibited in byte 4 so that cell, C_2 , may not start receiving data until byte 8 has been transmitted.

6. Controller Cell to tell one cell, C_1 , to input from another cell, C_2 . (Same as above.)

7. Controller Cell to command a cell, C_1 to reconfigure some control state.

Sender: C_0 C_0 C_0
 Receiver: A A A
 L L L
 L L L
 BYTE: 1 1 2

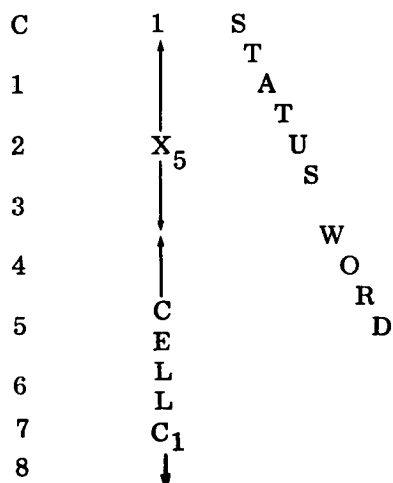
Line



8. Controller to Command a cell, C_1 , to report its status word.

Sender: C_0 $C_1 \rightarrow$
 Receiver: A A
 L L $C_0 \rightarrow$
 L L
 BYTE: 1

Line



This command forces cell, C_1 , to output a status word to the controller.

It should be noted that there are a number of alternatives in deriving the formats presented above. For example, it is possible to only use one byte for the X_3 input command, since the first word transmitted could contain address and word count information. However, this provides no transmission time saving on the bus and contaminates the data being sent and received with control information. It is also possible to not use the variable command scheme requiring the 0-1 transition on line 1 for certain commands (X_3 , X_4 and X_7). It would thereby provide for up to 128 words with three bytes for the X_3 and X_4 commands. To go beyond 128 words, one could use a word count of 128 to signify that the next byte is to be used as an extended word count, this could result in some savings on the bus. However it would not be possible to use the 0-1 transition logically to delay the start of certain X_3 and X_4 commands as was explained above. To accomplish this would require additional logical circuitry and/or transmission of additional bytes so there may not really be any savings in eliminating this logical function.

10.1.2 Global Operations

This section will present a description of global control of the inter-cell bus; these operations are classified as number 6 in Table 10-1. The same format is used as presented in the introductory section (10.1). It was pointed out there that commands X_0 , X_1 , and X_7 are used for global control. Commands X_0 and X_1 do not specify a 5 bit cell address identification but use the bits for control purposes; these commands will be one 8-bit byte long. Command X_7 will utilize a 5 bit cell address identification. As mentioned previously it uses command extension and is variable in length to offer the possibility of many control commands within X_7 .

1. Format GC Instructions (X_0)

One byte is used here and the format is shown below:

lines:	C	1	2	3	4	5	6	7	8	P
	1	0	0	0	Y		L			Parity

- Y = 11 indicates format type GC
- L = 000 indicates end of DS data words
 - 001 indicates A - given address
 - 010 indicates D16 - 16 bit data word
 - 011 indicates A and D16
 - 100 indicates D32 - 32 bit data word
 - 101 indicates A and D32
 - 110 indicates I - Immediate data
 - 111 indicates DS - Data is being sent (A is also sent here)

The functional explanation and usage of this and the remaining global operations may be found in the chapter on group architecture (Section 6); reference should also be made to the instruction set (Section 9) for further clarification of these operations.

2. Level Control GC Instructions (Class 1) - (X₀)

The same one-byte format as given above is used here.

L is always the level number to be compared to the contents of the level register in the dependent cell.

Y = 01 G - indicates all cells at this level go to global state, instructions for this level follow

10 L - all cells at this level go to local control state

3. End of Transmission Command - (X₀)

lines	C	1	2	3	4	5	6	7	8	P
	1	0	0	0	0	0	0	0	0	Parity

This command is used to reset the bus busy flip-flop in every cell. Actually it will only have meaning in the controller cell since it is the only cell controlling the bus use. It is the only command that may be generated by any cell other than a controller cell.

4. Level Control GC Instructions (Class 2) - (X₁)

One byte is used here and the format is shown below:

lines:	C	1	2	3	4	5	6	7	8	P
	1	0	0	1	Y		L			Parity

L is always the level number

Y = 00 R - all dependent cells at this level reply with a constant, to be sent on the inter cell bus.

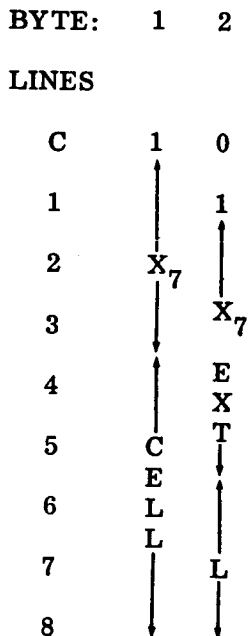
01 R, DG - all dependent global cells at this level reply with a constant.

10 IND - all cells at this level go to the independent state.

11 W - all cells at this level go to the wait state.

5. Individual Cell GC Instructions - (X₇)

Recall that the X₇ extended format command is used here, the format is shown below:



The three variable fields have the following meanings:

CELL The cell address that is compared to the cell's ID register.

L: The level number that is to be loaded into the cell's level register.

X₇EXT: The operation to be performed by the receiving cell.

When X ₇ EXT = 1000	IND, L	Go to independent state, set level
1001	IND	Go to independent state, no level change
1010	G, L	Go to dependent global state, set level
1011	G	Go to dependent global state, no level change
1100	W, L	Go to dependent wait state, set level
1101	W	Go to dependent wait state, no level change
1110	L, L	Go to dependent local control state, set level
1111	L	Go to dependent local control state, no level change
0111	CC	Go to the controller cell state.

The remaining X₇ EXT codes are used for other operations as given in Section 10.1.1.

The use of the instructions presented in this section was given in Section 6 and reference should be made to clarify their use in global operation of the bus.

10.2 INTER-CELL/INTER-GROUP COMMUNICATIONS

This section will describe how the intergroup communications are carried out. This requires a description of how the group switch uses the inter-cell bus and how it uses the intergroup bus.

Communications over the intergroup bus are somewhat different from that described in Section 10.1 for the communications within a group. The difference is that the cells in a group are under immediate control of the controller cell and immediately respond to commands over the inter-cell bus. However, the executive group that is in charge of controlling the intergroup bus use does not have the groups (represented by the group switches) under its immediate control. The groups periodically sample the group switches to see if any commands have been placed in the "Executive Group Command Register" by the executive group. (Any group can do this; however only the group acting as the executive group will be issuing such commands.) The groups then respond to these commands. It can be seen that there will be some delay in getting a response to these commands. This delay is primarily dependent upon the rate at which the group switch is sampled by the controller cell of a group for these commands. Once the controller cell picks up these commands, it will immediately respond by commanding the proper cells in its group to begin communications.

The groups will place requests for intergroup use in the group switch (in the GROUP REQUEST REGISTER). These requests are periodically sampled by the executive group. It is based upon these requests and any individual requests of the executive group, that commands will be placed in the group switches for intergroup bus control.

It is expected that the sampling of the group switches will be a fairly high rate periodic program. Therefore, the delay in response should be minimized by proper programming.

The following is a list of the desired communication functions over the intergroup bus:

1. Executive group to scan groups for requests, one group to request to send words to another group (could be the executive group).
2. Executive group to scan groups for requests, one group to request words from another group (could be executive group).
3. Executive group to request to send words to another group directly.
4. Executive group to request words from another group directly.
5. Executive group to command another group switch to turn off power.

To accomplish the desired communications functions listed above various commands are required over the inter-cell bus and the intergroup bus. These commands are given in Table 10-3. It should be noted that the group switch contains an ID

(address register) just as a cell does. It will therefore respond to commands addressed to it just as a cell does. However, how it interprets a command may be different than how a cell does. Each command will use the same format as that given previously namely 3 bits for the X (command) code and 5 bits for the address.

Table 10-3. Inter-Cell Bus Group Switch Commands

Command	Code	Description
X_0	000	End of transmission command
X_2	010	Request "Executive Group Command" register
X_3	011	Send out a load "Executive Group Command" register command
X_4	100	Load "Group Request" register
X_5	101	Request "0" and "I-G Busy" registers
X_6	110	Initialize "Group Request" register
X_7	111	Extended command format
X_{7_1}	111	Send out a request "Group Request" register command
X_{7_2}	111	Set BITE TIMING FF
X_{7_3}	111	Send out a load "0" register command
X_{7_4}	111	Send out a power off command

Each of the commands will be explained below:

X_0 : This command signifies that the transmission is complete. It will reset the "I-G busy" flip-flop signifying that the inter-group bus is now free. It will also be sent out on the intergroup bus by the group switch. This will also reset the "I-G Busy" flip-flop in the other group switches. It should be noted that this is the only "global" command valid in a group switch (X_1 command codes are not sampled). The 5 bits for the address portion of this command are always all "0's."

X_2 : This command tells the group switch to send the data contained in the "Executive Group Command" register on to the inter-cell bus. This results in 4 bytes sent on the bus and the register being set to 0.

- X_3 : Reception of this command forces the group switch to send out a command (to be described below in Table 10-4) over the intergroup bus to load the "Executive Group Command" register in another group switch. The X_3 command received over the inter-cell bus will be followed by 5 data bytes. These bytes are comprised of Group₁ (5 bits), Cell (5 bits), Address (9 bits), Count (9 bits), I/O (1 bit), Group₂ (5 bits). These bytes are all sent out to the other group by the group switch. The group switch identifies which group they will be sent to by "Group₂".
- X_4 : This command tells the group switch to load the "Group Request" register with the next 4 data bytes on the inter-cell bus. The bytes are comprised of Group (5 bits), Cell (5 bits), Address (9 bits), Count (9 bits), and I/O (1 bit).
- X_5 : This command requests the group switch to send out a byte on the inter-cell bus. Two bits are of interest in the byte: (1) the "0" register (an output go command) and (2) the I-G Busy register (signifies if intergroup bus is tied up with a transmission).
- X_6 : This command will set the group request registers in the group switch to all "0's."
- X_7 : This command requires the reception of another byte and the subsequent decoding of the first two bits of that byte to determine what the actual command is. This then provides for 4 total commands from the X_7 command code.
- X_{7_1} : This command tells the group switch to output on the intergroup bus a command to another group switch requesting a "Group Request" register. The group switch will then let the response thru itself onto the inter-cell bus. The second byte contains: Code (2 bits), Group (5 bits) and Spare (1 bit). The "group" will be used to tell the group switch what group switch it should send out the intergroup bus command to.
- X_{7_2} : This command results in the BITE timing flip flop being set/reset. It simply uses the Code (2 bits) in the second byte for its identification. (The remaining 6 bits could be used as a check code by the group switch.)
- X_{7_3} : This command tells the group switch to send out on the intergroup bus a command to tell another group switch to set its "0" register to a 1. This signifies that the group can now output over the intergroup bus (the inputting group is ready). The second byte contains Code (2 bits), Group (5 bits), and 1 spare bit. The group bits identify what group switch to send the command to.

X_7 :
4

This command tells the group switch to output a command over the intergroup bus to tell another group switch to turn off power. The second byte contains: Code (2 bits), Group (5 bits) and 1 spare bit.

Table 10-4. Intergroup Bus Group Switch Commands

Command	Code	Description
X_0	000	End of transmission command
X_3	011	Load "Executive Group Command" register
X_4	100	Load "0" register
X_5	101	Turn off power
X_7	110	Send out "Group Request" register over the intergroup bus

These commands will be explained below: (the same command format is used: 3 bits for X and 5 bits for the group switch ID).

X_0 :

This command causes all I-G Busy flip-flops in the group switches to be reset.

X_3 :

This command tells the group switch to load its Executive Group Command register with the next 4 bytes on the intergroup bus. The bytes will contain: Group (5 bits), Cell (5 bits), Address (9 bits), count (9 bits) and I/O (1 bit).

X_4 :

This command tells the group switch to set its "0" register to a one. It is one byte long.

X_5 :

This command tells the group switch to turn off its power. It is one byte long.

X_7 :

This command tells the group switch to output its "Group Request" register over the intergroup bus.

Now that the commands have been presented, the mechanization of an intergroup communication operation may be discussed. The operation will be for the executive group (EG) to scan a group for a request and the group (G_1) to request words from another group (G_2). The complete operation will be described starting from when the requesting group (G_1) places its request in its group switch.

<u>Group</u>	<u>Inter-Cell Bus Command</u>	<u>Inter-Group Bus Command</u>	<u>Description</u>
G1	X ₄		The group switch of G1 is loaded with the request.
EG	X _{7₁}	X ₇	The group switch of EG fetches the request from the group switch of G1. This request will pass right onto the inter-cell bus of the EG where it will be examined.
EG	X ₃	X ₃	The EG, if it approves the request, will command the group switches of G1 and G2 to be loaded with an input and output command, respectively. Simultaneously, when the group switch of EG sends out an X ₃ intergroup bus command, it sets the I-G Busy flip flop.
EG	X ₃	X ₃	
G2	X ₂		G2 requests the executive command register contents to be sent on the inter-cell bus. Internally, G2 will send out the proper inter-cell bus commands to its cells to prepare to output (as described in Section 10.1 previously).
G2	X ₅		G2 will be continually sampling the "0" register in its group switch to determine if G1 has said it is ready to input. Then it will issue the necessary inter-cell bus command to its cells to tell it to start outputting. This results in a direct transmission path from a cell in G2 to a cell in G1.
G1	X ₂		G1 requests the executive command register contents to be sent over the inter-cell bus. Internally G1 will send out the necessary inter-cell bus commands to a cell to tell it to input.
G1	X _{7₃}	X ₄	G1 then immediately tells its group switch to tell the group switch in G2 to set its "0" register. The communications begin as soon as G2 has sampled the set "0" register as mentioned above.

<u>Group</u>	<u>Inter-Cell Bus Command</u>	<u>Inter-Group Bus Command</u>	<u>Description</u>
G2	X_0	X_0	The cell that was transmitting in G2 will send out as its last byte an X_0 , end of transmission command. This command will reset all the bus busy flip-flops that were set due to this transmission (I-C Busy in G1 and G2 and I-G Busy in group switch of EG).

The commands for the intermediate operations carried out by the controller cells in G1 and G2 have been left out for clarity. They are identical to that presented previously in Section 10.1 for communications among cells.

It should be noted that the other types of intergroup communications presented earlier in this section are mechanized quite similarly to the operation presented above and will therefore not be presented in detail.

10.3 MECHANIZATION OF INTER-CELL COMMUNICATION BUS COMMANDS AND CONTROL WORDS

A functional description of the commands and operations on the bus was given in the previous two sections. The control words that are required for the operations that will take place over the bus were introduced and the contents of each of the bytes comprising a control word were given. This section will present a detailed description of how the control words are formed or mechanized. It should be recalled that each of the operations was mechanized by one or more control words. These control words were identified by commands X_0 thru X_7 (recall that the first byte of a control word is identified as a command byte). Each of the control words may be identified by the command that it uses. Therefore, the term mechanization of the commands or control words will be used interchangeably below. The controller cell will be the only cell given the capability of mechanizing the commands or control words except for the X_0 , end of transmission command. (Each cell is identical, however, only the controller cell is capable of using certain logical functions in the cell.)

The commands for operation of the bus are generated by the controller cell when it executes a CC instruction. Referring back to Section 9, one can see the various codes used in the CC instruction to derive these commands. A description of how the commands are actually formed and sent out over the bus will be given below. The operations that take place in the controller cell during each clock time in the formation of these commands will be given. Note however that only the operations inherent in forming the commands are given. Other operations inherent in the processor's internal operation such as decoding the CC instructions, transferring the op code to the instruction register, etc. are not given. These were omitted to illustrate only the fundamental operations used in forming the commands.

A list of some abbreviations used throughout this section is given below:

CC:	The Controller Cell instruction to generate a Global Command described in Section 9 as op code 62.
SCR:	Shift Count Register
MB:	Memory Buffer Register
BCR:	Buffer Communications Register
IC Bus, Bus:	The Inter Cell Communication Bus
P:	Program Counter
U, A:	Upper Hardware Accumulator
L:	Lower Accumulator

10.3.1 X₀ and X₁ Commands

X₀ is generated based on the following conditions in the CC instruction:

C field = 01x

and L field = 000

note: x means this bit can be a 1 or 0

X₁ is generated based on the following conditions in the CC instruction:

C field = 01x

and L field = 001

Clock Time

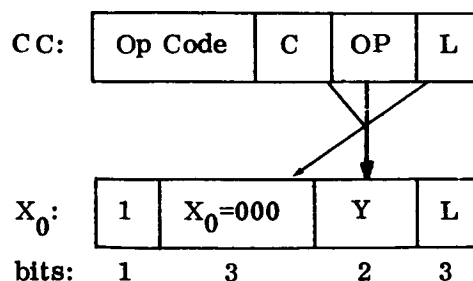
Operations

1

a. CC instruction → MB

2

b. selected bits from MB → BCR



Clock Time

Operations

L is transferred directly

Y is formed directly from OP and the first bit of C

X_1 : identical to X_0 except L will now contain 001, $X_1 = 001$

3

c. BCR is sent out on the IC bus

10.3.2 X_2, X_5, X_6 Commands

X_2, X_5 , and X_6 are generated based on the following conditions in the CC instruction:

C field = 01x

and L field = (010 + 101 + 110)

Clock Time

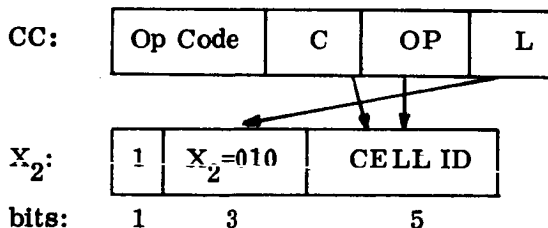
Operations

1

a. CC instruction $\rightarrow$ MB

2

b. selected bits from MB $\rightarrow$ BCR



X_2 is formed from L

Cell ID is formed from OP and the last bit of C

X_5 : Same as X_2 except L will now be 101

X_6 : Same as X_2 except L will now be 110

3

c. BCR is sent out on the IC Bus

Generation of an X_2 command will also result in the Cell ID being stored in the Shift Count Register. The reason for this will be explained in Section 10.4.

10.3.3 X_3, X_4 Commands

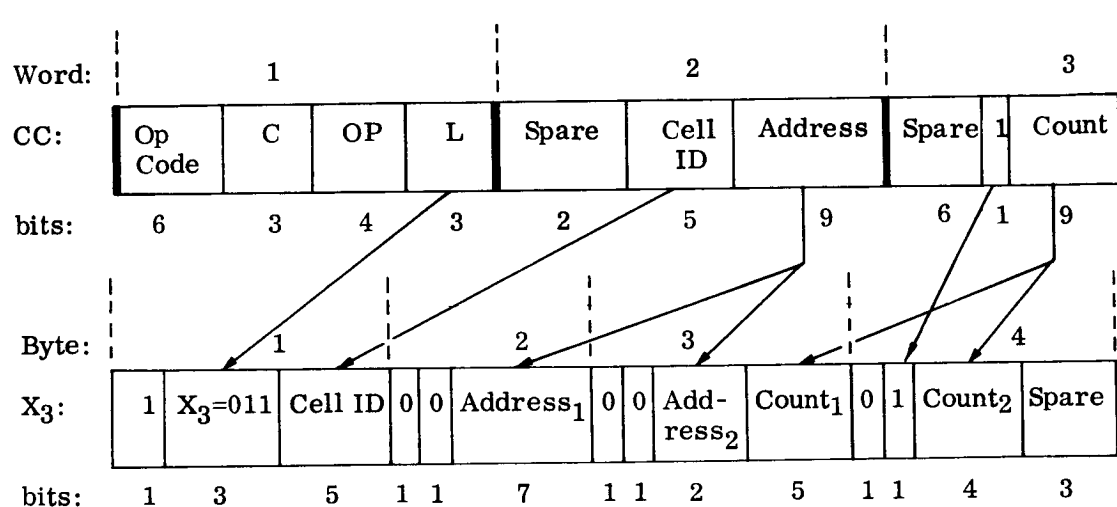
X_3 and X_4 commands are generated based on the following conditions in the CC instruction:

C field = 010

and L field = (011 + 100)

The CC instruction uses an extended format to mechanize these commands. Below is a diagram showing how the bits in the control words using commands X_3 and X_4 are formed from the CC instruction words.

Condition 1 : 1st 2 bits of OP = 00



X_4 is formed the same way except L will now contain $X_4 = 100$

The extended format (3 words) for the CC instruction is called for whenever:

C field = 01x

and L field = (011 + 100)

and OP field = 00xx

Clock Time

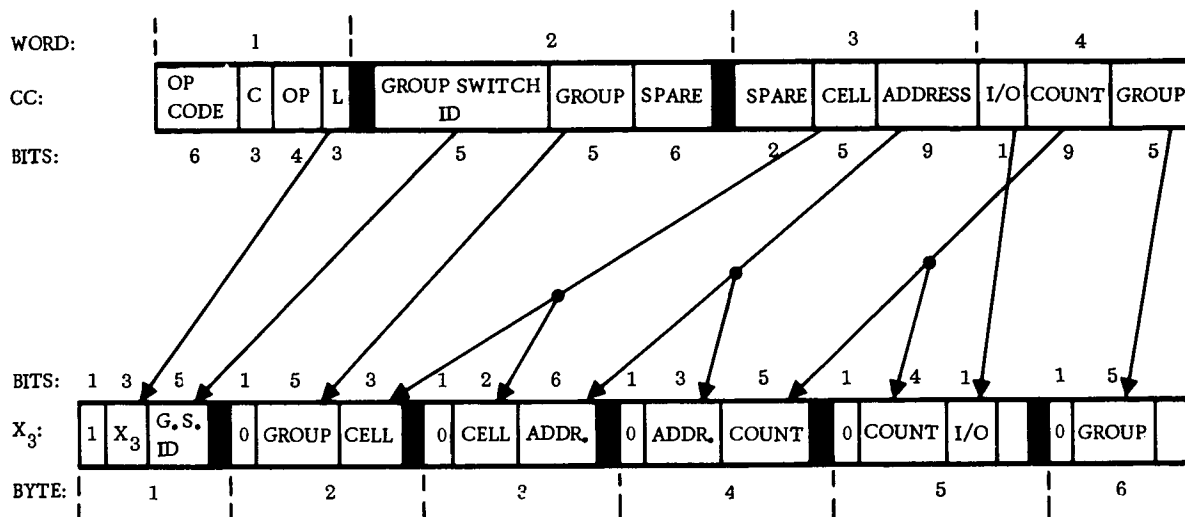
Operations

1	a. CC instruction Word 1 $\rightarrow$ MB
2	b. L from CC ₁ $\rightarrow$ BCR
	c. $P + 1 \rightarrow P$
3	d. $P \rightarrow$ Memory (read)

<u>Clock Time</u>	<u>Operations</u>
4	e. CC instruction Word 2 $\rightarrow$ MB f. $P + 1 \rightarrow P$
5	g. Cell ID from $CC_2 \rightarrow$ BCR h. $BCR \rightarrow$ IC Bus i. $Address_2$ from $CC_2 \rightarrow$ Saved
6	j. $Address_1$ from $CC_2 \rightarrow$ BCR k. $BCR \rightarrow$ IC Bus l. $P \rightarrow$ Memory (read)
7	m. $CC_3 \rightarrow$ MB n. $P + 1 \rightarrow P$
8	o. $Address_2 \rightarrow$ BCR p. $Count_1$ from $CC_3 \rightarrow$ BCR q. $BCR \rightarrow$ IC Bus r. $P \rightarrow$ Memory (read) s. $Count_2$ from $CC_3 \rightarrow$ saved
9	t. $Count_2 \rightarrow$ BCR u. Next instruction $\rightarrow$ MB
12	v. $BCR \rightarrow$ IC BUS

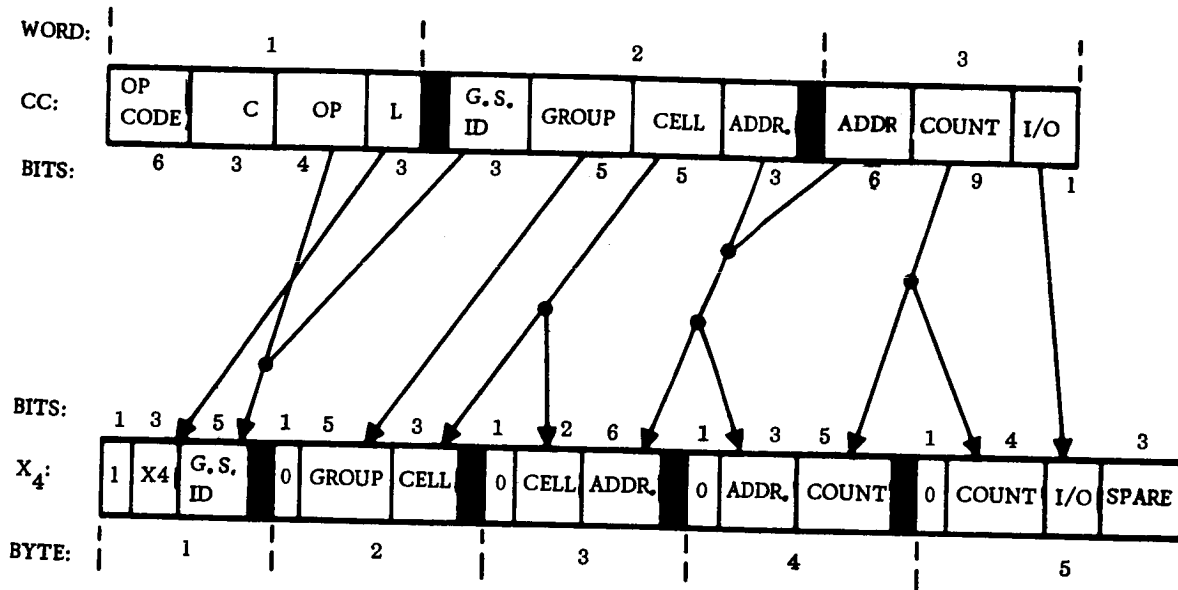
Note in the above timing description that the last byte on the bus is sent at clock time 12 instead of at clock time 9 when it is ready. The reason for this is in easing the mechanization within the receiving cell as will be pointed out in Section 10.4.

CONDITION 2: 1ST 2 BITS OF OP = 01



The above condition results in 4 words being called for in the extended CC format. This condition is used when an X₃ inter-cell bus group switch command is to be generated. The timing is somewhat similar to that given above except slightly larger due to more bytes being outputted on the bus.

CONDITION 3: 1ST 2 BITS OF OP = 10



As in condition 2 this X₄ is generated for an inter-cell bus group switch command. However, it only requires 3 CC words for its mechanization.

An X₃ command can also be generated under another condition:

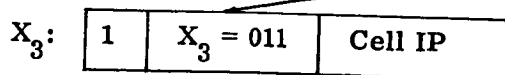
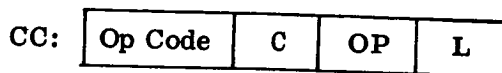
C field = 011

and L field = 011

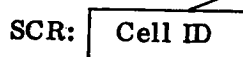
Clock Time

Operations

- 1 a. CC instruction → MB
- 2 b. Selected bits from MB → BCR
SCR register → BCR



bits: 1 3 5



Clock TimeOperations

3

c. BCR → IC Bus

This X_3 command is simply one byte long and therefore does not use the extended CC instruction format for the previous X_3 command. It also requires that the shift count register (SCR) be loaded with the Cell ID prior to executing this CC instruction. Recall that execution of the CC instruction forming an X_2 command loads the SCR with the Cell ID.

10.3.4 X_7 Commands

X_7 commands use an extension format to derive many commands from the basic X_7 code. These commands will be generated based on the following conditions in the CC instruction:

C field = 010

and L field = 111

Condition 1

or

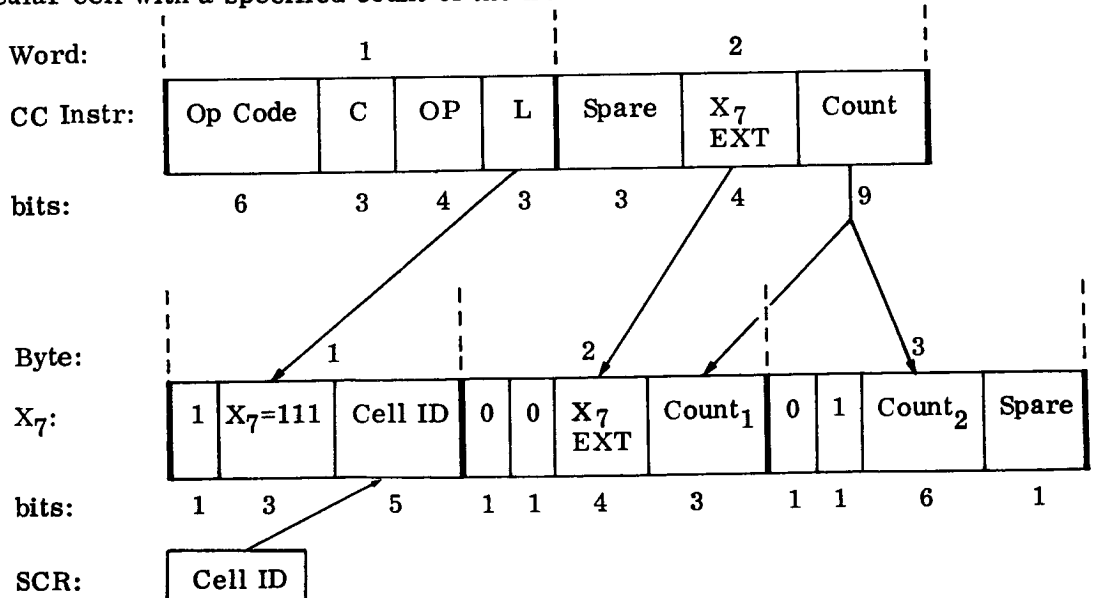
C field = 011

and L field = 111

Condition 2

An X_7 command will use an extended word CC format, in particular two words will be used to form the CC instruction.

Below is an illustration showing how the X_7 command is formed for the first condition. This particular X_7 extension illustrated is an output command to a particular cell with a specified count of the number of words to be output.

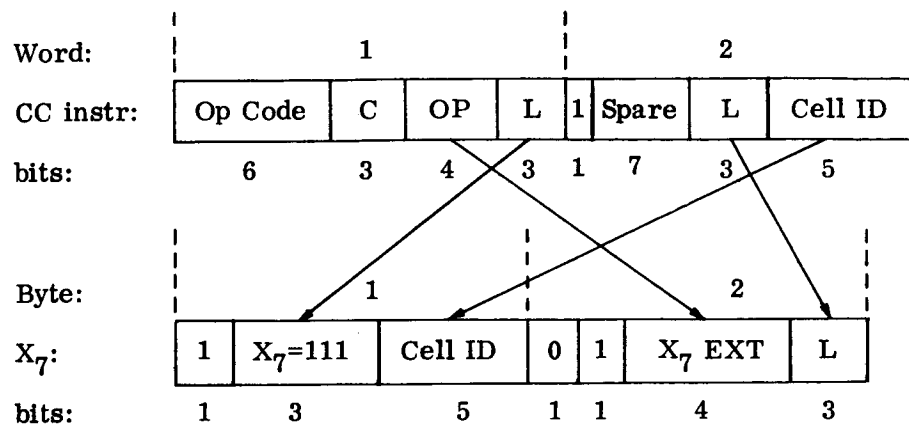


The X_7 extension command illustrated above requires 3 bytes to be sent out on the IC bus. The "Cell ID" is formed from the Shift Count Register. It is stored in this register by the prior formation of an X_2 command. The "count" is taken from the 2nd word of the CC instruction and used in the 2nd and 3rd bytes.

Timing for Condition 1

<u>Clock Time</u>	<u>Operation</u>
1	a. CC instruction word 1 $\rightarrow$ MB
2	b. L from $CC_1 \rightarrow$ BCR c. SCR register $\rightarrow$ BCR d. $P + 1 \rightarrow P$ (read)
3	e. $P \rightarrow$ Memory
4	f. CC instruction word 2 $\rightarrow$ MB
5	g. BCR $\rightarrow$ IC Bus h. X_7 EXT from $CC_2 \rightarrow$ BCR i. part of count from $CC_2 \rightarrow$ BCR j. $P + 1 \rightarrow P$ (read)
6	k. BCR $\rightarrow$ IC Bus l. part of count from $CC_2 \rightarrow$ BCR m. $P \rightarrow$ Memory n. Next instruction $\rightarrow$ MB o. BCR $\rightarrow$ IC Bus

Next an illustration of how the X_7 command is formed when the second condition occurs will be given below. This particular forming of the X_7 bytes is followed if the 1st bit of the 2nd CC word is a 1. If it is 0, a slightly different forming is followed for the 2nd byte. The first case is for X_7 commands to cells and the latter for X_7 commands to group switches.



The timing for condition 2 will be given below:

<u>Clock Time</u>	<u>Operation</u>
1	a. CC instruction Word 1 → MB
2	b. L from CC ₁ → BCR c. OP from CC ₁ → Saved d. P + 1 → P (read)
3	e. P → Memory
4	f. CC instruction Word 2 → MB
5	g. Cell ID → BCR h. BCR → IC Bus i. P + 1 → P
6	j. P → Memory (read) k. L from CC ₂ → BCR l. OP → BCR m. BCR → IC Bus
7	n. Next instruction → MB

10.4 MECHANIZATION OF INTER-CELL COMMUNICATION BUS OPERATIONS IN A CELL

This section will discuss the mechanization, within a cell, of the bus communication operations (the operations upon the generation or reception of X commands by a cell). These operations were presented in section 10.1 where a description of the control word bytes were given. A detailed description of the logical properties required in the cell to carry out the operations will be given below. Each of the operations will be presented in the same order as presented in section 10.1.

10.4.1 Controller Cell, C_0 , to Send Data to a Cell C_1

Two basic approaches were considered to mechanize this operation as well as all the others, namely software and hardware approaches. The approach selected was to use a hardware mechanization. The primary reason behind this choice is that software is very inefficient with respect to the time it takes to mechanize. For example, in this operation, the basic function to be performed once the command from the bus has been decoded, is to simply store a set of words in memory that arrive via the bus. These words may arrive at a rate up to 1 word every 2 clock cycles since the bus is a 1/2 word in parallel. However, a software routine would require a three instruction loop:

1. Test, Increment Index Register by 1, Jump on 0
2. STORE
3. JUMP

This loop would take at a minimum 7 memory cycles to execute. Assuming 2 clock cycles per memory cycle, this results in 14 clock cycles for each word to be stored. Therefore, this would result in very inefficient use of the bus.

The hardware approach will now be described. Reference should be made to Section 10.1 for this operation and the ones that follow for a description of the control word bytes. The mechanization will be described for the two cells, C_0 and C_1 . First the mechanization within the controller cell, C_0 , will be described. The controller cell will execute the following instructions:

1. Double Precision Load A and L
2. CC (X_3)
3. CC (Output)

The first instruction will load A and L with the address and count of the number of words to be sent out. Instruction 2 is the CC command to generate the format X_3 as described in Section 9 of this report. This particular CC instruction requires 3 words for its mechanization as discussed previously. It will result in the 4 X_3 command bytes being sent out over the bus. The last instruction is the CC command that tells the controller cell to start outputting data on the bus. This instruction simply exchanges the accumulator (previously loaded with the address) and the

program counter and sets the output mode flip-flop. The controller cell will now simply read data from its memory and output it over the bus. Each time the program counter is incremented, the count contained in L will be decremented by 1 and checked against 0. When the count equals 0 the output mode flip-flop will be reset and the accumulator will be transferred to the program counter.

A listing of the sequence of operations, starting from the time the first byte generated by the second instruction (X_3 commands) is output over the bus, is given below.

Operations in Controller Cell C_0

<u>Clock Time</u>	<u>Operation</u>
1	X_3 Byte 1 $\rightarrow$ Bus
2	X_3 Byte 2 $\rightarrow$ Bus
3	---
4	X_3 Byte 3 $\rightarrow$ Bus
5	Next instruction (CC Output) $\rightarrow$ MB
6	U (Accumulator) $\leftrightarrow$ P (Program Counter) P $\rightarrow$ Memory (read)
7	1st data word $\rightarrow$ MB L (Lower Accumulator) $- 1 \rightarrow$ L P + 1 $\rightarrow$ P
8	X_3 Byte 4 $\rightarrow$ Bus 1st word (1/2) $\rightarrow$ BCR (Buffer Communication Register) P $\rightarrow$ Memory (read)
9	BCR $\rightarrow$ Bus 1st Word (1/2) $\rightarrow$ BCR P + 1 $\rightarrow$ P L - 1 $\rightarrow$ L 2nd Word $\rightarrow$ MB

<u>Clock Time</u>	<u>Operation</u>
10	BCR $\rightarrow$ Bus 2nd Word (1/2) $\rightarrow$ BCR P $\rightarrow$ Memory (read)
11	BCR $\rightarrow$ Bus 2nd word (1/2) $\rightarrow$ BCR P + 1 $\rightarrow$ P L - 1 $\rightarrow$ L 3rd Word $\rightarrow$ MB
12	etc. until L = 0, then A $\rightarrow$ P

The mechanization within the receiving cell, C₁, will be discussed below. The first byte received by C₁ will contain the X₃ command code and the cell's, C₁, identification code. Cell, C₁, will recognize that the command is for itself and will decode the command and issue an interrupt. The bus communications will be handled on an interrupt basis with the interrupt halting the execution of the present instruction. This approach was taken since the cell could be in the process of executing a long instruction such as multiply or divide and it would be inefficient to tie the bus up waiting for instructions to be completed.

The interrupt will result in the program counter, accumulator, memory buffer and possibly some control flip-flops being stored in fixed locations in memory. Basically the functions that will occur in mechanizing the bus operation are:

Store registers on interrupt

Place address in P

Place count in A

Assemble bytes from the bus in MB

Use P as memory address to store MB

Decrement A by 1, test for 0

Increment P by 1

The sequence of operations that will occur in the cell, starting from the time the first byte is received over the bus, are listed below. It should be kept in mind that the buffer communication register (BCR) will be mechanized with master slave type of flip-flops. Therefore it is possible to load byte i + 1 into BCR while byte i is being read from it in the same clock time.

Operations in Receiving Cell, C₁

	<u>Clock</u>	<u>Operation</u>
4 Command Bytes Recv'd By The Cell	1	Byte 1 → BCR, Decoded, Interrupt
	2	Byte 2 → BCR, Processor Stopped, Fixed address sent to memory (write)
	3	P → Memory (P is stored)
	4	BCR → part of P, Byte 3 → BCR. Fixed address sent to memory (write)
	5	Part of BCR → part of P, MB → Memory
	6	Fixed address sent to memory (write)
	7	A → Memory (A is stored)
	8	Part of BCR → A
Data Recv'd		Byte 4 → BCR
	9	BCR → A
		Data Byte 1 → BCR
	10	BCR → MB, Data byte 2 → BCR, P (Address) → Memory (write)
	11	BCR → MB, MB → Memory, Data byte 3 → BCR, A - 1 → A, P + 1 → P
	12	BCR → MB, Data byte 4 → BCR, P → Memory
	13	BCR → MB, MB → Memory, Data byte 5 → BCR, A - 1 → A, P + 1 → P, etc. until A = 0 then reload stored registers and start processor

As noted the above operations are all performed solely by hardware. Referring back to Section 10.1, it can be seen that the 1st line in the byte from the bus is used for control purposes. Following the first command byte (identified by the C (control line) being a 'one'), the first line in the byte will be a '0' and will be a 'one' only when the command sequence is complete and the next bytes on the bus represent data. Therefore this 0 to 1 transition will be used to set the communication bus control logic to accept information on the bus as data. The sequence of operations described above will then be clocked based on the occurrence of data bytes received via the bus.

It may be noted above that command byte 4 is received at clock 8. If this command byte were received earlier extra hardware would be required to store information from the command bytes. This is the case since MB cannot be used until A has been stored. Therefore MB and A would both have to be stored before information could be stored in them. This is why P is stored first and used immediately to store information (address) from the command bytes. As can be seen from the mechanization within the controller cell, C_0 , the fastest it can output the first data byte is at clock 9. Therefore, there is no penalty in requiring it to output command byte 4 at clock 8; in fact this simplifies the mechanization in cell, C_1 .

10.4.2 Controller Cell, C_0 , to Request Data From C_1

The mechanization within the controller cell will be described first. The controller cell will execute the following instructions:

1. Double Precision Load A and L
2. CC (X_4)
3. CC (Input)

The operations are quite similar to that described for the previous operation, 10.4.1; A and L are loaded with the address and count of the words that will be read out of the controller cell's memory. Instruction 2 is a CC instruction that generates 4 X_4 bytes to be output over the bus as described in Section 10.3.3. Finally the last instruction is a CC instruction that sets the input mode flip-flop and exchanges A and P. The controller cell is now ready to input words from the bus. Each time a word is stored in memory, the program counter is incremented by 1 and L decremented by 1 and checked against 0. When the count is 0 the contents of A are transferred back to P and the controller cell proceeds with its program. It should be noted that the cell will contain a simple one shot type of timer that is set whenever the CC (input) instruction is executed. The purpose of this timer is to allow a certain maximum time for a cell to respond with data. If the cell fails to respond within the maximum time interval an interrupt will be issued that will be processed by a software routine in the controller cell (generally this will indicate a failed cell, C_1). In addition, the controller cell contains another simple one-shot type of timer that times the time interval between words on the bus. If the time interval exceeds that allowed by the timer, an interrupt is issued. This prevents the cell C_1 from transmitting only a part of the words and hanging up the controller cell by having it keep on waiting for the remainder of the words. Of course, once the count in L is zero this timer is inhibited.

A listing of the sequence of operations, starting from the time the first byte generated by the second instruction (X_4) is output over the bus is given below.

Operations in Controller Cell C₀

<u>Clock</u>	<u>Operation</u>
1	X ₄ Byte 1 → Bus
2	X ₄ Byte 2 → Bus
3	---
4	X ₄ Byte 3 → Bus
5	Next instruction (CC Input) → MB
6	A ↔ P Timer Set
7	---
8	X ₄ Byte 4 → Bus
9	---
10	---
11	Word 1 ₁ → BCR
12	P → Memory (write) BCR → MB Word 1 ₂ → BCR
13	BCR → MB MB → Memory (Word 1 stored) Word 2 ₁ → BCR A - 1 → A P + 1 → P
14	BCR → MB Word 2 ₂ → BCR P → Memory (write)

Clock

15

OperationBCR $\longrightarrow$ MBWord 3₁ $\longrightarrow$ BCRMB $\longrightarrow$ Memory (Word 2 stored)A - 1 $\longrightarrow$ AP + 1 $\longrightarrow$ P

The mechanization within the sending cell, C₁, will be discussed below. The first byte received by C₁ will contain the X₄ command code and the cell's, C₁, identification code. Cell, C₁, will recognize that the command is for itself and will decode the command and issue an interrupt. The same process as described for the previous operation, 10.4.1, will be followed on the interrupt. The primary difference in the mechanization given below is that words will be readout of C₁'s memory instead of into it as in 10.4.1.

Operations in Sending Cell, C₁ClockOperation

4
Command
Bytes
Rcv'd
by C₁

1

Byte 1 $\longrightarrow$ BCR, Decoded, Interrupt

2

Byte 2 $\longrightarrow$ BCR, Processor Stopped,
Fixed Address $\longrightarrow$ Memory (write)

3

P $\longrightarrow$ Memory (P is stored)

4

BCR $\longrightarrow$ Part of PByte 3 $\longrightarrow$ BCRFixed Address $\longrightarrow$ Memory (write)

5

Part of BCR $\longrightarrow$ part of P
MB $\longrightarrow$ Memory

6

Fixed Address $\longrightarrow$ Memory (write)

7

A $\longrightarrow$ Memory (A is stored)

8

Part of BCR $\longrightarrow$ A
Byte 4 $\longrightarrow$ BCR
P $\longrightarrow$ Memory (read)

	<u>Clock</u>	<u>Operation</u>
	9	BCR $\longrightarrow$ A Word 1 $\longrightarrow$ MB P + 1 $\longrightarrow$ P
	10	Word 1 ₁ $\longrightarrow$ BCR P $\longrightarrow$ Memory (read) A - 1 $\longrightarrow$ A
Data on the bus	11	BCR $\longrightarrow$ Bus Word 1 ₂ $\longrightarrow$ BCR Word 2 $\longrightarrow$ MB P + 1 $\longrightarrow$ P
	12	Etc. until A = 0, Then restart Processor, Reload P, A, and MB registers

10.4.3 Controller Cell, C₀, to Scan a Cell for Requests - Cell, C₁, to Request Words from Cell, C₂

This operation is initiated by the controller cell executing a CC (X₂) instruction. This X₂ command forces cell C₁ to respond with a request word. Upon executing the CC (X₂) instruction the controller cell automatically enters the idle state. It will also set a simple one shot timer (previously introduced in Section 10.4.2) so that the response from cell C₁ is received within some maximum time limit. If it is not received within this time limit an interrupt will occur as previously described.

The response word will be placed in the A and L registers under hardware control and once this is accomplished the controller cell will be released from the idle state and allowed to execute its next instruction. This instruction is the start of a software routine to process the request response word from the Cell C₁. This routine will check what request is being made if any, Input or Output, what cell and location communication is desired with and the count of the number of words to be communicated. It will check for valid requests and may possibly reduce the amount of words desired to be communicated. At the end of the software routine the proper CC instructions will be loaded with the necessary information so as to initiate the requested communications.

In the operation under consideration here Cell C₁ is requesting to input words. It will therefore receive an X₃ (input) command from the controller cell if its requested communications will be honored. If its request is not honored, the controller cell will send it an X₇ command that signifies request not honored. This X₇ command results in an interrupt in cell C₁ that forces it to enter its communication executive routine. Now once cell C₁ has been told to input, it is ready to accept data. However, the number of words it receives may be different from the number it requested. This situation is handled by the simple one-shot timer presented in Section 10.4.2. This timer allows a maximum time between words received over the bus. If this time is exceeded, an interrupt occurs that sends cell C₁ to a software

routine that will note the count of the number of words remaining to be received indicated in the accumulator. This routine will use this information to update its communication bus executive tables. If the count requested is received, the accumulator will count down to zero and this will restart the processor section of cell C_1 so that it may complete its interrupted instruction.

For further details on the software routines discussed above, both in the controller cell and working cell, C_1 , the reader is referred to Section 11 of this report. In addition this section will show how these routines tie in with the other executive functions within a cell. A detailed mechanization of the hardware aspects of the operation will be given below.

Operations in Controller Cell, C_0

<u>Clock</u>	<u>Operation</u>
1	CC (X_2) instruction $\longrightarrow$ MB
2	X_2 Command $\longrightarrow$ BCR Cell ID $\longrightarrow$ SCR One-Shot Timer ₁ $\longrightarrow$ Set
3	BCR $\longrightarrow$ Bus
4-8	Idle
9	Response Byte 1 $\longrightarrow$ BCR $P + 1 \longrightarrow P$
10	BCR $\longrightarrow$ A Response Byte 2 $\longrightarrow$ BCR $P \longrightarrow$ Memory (read)
11	BCR $\longrightarrow$ A Response Byte 3 $\longrightarrow$ BCR Next Instruction $\longrightarrow$ MB
12	BCR $\longrightarrow$ L Response Byte 4 $\longrightarrow$ BCR (For details on next instruction and the remainder of the software routine to process the response word see section 11.2.) (It is assumed that this routine will take j clock times to complete)
13	BCR $\longrightarrow$ L
$j + 10$	CC (X_3) instruction $\longrightarrow$ MB
$j + 11$	X_3 Command $\longrightarrow$ BCR $P + 1 \longrightarrow P$

<u>Clock</u>	<u>Operation</u>
j + 12	BCR $\longrightarrow$ Bus P $\longrightarrow$ Memory (read)
j + 13	CC (X_4) instruction $\longrightarrow$ MB
j + 17	X_4 Byte 1 $\longrightarrow$ Bus
j + 18	X_4 Byte 2 $\longrightarrow$ Bus
j + 20	X_4 Byte 3 $\longrightarrow$ Bus
j + 24	X_4 Byte 4 $\longrightarrow$ Bus
	(See Section 10.3.3 for details on how these bytes are formed)

Operations in Requesting Cell, C_1

<u>Clock</u>	<u>Operation</u>
3	X_2 Command $\longrightarrow$ BCR, Decoded, Interrupt
4	Fixed Address $\longrightarrow$ Memory (write)
5	MB $\longrightarrow$ Memory (stored)
6	Fixed Address $\longrightarrow$ Memory (read)
7	Response Word 1 $\longrightarrow$ MB
8	Response Word 1 ₁ $\longrightarrow$ BCR Fixed Address $\longrightarrow$ Memory (read)
9	BCR $\longrightarrow$ Bus Response Word 1 ₂ $\longrightarrow$ BCR Response Word 2 $\longrightarrow$ MB
10	BCR $\longrightarrow$ Bus Response Word 2 ₁ $\longrightarrow$ BCR
11	BCR $\longrightarrow$ Bus * Response Word 2 ₂ $\longrightarrow$ BCR
12	BCR $\longrightarrow$ Bus
13	9 bit count from Response Word 2 saved in BCR Fixed Address $\longrightarrow$ Memory (write)

<u>Clock</u>		<u>Operation</u>
14		$A \rightarrow \text{Memory}$
15		Fixed Address $\rightarrow$ Memory (write) BCR (Count) $\rightarrow A$
16		$P \rightarrow \text{Memory (stored)}$
17		Fixed Address $\rightarrow$ Memory (read)
18		Address where to store data $\rightarrow MB$
19		$MB \rightarrow P$
20		idle until command from C_0
$j + 12$		X_3 Command $\rightarrow$ BCR, Decoded, Logic set to input over the bus
DATA RCV'D From C_2	$j + 28$	Word $1_1 \rightarrow \text{BCR}$
	$j + 29$	$\text{BCR} \rightarrow MB$ Word $1_2 \rightarrow \text{BCR}$ $P \rightarrow \text{Memory (write)}$
	$j + 30$	$\text{BCR} \rightarrow MB$ $MB \rightarrow \text{Memory (Word 1 stored)}$ Word $2_1 \rightarrow \text{BCR}$ $A - 1 \rightarrow A$ $P + 1 \rightarrow P$
	$j + 31$	$\text{BCR} \rightarrow MB$ Word $2_2 \rightarrow \text{BCR}$ $P \rightarrow \text{Memory (write)}$
	$j + 32$	$\text{BCR} \rightarrow MB$ $MB \rightarrow \text{Memory (Word 2 stored)}$ Word $3_1 \rightarrow \text{BCR}$ $A - 1 \rightarrow A$ $P + 1 \rightarrow P$
	$j + 33$	ETC. Until $A = 0$ or one-shot timer that times interval between words exceeds some maximum time, then if $A \neq 0$, interrupt.

It should be noted that the logic to input data in C_1 is not enabled until the last X_4 byte to C_2 has been sent out by C_0 since this byte has a '1' on line 1. (Recall that this logical property is required to enable the bus communication logic for X_3 , X_4 and X_7 commands.)

Operations in Sending Cell C_2

<u>Clock</u>	<u>Operation</u>
(See 10.4.2 for these operations, since they are identical)	
$j + 28$	Word $1_1 \longrightarrow$ Bus
$j + 29$	Word $1_2 \longrightarrow$ Bus
	ETC.

10.4.4 Controller Cell, C_0 , to Scan a Cell for Requests - Cell, C_1 , to Request to Send Words to Cell, C_2

This operation is very similar to that described above in 10.4.3. The primary difference being that upon examining the response word from C_1 , the controller cell will send out different X commands. Cell C_1 will of course be following a procedure for outputting data; while cell C_2 will follow the procedure presented in 10.4.1 to input data.

Operations in Controller Cell

<u>Clock</u>	<u>Operation</u>
1	CC (X_2) instruction $\longrightarrow$ MB
2	X_2 Command $\longrightarrow$ BCR Cell ID $\longrightarrow$ SCR One-Shot Timer $_1$ $\longrightarrow$ Set
3	BCR $\longrightarrow$ Bus
4-8	Idle
9	Response Byte 1 $\longrightarrow$ BCR $P + 1 \longrightarrow P$
10	BCR $\longrightarrow$ A Response Byte 2 $\longrightarrow$ BCR $P \longrightarrow$ Memory (read)
11	BCR $\longrightarrow$ A Response Byte 3 $\longrightarrow$ BCR Next Instruction $\longrightarrow$ MB
12	BCR $\longrightarrow$ L Response Byte 4 $\longrightarrow$ BCR (For details on next instruction and the remainder of the software routine to process the response word see Section 11.2. It is

ClockOperation

assumed that this routine will take k clock times to complete.) (This routine will, of course, branch differently than that in the previous operation 10.4.3, since the response word now contains an output request.)

13	BCR $\longrightarrow$ L
$k + 10$	CC (X_3) instruction $\longrightarrow$ MB
$k + 14$	X_3 Byte 1 $\longrightarrow$ Bus
$k + 15$	X_3 Byte 2 $\longrightarrow$ Bus
$k + 17$	X_3 Byte 3 $\longrightarrow$ Bus
$k + 18$	CC (X_7) instruction $\longrightarrow$ MB
$k + 21$	X_3 Byte 4 $\longrightarrow$ Bus (see section 10.3.3 for detailed mechanization of the CC (X_3) instructions)
$k + 22$	X_7 Byte 1 $\longrightarrow$ Bus
$k + 23$	X_7 Byte 2 $\longrightarrow$ Bus
$k + 24$	X_7 Byte 3 $\longrightarrow$ Bus

Operations in requesting cell C_1 ClockOperation

3	The operations for clock times 3-20 are identical to those given in 10.4.3 except for clock time 13, the count is not saved in BCR and clock time 15, BCR is not transferred to A
-	
-	
-	
20	
$k + 22$	X_7 Byte 1 $\longrightarrow$ BCR, Decoded
$k + 23$	X_7 Byte 2 $\longrightarrow$ BCR, Output Operation Decoded
$k + 24$	Part of BCR $\longrightarrow$ A and MB X_7 Byte 3 $\longrightarrow$ BCR Fixed Address $\longrightarrow$ Memory (write)

<u>Clock</u>	<u>Operation</u>
$k + 25$	Part of BCR $\longrightarrow$ A and MB MB $\longrightarrow$ Memory (count is stored)
$k + 26$	P $\longrightarrow$ Memory (read)
$k + 27$	Word 1 $\longrightarrow$ MB P + 1 $\longrightarrow$ P
$k + 28$	Word 1 ₁ $\longrightarrow$ BCR P $\longrightarrow$ Memory (read) A - 1 $\longrightarrow$ A
$k + 29$	BCR $\longrightarrow$ Bus (1st data byte out) Word 1 ₂ $\longrightarrow$ BCR Word 2 $\longrightarrow$ MB P + 1 $\longrightarrow$ P
$k + 30$	ETC. until A = 0, then complete interrupted instruction

Operations in Receiving Cell C_2

<u>Clock</u>	<u>Operation</u>
$k + 14$	X_3 Command $\longrightarrow$ BCR, Decoded, Interrupt
$k + 15$	X_3 Byte 2 $\longrightarrow$ BCR
	ETC. (the same procedure is followed as in 10.4.1) at clock time $k + 22$ the cell is ready to accept data, however this first data byte will be received at clock time $k + 29$

10.4.5 Controller Cell to Tell One Cell, C_1 , to Output to Another Cell, C_2

This operation is simply a combination of the X_3 and X_4 commands described in 10.4.1 and 10.4.2. Therefore the description here will be minimal.

Operations in Controller Cell

<u>Clock</u>	<u>Operation</u>
1	CC (X_3) instruction $\longrightarrow$ MB
5	X_3 Byte 1 $\longrightarrow$ Bus
6	X_3 Byte 2 $\longrightarrow$ Bus

<u>Clock</u>	<u>Operation</u>
8	X_3 Byte 3 $\longrightarrow$ Bus
9	CC (X_4) instruction $\longrightarrow$ MB
12	X_3 Byte 4 $\longrightarrow$ Bus
13	X_4 Byte 1 $\longrightarrow$ Bus
14	X_4 Byte 2 $\longrightarrow$ Bus
16	X_4 Byte 3 $\longrightarrow$ Bus
20	X_4 Byte 4 $\longrightarrow$ Bus

Operations in Receiving Cell C_2

<u>Clock</u>	<u>Operation</u>
5	$X_3 \longrightarrow$ BCR, Decoded, Interrupt
-	
-	
-	
13	Ready to Accept Data

Operations in Sending Cell C_1

<u>Clock</u>	<u>Operation</u>
13	$X_4 \longrightarrow$ BCR, Decoded, Interrupt
-	
-	
-	
23	1st Data Byte on the Bus

It should be recalled here that the receiving cell C_2 uses the 0 to 1 transition on line 1 in the 4th command byte of X_4 to enable its data inputting circuitry.

10.4.6 Controller Cell to Tell One Cell, C_1 to Input from Another Cell, C_2

This operation is mechanized identically to that above.

10.4.7 Controller Cell to Tell Another Cell, C_1 to Reconfigure Some Internal Control State

Case 1: CC (X_6)

Operations in Controller Cell

<u>Clock</u>	<u>Operation</u>
1	CC (X_6) instruction $\longrightarrow$ MB
2	$X_6 \longrightarrow$ BCR
3	BCR $\longrightarrow$ Bus

Operations in Cell C_1

<u>Clock</u>	<u>Operation</u>
3	$X_6 \longrightarrow$ BCR, Decoded, Interrupt
4	Change Control State

Case 2: CC (X_7):

Operations in Controller Cell

<u>Clock</u>	<u>Operation</u>
1	CC (X_7) Instruction $\longrightarrow$ MB
5	X_7 Byte 1 $\longrightarrow$ Bus
7	X_7 Byte 2 $\longrightarrow$ Bus

Operations in Cell C_1

<u>Clock</u>	<u>Operation</u>
5	$X_7 \longrightarrow$ BCR, Decoded, Interrupt
7	X_7 Byte 2 $\longrightarrow$ BCR, Decoded
8	Change Control State

10.4.8 Controller Cell to Request Status Word From Cell C₁

Operations in Controller Cell

<u>Clock</u>	<u>Operation</u>
1	CC (X ₅) instruction → MB
2	X ₅ → BCR One Shot Timer ₁ → Set
3	BCR → Bus
4-7	IDLE
8	Status Word → BCR (A software routine will process the Status Word)

Operations in Cell C₁

<u>Clock</u>	<u>Operation</u>
3	X ₅ → BCR, Decoded, Interrupt
4	Fixed Address → Memory (write)
5	MB → Memory (Saved)
6	Fixed Address → Memory (read)
7	Status Word → MB
8	MB → BCR BCR → Bus

It should be noted here that the one-shot timer introduced in Section 10.4.3 is again used here to prevent a no response condition from hanging up the controller cell.

10.4.9 Global Operations - Format GC Instructions

The previous described operations were all local operations. The operations that follow will now be classified as Global. The first of these to be described is the Format GC Instruction.

Operations in Controller Cell

<u>Clock</u>	<u>Operation</u>
1	CC (X_0) Instruction $\longrightarrow$ MB
2	$X_0 \longrightarrow$ BCR
3	BCR $\longrightarrow$ Bus

Operations in the Cells

<u>Clock</u>	<u>Operation</u>
3	a) $X_0 \longrightarrow$ BCR b) if $Y = 11$ and cell $\neq$ dependent global state, then ignore the command c) X_0 decoded and proper GC Format Signal generated and sent to processor control logic

10.4.10 Level Control GC Instructions (X_0)

Operations in Controller Cell

<u>Clock</u>	<u>Operation</u>
1	CC (X_0) Instruction $\longrightarrow$ MB
2	$X_0 \longrightarrow$ BCR
3	BCR $\longrightarrow$ Bus

Operations in the Cells

<u>Clock</u>	<u>Operation</u>
3	a) $X_0 \longrightarrow$ BCR b) if $Y = 01 + 10$, and cell = any dependent state, and L = Level Register in the cell; then decode the command c) X_0 decoded and change of state issued to the cell

10.4.11 Level Control GC Instructions (X_1)

Same as 10.4.10 except an X_1 code is used and Y does not have to be checked.

10.4.12 Individual Cell GC Instructions (X_7)

Operations in Controller Cell

<u>Clock</u>	<u>Operation</u>
1	CC (X_7) Instruction Word 1 $\longrightarrow$ MB
5	X_7 Byte 1 $\longrightarrow$ Bus
7	X_7 Byte 2 $\longrightarrow$ Bus
	(See Section 10.3.4 for details on X_7 mechanization)

Operations in Receiving Cell C_1

<u>Clock</u>	<u>Operation</u>
5	$X_7 \longrightarrow$ BCR, Decoded
7	X_7 Byte 2 $\longrightarrow$ BCR
	X_7 EXT is decoded and the proper change of state is performed (see 10.1.2 for these operations); if indicated by X_7 EXT, L is used to set the level register

It should be noted in the above operation that the X_7 is not decoded as a GC command until X_7 EXT is received.

The following procedures are used to change the states. From the independent state to the dependent local state the following is done. The program counter is saved in the same location as used for interrupts and the present instruction completed, then the hardware accumulator is stored in its memory location. The next instruction will then be taken from a fixed location in memory. From the independent to the dependent wait or global state the same is done except that no next instruction is fetched. It should be noted that it will be the responsibility of the program that follows to save the necessary accumulators and index/bank registers in this cell; essentially this may be considered an interrupting program.

To go from a dependent state to the independent state, the program counter is restored from the same location that it is automatically stored in above and the cell is restarted independently.

To enter the controller cell state, a fixed location will be used to fetch the next instruction. If the cell was in an independent state or dependent local state prior to this, the program counter and hardware accumulator would be saved as done above and the present instruction completed.

10.4.13 End of Transmission (X_0)

This command is generated by the bus I/O control hardware in a cell upon the completion of an X_4 output command. It is the only command that can be sent from a cell that is not a controller cell. The command simply consists of the control line going to the one state and all 0's on the 8 byte data lines. It is thus very simple to generate in the control section. This command will result in the IC Busy register (Inter-Cell bus busy FF) being reset.

11. EXECUTIVE DESIGN

11.1 INTRODUCTION

The distributed processor has many levels of control as described previously. This results in many levels of control (executive action) in the software to program the computer. The executive programs may be considered at three levels: (1) the system executive, (2) the group executive, and (3) the cell executive.

The group executive will be discussed first in this section. Basically this executive controls a group of cells. Each group must have allocated one cell as a controller cell; this cell will act as the group executive. Since each cell in a group is identical any cell can perform as a controller cell. The group executive functions will reside in one or more cells in a group (more than one as required due to storage limitations of an individual cell) and one of these cells will be acting as a controller cell at any one time. Although the different programs that comprise the group executive may be in different cells, this set of cells (which always includes the present cell that is in the controller cell state) is called the group executive.

The cell executive is that program that resides in an individual cell and controls its performance. As may be expected, this program is considerably simpler and smaller than the group executive. Each cell will have its own executive and in general they will all be different although containing many basic similarities.

One of the group executives will have in addition to the group executive functions, functions that control the operation of the overall computer system. These functions will be called the system executive and this group will be said to contain the system executive.

11.2 GROUP EXECUTIVE

The group executive has many functions. Basically, it must control the operation of the cells in the group and the inter-cell communication bus, it must reconfigure the group resources due to changes in processing requirements and due to failures, and finally it must interface with other groups. The functions may be broken down into six subfunctions:

1. Control system resources
2. Furnish global programs to dependent cells
3. Allocate system resources
4. Test system hardware and software, and respond to any malfunctions

5. Reconfigure system upon

- a. Normal phase change
- b. Malfunction

6. Interface with other groups, including the system executive

Each of these subfunctions will be discussed below.

11.2.1 Control System Resources

The group executive has many more resources to control than the usual executive of traditional computer systems. In some ways this job is easier in the distributed computer system, in some ways more difficult. This section of the group executive is concerned with the controller cell interface with the operational cells as the cells require services of the executive. No malfunctions are considered here, nor is reconfiguration, these problems are discussed in Sections 8 and 11.2.4.

The computational tasks will be performed in the operational cells. These cells may require executive services such as getting I/O data over the inter-cell bus, getting intermediate computational data from other operational cells, setting clocks, etc. The group executive will provide such services under the subfunction of "control system resources." These services will be provided in two modes: periodic and background. Periodic services are those required at fixed predetermined intervals of time while background services are those fitted in between the periodic according to some priority scheme. Each of these two modes of service will be described below. The services will also be provided in two ways: by command or request. Command services are those completely controlled by the controller cell, e.g. the controller cell commanding one cell to send another cell certain data. Request services are those requested by the operational cells, e.g. an operational cell requesting data from another cell.

It should be noted here that the group executive function of control system resources is actually controlled or formed by another group executive function namely that of "allocate system resources" which will be discussed later (section 11.2.3). The function of controlling the system resources may also be considered as one of controlling the control and data flow over the inter-cell bus; this will be made clearer in the discussion below.

The periodic mode will be described first. The periodic programs and the times they require service will be listed in a table as shown in Table 11-1. These table entries are described below. It should be kept in mind when reading this section that the actual programs are contained in the operational cells and are executed therein. The controller cell is only performing certain executive functions.

The next interrupt time is the time at which the cell is expected to request service, or will need service. Unlike a traditional computer system, each cell contains a clock. The clocks will all be synchronized.

The period for interrupt time is the time interval between requests. This table entry may be changed by the controller cell or the problem cells as required.

Table 11-1. Control Program Data

Next Interrupt Time
Period for Interrupt Times
Status of Task
Task Identification
Valid Request List

The task status tells the controller cell whether the task is active, or inactive and not requiring attention, or has been suspended for a more important higher priority task.

A task may make certain requests of the group executive. These include input request, output request, data transfer to another cell, global program data from controller cell, etc. Each task is expected to make only certain types of requests. There are task requests that are not valid, and the valid request list may be used to prevent these from taking place. Certain cells, for example, may never be written into, or over-layed with new programs.

The table entries can be time ordered or placed in a linked list as shown in Figure 11-1, where each request contains a link to the next time. With the latter method, inserting new requests and deleting requests that are no longer needed is simplified. Although this method may not be used in actual practice, it will be assumed here in this section.

A real time clock (RTC) in the processor section of the controller cell will be used to time the interrupts to provide periodic servicing of the tasks listed in Table 11-1. The flow diagram in Figure 11-2 shows how the RTC interrupt is processed in the controller cell.

When the clock becomes zero, the interrupt hardware saves the program counter and any other status information as required in the processor. The instruction at the dedicated memory location for the RTC interrupt is then fetched. This is done in block 1.

Block 2, using the start link address, gets the next clock time at which an interrupt is desired. The link is changed to put the top item (which is being processed now) in a new, different list. The Controller Cell (CC) interrupted program counter is saved in this list. The item for which the clock is being set now becomes the top item. The remaining list items remain the same. Block 4 will now allow a RTC interrupt. The link list has been updated so now the interrupts can be stacked.

Block 5 may be omitted if the nature of the service to be performed is known to the CC, such as the next part of a global instruction sequence. This flow chart assumes a worst case in that the cell address must be fetched so that the CC can send

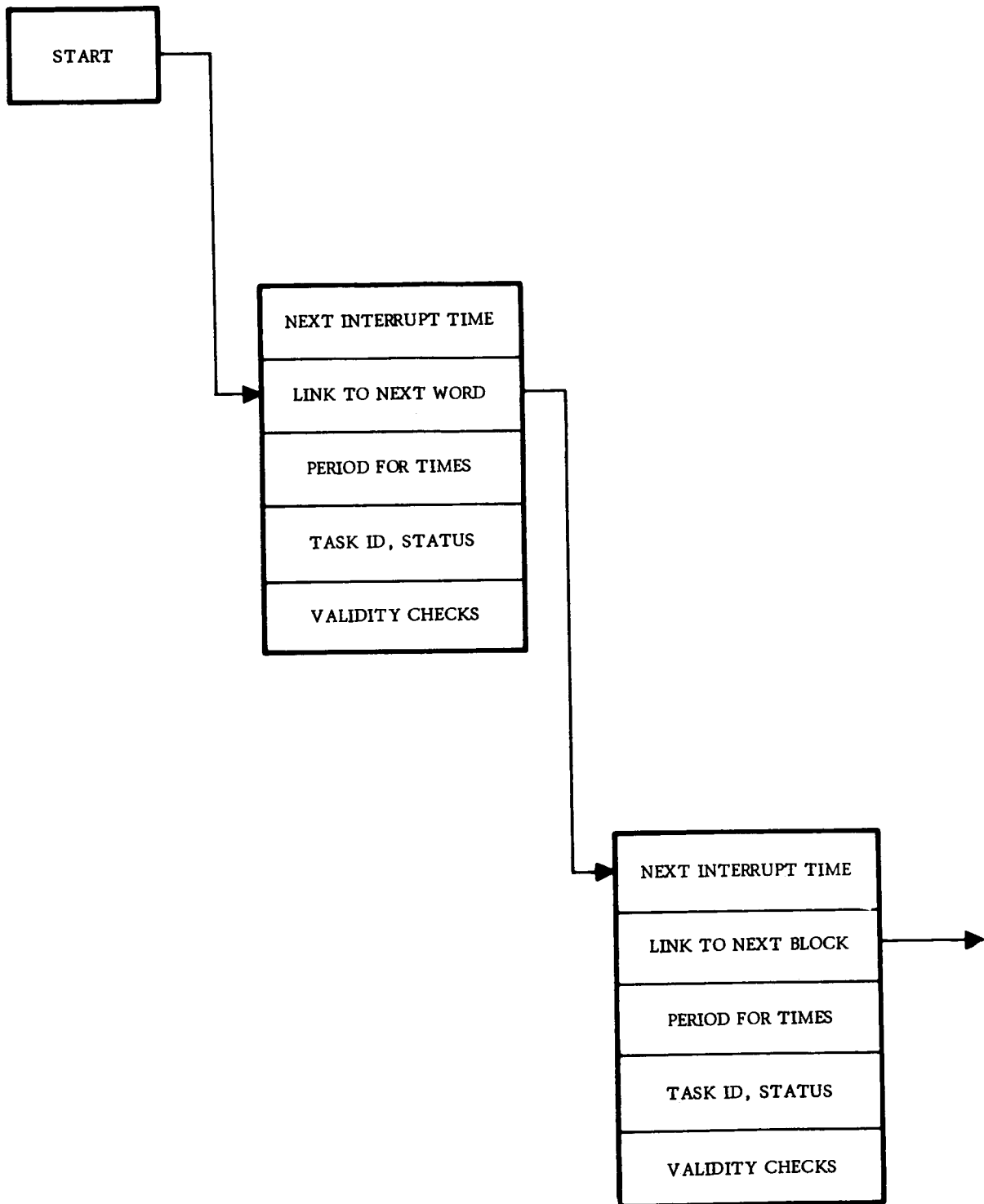


Figure 11-1. Link List for RTC Requests

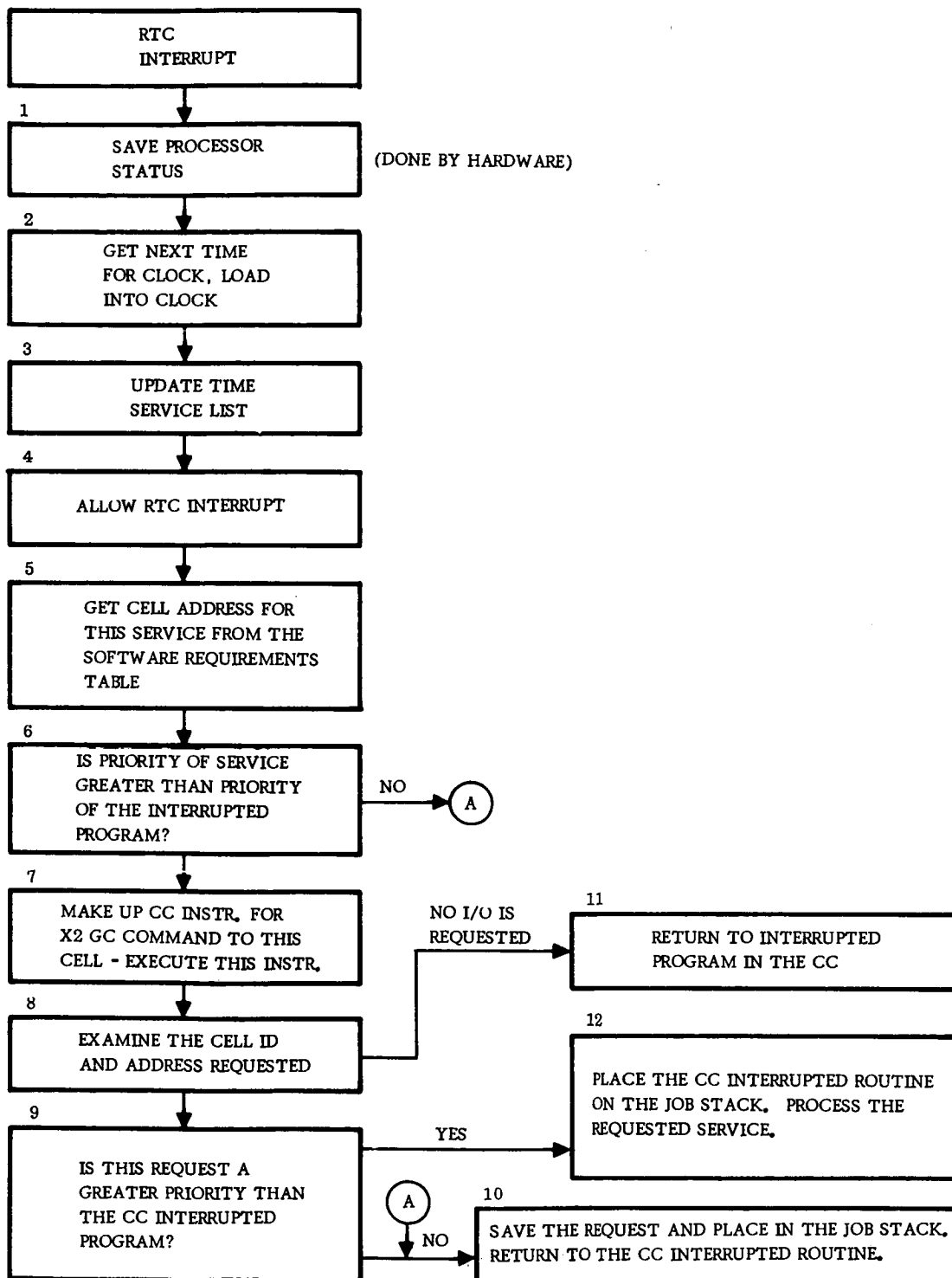


Figure 11-2. Periodic Control of System Resources by Real Time Clock

this cell a command to report its communication request status. (A detailed discussion of commands sent and received on the inter-cell bus is given in Section 10; this chapter simply discusses how some of the commands are used and not how they are mechanized.)

The Task Identification in Table 11-1 is an identifier that points to a table in the CC (Table 11-3). This table contains, for this Task Identification, the priority and the cell address to which the report communication request GC command (hereafter called x_2) should be sent. The priority will be compared to the CC interrupted program priority. The cell address, obtained from this table, is used to make up a CC instruction to send the X_2 command to the cell given by the address. The CC instruction will then be executed in the CC (Block 7). The receiving cell will return a response request word. How the cell forms and sends the request word is described in the cell executive (Section 11.4). The request word will contain whether input or output over the bus is required, an address and a word count. The group executive will examine the requested cell address, address, and word count (Block 8). Knowing the priority and what type of request may be expected by examining the control program data table (Table 11-1), the group executive will decide whether this cell has a higher priority than the interrupted program in the CC and whether the request is valid.

The "Next Job Stack" will contain the list of jobs that need to be done. The concept is used here to explain how the jobs are listed. The group executive can select which job is to be continued with and which is to be held in the stack until its priority has become larger than all the other jobs. Here, in the RTC routine, there are two jobs to consider: the requested service by the responding cell and the CC program that was interrupted. The relative priorities of these two jobs are known to the group executive and thus a decision can be made as to which must be processed first.

Blocks 8, 9 and 12 are described in more detail in the flow chart in Figure 11-3. The CC first examines the response request word received from the cell. Some of the possible requests are: Input, Output, Global Program, No request, etc. Further software study would be required to define the full spectrum of the requests. However, most requests may simply be defined as either input or output. For example, if the cell desired a global program be sent to it, it could request input service from the CC at a certain address in the CC. The CC can identify this from Table 11-1 as a global program request. The CC checks the priority and validity of the request.

A request will be processed by the CC by forming the proper CC instructions in the CC and executing them. These CC instructions may be to send out input/output GC commands over the bus (referred to as X_3 and X_4 commands in Figure 11-3). The GC commands on the bus will then initiate the desired service to be accomplished. Section 10 contains a detailed description of the operations on the bus and the mechanization of the commands sent on it. The intent in this section is to show how these commands interface with the executives and not their detailed implementation.

It should be noted here that as part of the validity check on the request, the CC will determine if the request can be completed in time before the next interrupt occurs. It will do this by examining the word count and the time to the next interrupt.

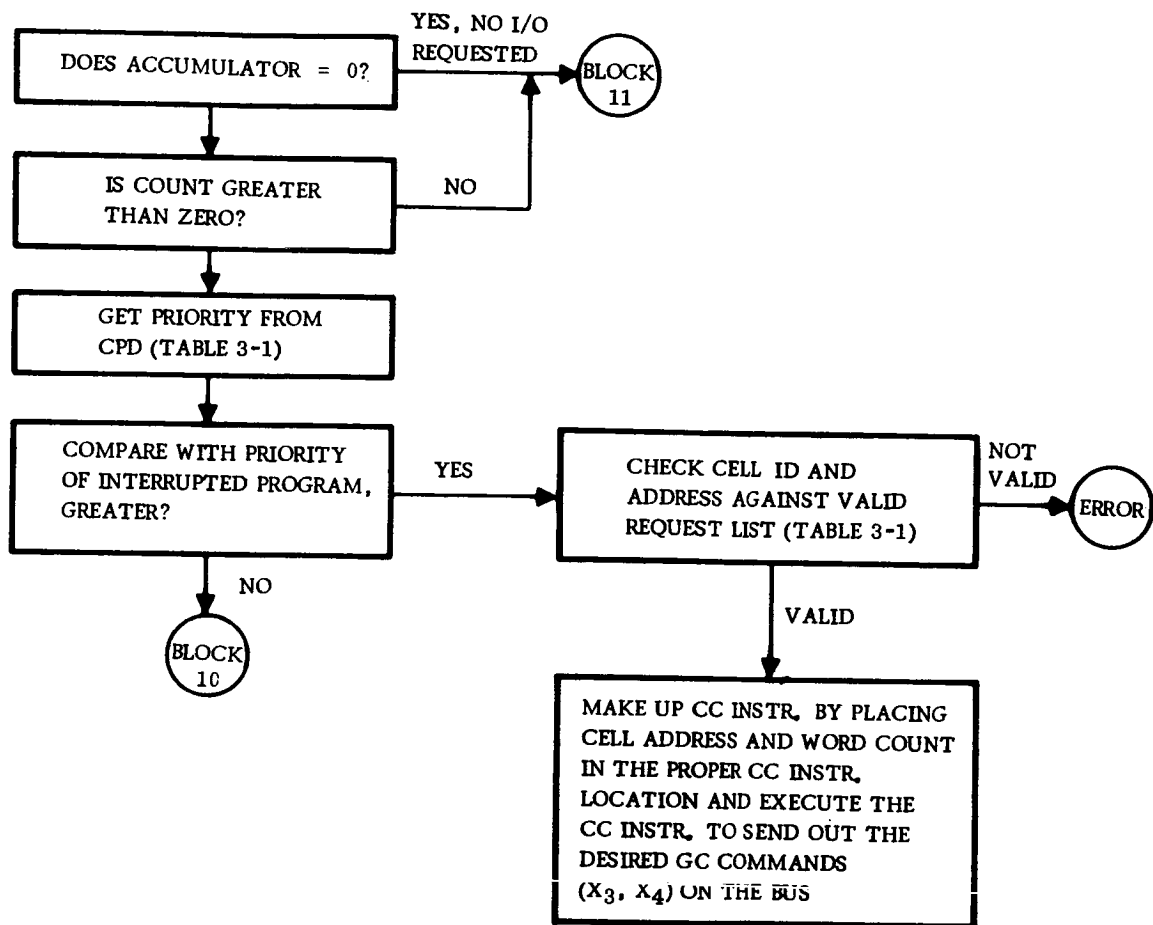


Figure 11-3. Processing a Cell's Request

Since the time for transmissions over the inter-cell bus is known, the CC will determine if the desired word count can be accommodated. If it cannot, the CC can either reduce the word count or treat this as an invalid request. It should be noted that this occurrence should be very rare for the periodic service.

The background mode of service will be described next. This mode of service is handled by periodically entering the "Cell Poll Executive" routine shown in Figure 11-4. The purpose of this routine is to poll or request response for service words from the cells according to some priority scheme. The CC then examines the requests and initiates service as previously described for the periodic services.

This routine will be entered periodically. However, its length will be unknown since the amount of requests that will be received are unknown or cannot be anticipated. It will also be interrupted whenever the periodic service routine needs to be entered as determined by the RTC. Therefore, this routine can be visualized as service being filled in around the periodic service, essentially this is background service. In actual use the CC will be spending most of its execution time on the periodic and background service routines; one can visualize the CC then as periodically servicing certain cells or tasks and then at the spare or "dead" time servicing the other cells or tasks.

As shown in Figure 11-4, the request poll program starts with the task that has the highest priority. The CC forms a CC instruction and executes it to send out a GC (X₂) command that will request a response word from a particular cell. After the response word is received it is examined to determine the "status of the request." This is performed very much the same as described previously under the periodic service: the type of request is examined, cell address and word count are checked. The validity of the request is checked and the appropriate action taken similarly to the previous description.

A problem that will occur in this routine much more frequently than in the periodic service routine is that of completing the requests in time before the RTC interrupt. If the request is for input/output communications over the inter-cell bus, the CC will examine the word count requested. It will determine if the desired communications can be accomplished in the time interval before the next RTC interrupt occurs. Actually, this interrupt will not bother the communications, since it affects only the controller cell (unless the controller cell is one of the parties to the communication). However, the RTC will need the intercell bus (usually) to send a GC command and receive the response. Thus I/O should not be started if it cannot be completed before the RTC runs down.

If there is not time, the executive will either reduce the count so that there is time, or will decide to not allow the I/O to be performed. If the latter course of action is chosen, special action needs to be taken since the response has been fetched from the cell. The cell has set itself up for I/O transmission, and is expecting a GC command and data. One way is to return a GC command (input or output) with a count of zero. Thus the cell executive will note the count and try again later. Another approach is to use a special GC command to cause the cell to return to its status before the GC x2 was received on the inter-cell bus. Either approach can be used.

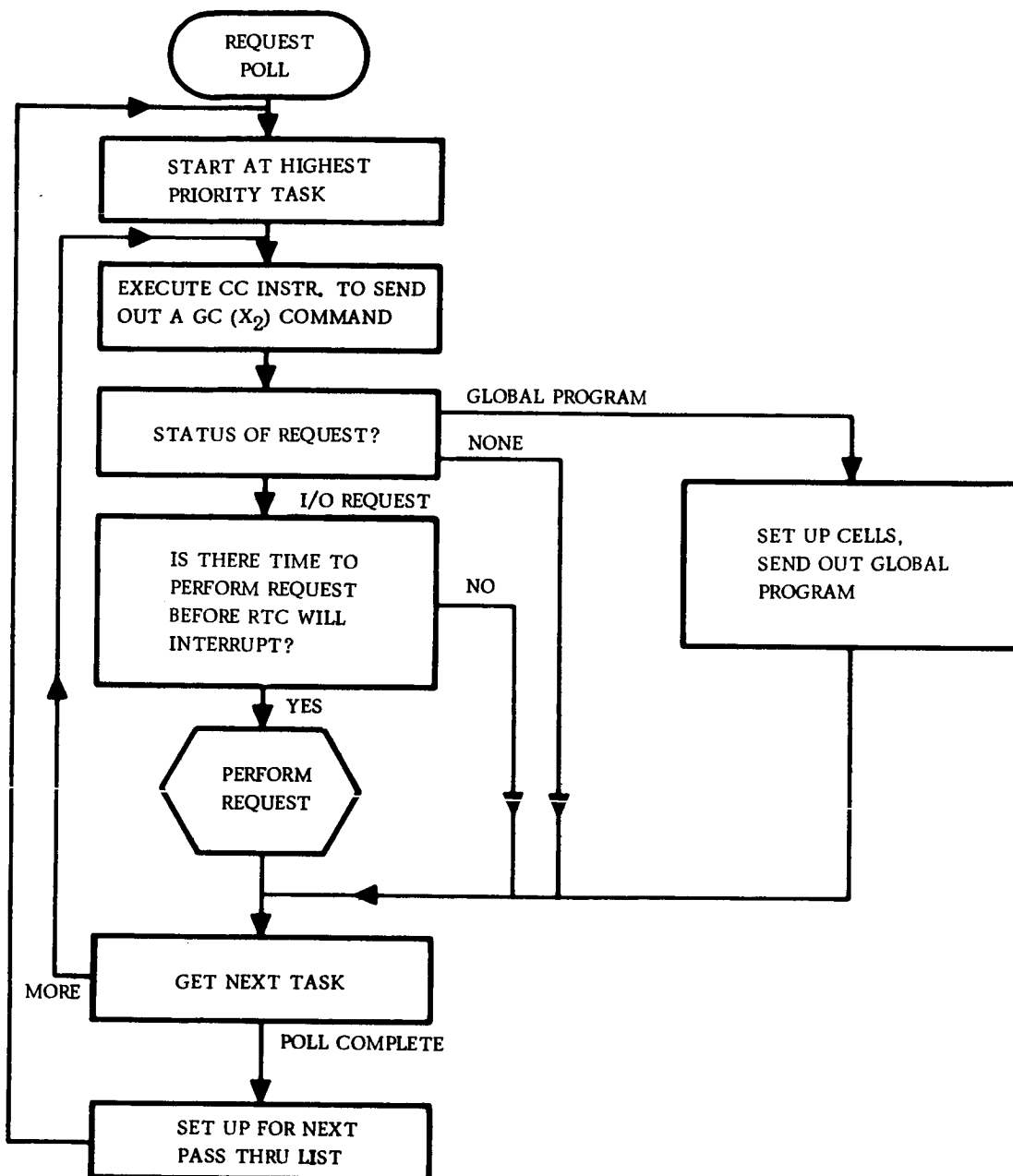


Figure 11-4. Cell Poll Executive Routine

Upon servicing this request, the next task according to priority will be polled. The rate at which each cell is interrogated depends upon the priority of the tasks and how the priority relates to the inter-cell bus communications requirements. The priority scheme is not defined here, nor can it be defined until the system requirements, I/O conditioner specifications, scientific experiments as well as the other tasks have all been defined. The only assumption made here is that the priority will require some cells to be interrogated more frequently than others. At the end of the list, the program is re-initialized for another pass through the list of tasks.

It should be noted here that for both modes of service, periodic and background, the CC will not be able to use the bus during the communications time for the request being serviced. That is, if Cell 1 wants data from Cell 2, during the time that Cell 2 is transmitting the data to Cell 1, the CC can not interfere by trying to use the bus. During this time the CC can do other tasks such as various internal house-keeping executives. (Of course, if the CC is one of the parties to the communications, then it is tied up communicating and cannot be doing any other tasks.) The only difficulty in doing these other internal tasks is that the time to accomplish the communications will be quite random. However, for any given request the CC will know the count of the number of words to be communicated. It will therefore know the approximate time to communicate and may thereby select which one, if any, of the internal tasks to take on. The CC may either keep on doing an internal task until it receives an "IC Busy" free interrupt (recall that the transmitting cell sends out a global command at the end of transmission that resets the IC Busy FF; this can then be used to generate an interrupt) or it may simply set its clock to interrupt itself after a specified interval of time. The "IC Busy" free interrupt will be used quite frequently during the background mode of service; the interrupt can cause the CC to then poll the next cell for a request.

11.2.2 Furnish Global Programs to Dependent Cells

The cell or cells that contains the group executive contains all the global instructions that are to be executed by the dependent cells. Because these instructions are sent on the inter-cell bus and the controller cell controls the inter-cell bus, the controller cell sends out all the global programs. These global programs may be simply sent out by the controller cell or they may be sent out in response to a request either by the periodic or background service routines.

At the time a global program is to be sent out, the controller cell will transfer to the start of the global program. This program will usually begin by addressing the individual cells that will be used and setting each one to the desired level. Once this is done, the rest of the global program is fetched from the controller cell memory and sent to the dependent cells. The controller cell transmit and execute mode instructions are used to control the sequence of instructions sent out from the controller cell. During this time, the inter-cell bus can not be used for input/output. If time is needed for input/output, the controller cell must discontinue the global program before any input/output instructions can be executed.

The instructions used by the controller cell to change modes and control the global program are described in section 6.

11.2.3 Allocate System Resources

The group is made up of hardware resources, software resources and time. The efficient use of these resources is required to perform all the tasks required. This efficiency can become important as the space flight continues and hardware malfunctions occur and the hardware fails. The remaining resources must perform efficiently to ensure mission success.

Before the executive scheduler program can begin to allocate system resources and schedule the use of these resources, there must be made available to this program a list of the hardware resources in the system and a list of what tasks must be performed by the group. Table 11-2 shows a hardware resources list. The resources are listed with their status. The status will contain such information as power off, malfunctioning, failed connections, etc. The tasks are all the tasks that require this hardware resource.

Another set of data required is the software requirements. Table 11-3 gives these requirements. This table is changed by the system executive as the needs of the computer system change. Each task that is performed has a priority and a list of major programs that make up this task.

Table 11-2. Hardware Resources Table

Resource	Status	Tasks using this Resource
Cell 01	3	2, 56
Cell 02	2	3
.		
.		
.		
I/O Conditioner 01	1	8
.		
.		
.		
I/O Connection to I/O Conditioner	8	7, 23, 9
.		
.		
.		
Inter-Cell Bus	1	
Group Bus Switch 1	1	
Group Bus Switch 2	1	

Table 11-3. Software Requirements

- | |
|--|
| <ol style="list-style-type: none">1. Task Identification2. Priority3. Programs Required by this Task |
|--|

Table 11-4 gives detailed program requirements. Usually a copy of this data will be kept in the bulk storage unit and called when required. This table is important, since it links the software to the hardware. Table 11-4 will be discussed in detail below.

Program, Task Identification

The program number, program, task identification or task number, identifies the program to the system and is also used in the hardware resources table.

Number of Cells Required

The number of cells required is the normal cell requirements. However, there exist programs whose cell requirements can vary. Some, such as data reduction programs with long global programs, can vary from one to as many cells as there are available in the group. For these special cases the number of cells may be varied by the executive scheduler to accommodate other requirements and restraints.

For most programs, there is a number of cells that makes good use of the storage and does not use too much inter-cell bus time. Fewer cells may require more I/O or inter-cell bus time, more cells may require more neighbor communication time, executive time and memory. If there are no problems the normal cell count will be used. The executive scheduler will use this program, as compiled, throughout the entire mission. If problems arise, such as (1) using all the spare cells in a group, (2) the intergroup bus fails or (3) a group fails; then the system executive may recompile some programs to use less of the system resources. Note that the system executive may wish to recompile certain critical programs to reduce their reliance upon resources that present a single point failure. For example, if one inter-group bus fails, the other is now a single point failure. Following the time of the inter-group bus failure, the system executive will direct one of the groups to recompile the critical programs to bypass the inter-group bus or to prepare to use alternate paths (such as through I/O conditioners or the manual consoles). Thus if the second (or last inter-group bus, if there are more than two) fails, the critical programs are prepared for this failure. Conversely, if an I/O conditioner should fail, these same critical programs may be recompiled to make more use of the inter-group bus and/or use critical data from alternate I/O conditioners.

The distributed computer system has the properties for providing many paths and alternate ways to accomplish tasks. Thus any rules that can be set down can be broken under 'certain' conditions, especially when failures occur. In present, nondistributed computer systems, there is a memory, a processor and I/O

Table 11-4. Program Requirements

1. Program, Task Identification
2. Number of cells required
3. Word Count for Each Cell
4. Priority, Etc.
5. Periodic or on Request
6. Time Period (if Periodic)
7. Input Requirements
8. Output Requirements
9. Neighbor Requirements
10. Cell Bus Usage
11. Starting Location
12. Restart Location
13. Other Programs Required by this Program, Such as Subroutines

conditioners. A program is in memory and instructions are executed periodically. In the distributed processor, there is no processor penalty for more rapid execution or more repetitive execution. Note that this does not apply to the I/O or inter-cell bus system.

The distributed processor can, by recompiling the programs, change:

1. Storage
2. Number of cells used
3. Inter-cell bus usage
4. Inter-group bus usage
5. Neighbor to neighbor usage
6. Execution time

Thus many variables are available to mold the system to meet certain objectives.

Word Count

The word count allows the executive scheduler program to best fit the programs in the group. This parameter is of particular importance when several programs must share a cell.

Priority, Etc.

This parameter depends upon the use of the program. A subroutine which is used by a very high critical program will be of the same priority, whereas the same subroutine, when used only by a low priority program, will have a low priority. The priority is very application dependent. At the time the system is firmly defined, the priority numbers may be assigned to the programs. The executive scheduler, using its knowledge of the system requirements at a particular point in time, may change the priorities to best insure the probability of mission success.

Periodic or Request

This refers to the fact that some programs are required to be performed at certain intervals of time. Other programs are executed only upon the request and demand of another program. Periodic programs, on the other hand, are executed on demand of the real time clock.

The periodic programs are divided into two categories, the absolutely periodic and the average periodic programs. The absolutely periodic programs must be started at periodic intervals, e.g., 0, 20, 40, 60, 80, ...

The average periodic programs may be varied in starting time, but the average starting times are constrained to be within some limits. For example, the sequence 0, 25, 45, 60, 80 is acceptable. The delay from 20 to 25 may have been caused by a higher priority program interrupt.

Input Requirements, Output Requirements

All required interfaces to the outside world are defined. The conditioners that are needed, the data rate, the identification of the data are all listed. The cell bus usage is not required here, as the cell bus usage is handled in another table entry.

Special input requirements, such as special sensor commands that must be output before the processing can begin are given in this table entry. Again this data is very application oriented.

Neighbor Requirements

Any spatial requirements imposed by neighbor cell communication are given here. It is expected that all the tasks that will not fit in a single cell will use a small number of cells, working together, to form a SET of cells. This set would be fitted with the other sets that make up the group. This fitting is a geometric relationship, fitting the sets of cells among the good cells of the group.

Cell Bus Usage

The executive scheduler must know the inter-cell bus usage requirements of the programs. These table entries give the inter-cell bus usage requirements of the program. This problem is discussed in considerable detail in the latter part of this section where the executive scheduler is discussed.

Start Location

The start location is the location where the program is to begin execution. If this is a dependent global program, the initial locations will usually be the controller cell setting the cells to the proper level.

Restart Location

The restart location is for those programs that have initialization procedures in the program. The restart location, or locations, are used when the calling programs have different initialization requirements.

Other Programs

Other required programs are listed for the reconfiguration programs and the schedules.

The executive scheduler will have the hardware resources table, the program requirements table, and the software requirements table at its disposal. The software requirements table, made up by the system executive, will contain a list of the absolutely essential programs, the high priority programs and the low priority programs.

Figure 11-5 contains a flow chart of the executive scheduler and will be discussed in detail in the remainder of this section.

The executive scheduler will take each task and fetch the Program Requirements table (PR table, Table 11-4) for this task, from the bulk storage unit. The PR table entry for this task may contain "other programs required" entry that calls out other programs that are needed. The executive scheduler program will add these to the list of required programs. The duplicated are then eliminated. Storage and the inter-cell bus requirements are added up. The I/O requirements are listed as I/O conditioners and I/O connections. After all the required (essential) tasks are processed, the storage and other hardware sources that are available (not failed) are compared and examined. A problem exists if the total storage is more than the total group capacity. Actually the storage available will be less than the total words of storage because of the overhead and some wasted space in some cells. Thus, if the simple sum of all memory storage requirements is greater than all the cell memories, there is no way to fit all the programs in. A similar line of reasoning holds for the inter-cell bus and I/O requirements. The expectation of the resources being a problem at this point in the executive scheduler is small. Later in the scheduled program there may be a cause for problems if the demand for service is too great. The next block in the flow diagram examines the inter-cell bus

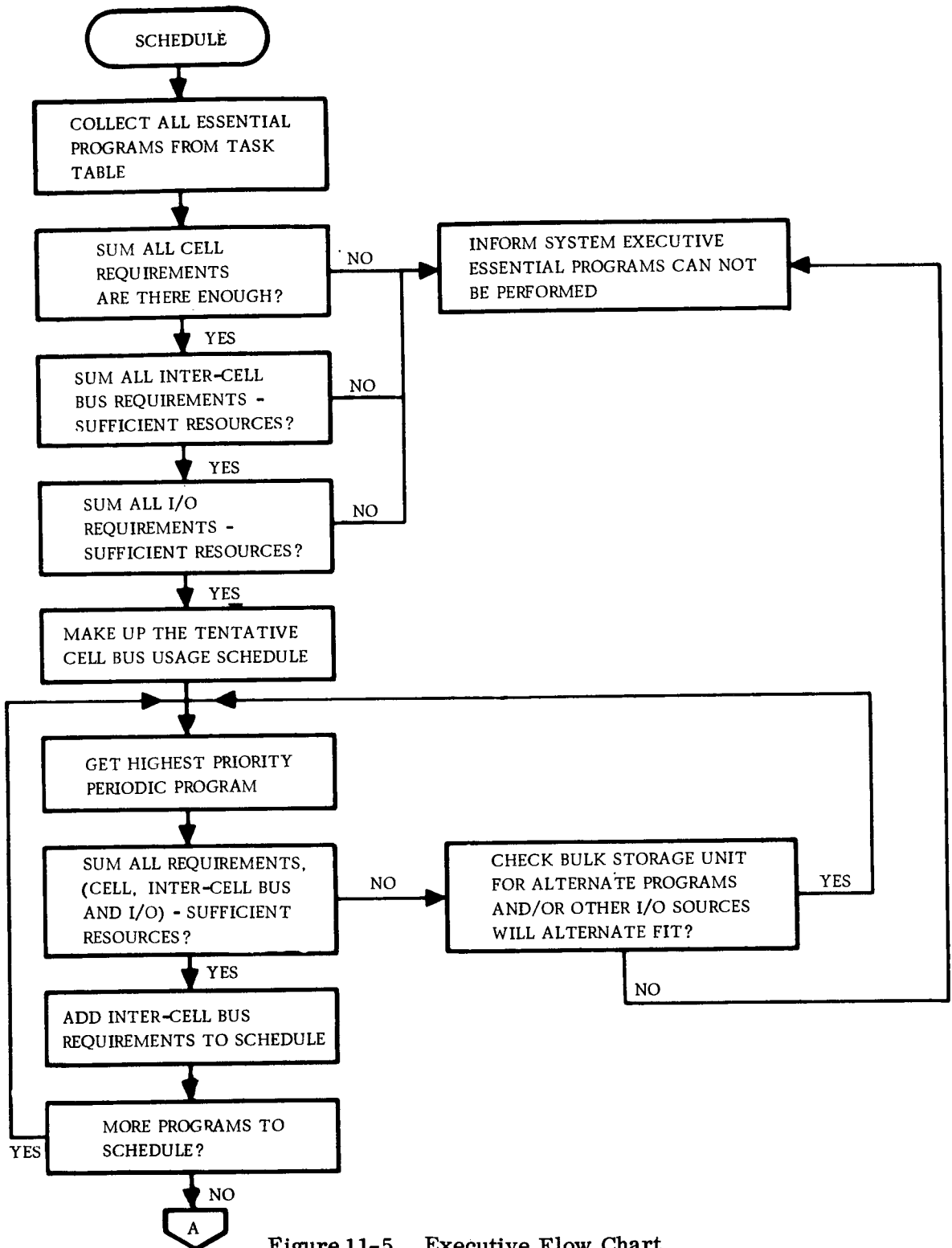


Figure 11-5. Executive Flow Chart

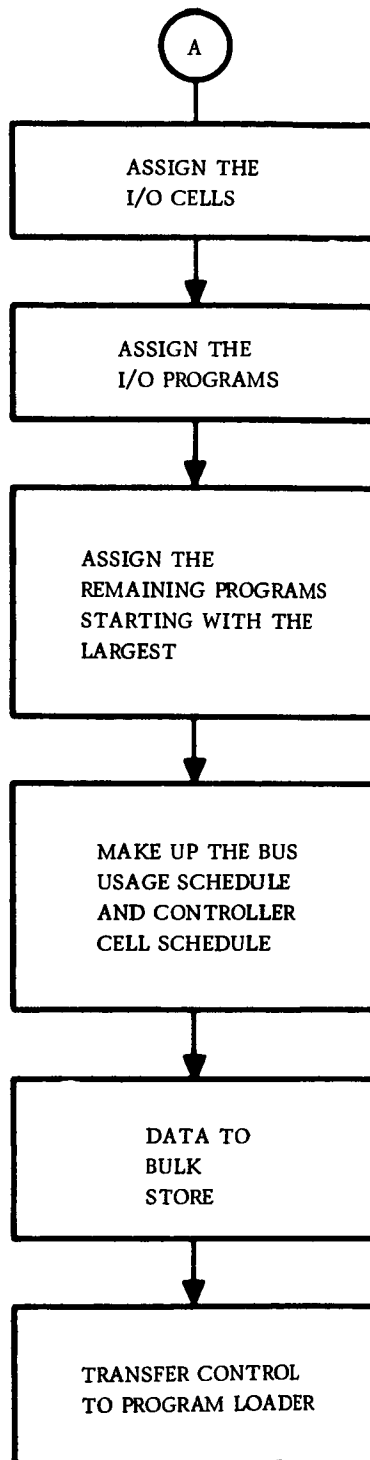


Figure 11-5. (Cont)

requirements and makes up a tentative bus schedule. A major cycle time is selected and the time requirements, rate and deviation of each task are noted. The software test and controller cell to system executive times are included. The result is:

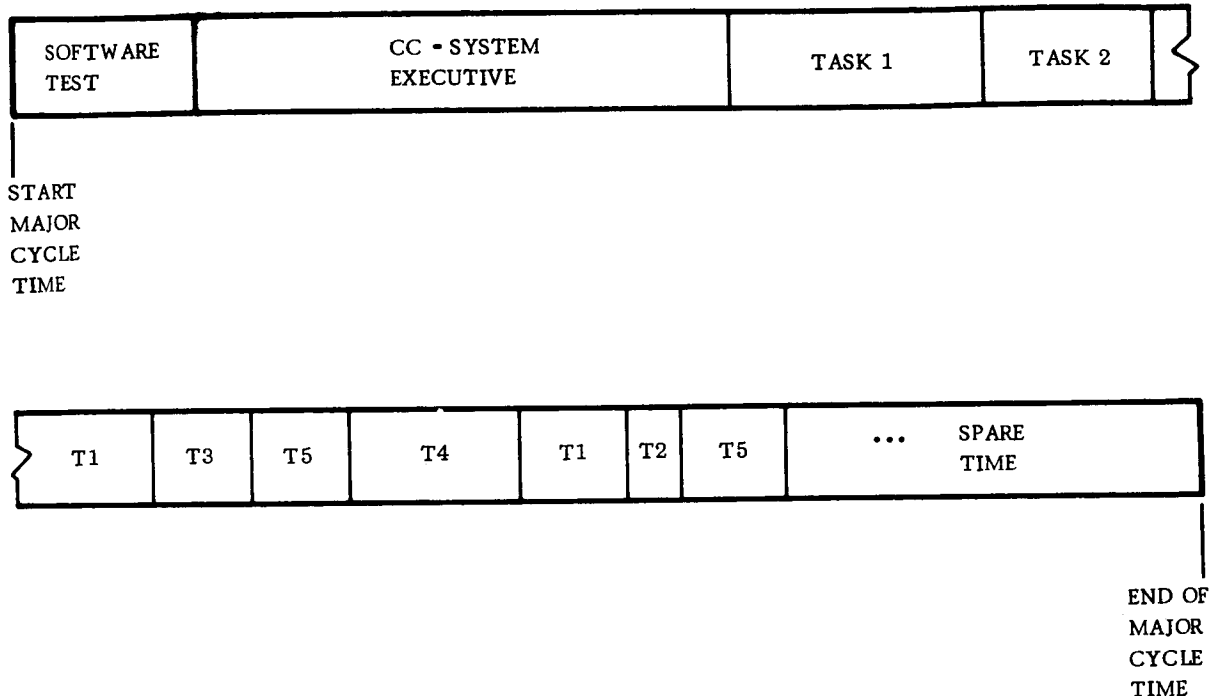


Figure 11-6. Preliminary Time Schedule, Inter-Cell Bus

Note that there is no attempt in Figure 11-6 to order the tasks, they are just placed in approximate order and duration. The software test, for example, is usually done last, not first. The contrast to the final order will be evident later on when Figure 11-10 is discussed. The exact periodic and fitting of times will also be done later.

Having gotten to this point the executive scheduler will begin to make up the final bus schedule. The highest priority task is selected and all the requirements for this task are listed; the storage is totaled and the number of cells required are noted. For example, if the controller cell requires 2-1/2 cells, one-half a cell is available for other use; 2-1/2 cells are removed from the group resource list. The inter-cell bus requirements (which for the controller cell will be the software test and communication with the system executive and bulk storage unit) are noted. The preliminary schedule (Figure 11-6) is used, the exact times are set and the durations are noted. This may push aside and cause some other tasks to be delayed. This task is now assigned time slots and if the program requires an exact periodic timing, these times will be fixed. Any special I/O requirements to any conditioners are noted.

The next highest priority program is obtained from the list of essential tasks and the storage requirements obtained. Cells are allocated to contain this task. The inter-cell bus requirements are obtained and put in the same spots on the preliminary schedule. However, this time the exact times are used; the tasks are placed next to one another (see Figure 11-7). If each task is an integer multiple of a common sub multiple, there will never be any time in which two tasks require the bus at the same time. The use of a major cycle time will also prevent the occurrence of two periodic tasks requiring the bus at the same time.

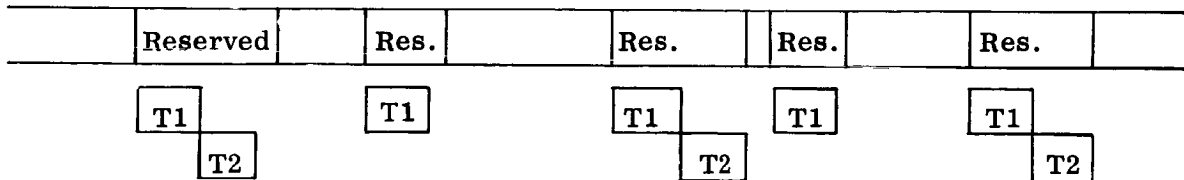


Figure 11-7. Constructing the Bus Schedule

By using a common submultiple for all essential periodic programs, the scheduling is simpler. One possible sequence is shown below; every periodic task is executed at one of the following rates – no other rates are used.

Times per second

. . . 1/16 1/8 1/4 1/2 1 2 4 8 16 32 64 128 . . .

Sometimes a task to be added to the schedule will require a block of time that is longer than the available time. First, a trial will be made to shift previous tasks to leave enough room as shown in Figure 11-8.

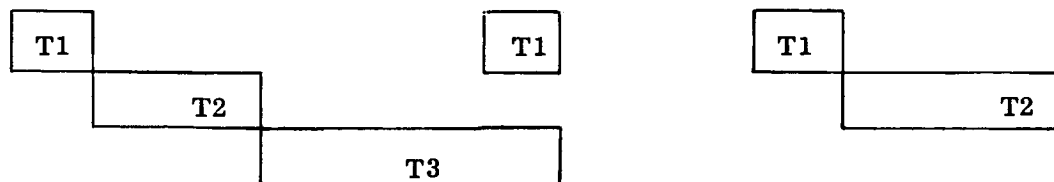


Figure 11-8. Shifting the Bus Schedule

Note that T3 is too long. By delaying the start of T3, BUT NOT THE PERIOD, the task can be made to fit (Figure 11-9).

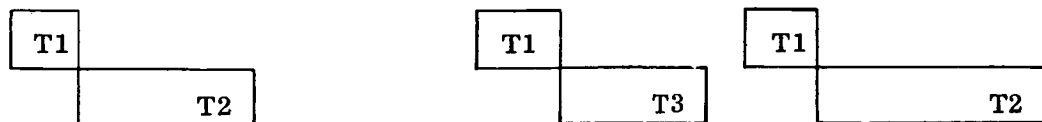


Figure 11-9. Fitting Periodic Tasks in the Bus Schedule

If this attempt is not successful, the scheduler program will split the program into sections. Note the tasks that are split are always of a lower priority than the tasks that are already scheduled. Thus the highest priority tasks will not be interrupted by a lower priority task (Figure 11-10).

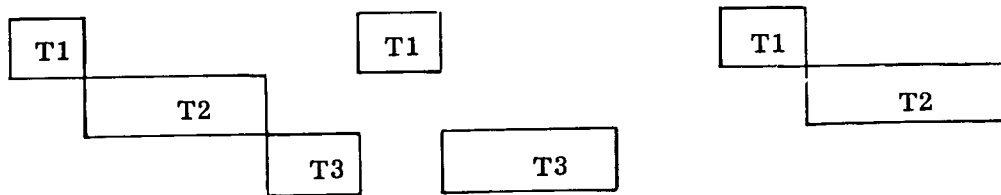


Figure 11-10. Splitting the Periodic Use of the Bus

The periodic and approximately periodic programs will require accurate time placement, the exact cell bus usage times will be scheduled and this time list will be used by the group executive when the object programs are being executed. For other programs, the request poll (see Section 11.2.1) in the group executive will be used. This poll program will start I/O over the inter-cell bus only when the periodic programs, which are already scheduled, are not using the inter-cell bus. However, some time, unused by the periodic programs must be allocated so the remaining programs can use the inter-cell bus. If all the inter-cell bus time is reserved for periodic programs, there will be no time left for the polled programs. The latter programs must be given some spare bus time. The preliminary time schedule made up initially will insure that sufficient time is available.

The cell assignments are made next. An example will be given to illustrate the procedure. It is assumed the compiler has calculated the word count needed for each task, the number of cells and neighbor lines. Figure 11-11 contains an example and shows the present step in the assignment procedure as shaded area, and past assignments as cross-hatched areas. An explanation of each assignment step is given below:

1. All tasks that require I/O conditioner connections (via an I/O line) are done first, Figure 11-11(1). These cells have special hardware connections to the conditioners that will not be changed unless a failure occurs.
2. All tasks using these I/O cells are assigned to cells that have a neighbor line connection to these I/O cells.
3. The task with the largest connected set of cells is assigned to the smallest set of cells in which it will fit. The executive scheduler program will try to leave the resulting area so there are no isolated squares. Thus Figure 11-11(3a) is preferred to 11-11(3b).
4. Since long closed-end connected cells are more difficult to fit than rectangular or open-end connected cell sets, the former will be fitted next. The term closed end means that the first cell is connected to the second, the second to the third, the third to the fourth, etc., and the last is

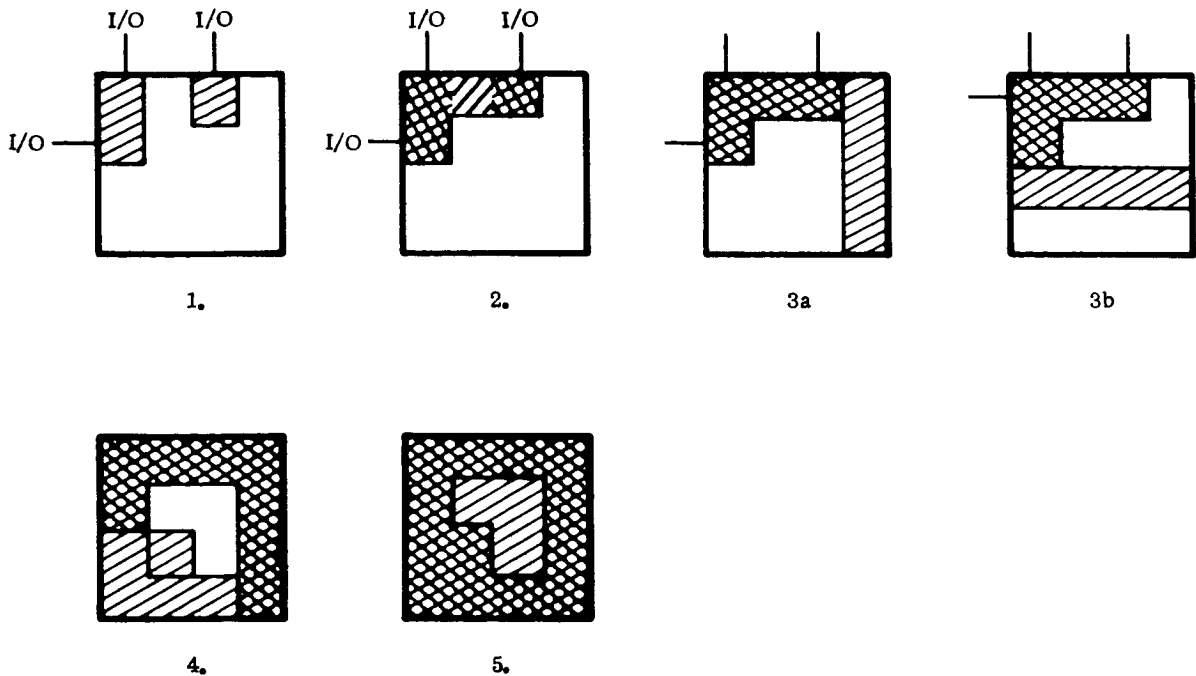


Figure 11-11. Cell Assignment Procedure

connected to the first. Also included here are all the sets of cells with complicated neighbor connections.

5. Finally, all the single or pairs of cells are fitted into the remaining areas.

Several trials may be required to get all the cells to fit within the group. By noting what cell geometry is required and the present spatial relationship, the allocation procedure can begin again, this time attempting to make room for the remaining cells. The algorithm to be followed to attempt to reassign the cells has not been defined at this time. Section 4 of this report discusses the problems associated with optimum assignment procedures for parallel computations. It was seen that this is a very complicated problem. Nevertheless, it is assumed that some algorithm will be available to the executive scheduler for reassignment purposes.

If the tasks cannot be all assigned to the group, there are many alternatives that can be resorted to. Some of these are:

1. Increase the efficiency of the existing tasks so more will fit in the capacities of the group. Compiler optimization techniques will be improved over what is being accomplished presently.
2. Reduce the rate at which tasks are executed. If a task is not required all the time, execute it only when requested.

3. Swap programs in and out of the bulk storage, thus sharing the group storage capability between tasks.
4. Schedule the use of resources, such as subroutines, I/O routines, etc., instead of repeating these routines wherever they are needed.
5. Increase or decrease the amount of applied parallelism to reduce the time and/or storage requirements. (This would generally be done in the reassignment algorithms.)

If there are problems, there will be a program or programs that will not fit. The programs of lowest priority may be deleted, or they may be brought in from the bulk storage unit when required. In this way, several programs may time share several cells, there being one program using the cells at one time. Or the lowest priority programs may be eliminated entirely. In all cases of eliminated programs the system executive will be notified because there is always the possibility that another group has room for these deleted programs.

It should be noted here that there may exist several assignment algorithms in the bulk storage unit and the executive scheduler may call the desired one depending upon the phase of the mission and/or the reason for using the algorithm. For example, an optimum assignment algorithm may be used in the heavily loaded Mars orbital phase either due to attempting to initially schedule the tasks or due to a failure. However, during the critical mission phases, if a failure occurred a suboptimal algorithm may be resorted to for reasons of quickly accomplishing a reconfiguration.

Having completed the assignment routine, the executive scheduler can assign the actual cell addresses, make up the inter-cell bus I/O commands, etc. The neighbor commands are all relative, so unless the scheduler had to change a relative position, there will be no need to change the compiler generated coding.

11.2.4 Test the System Hardware and Software

Section 8 of this report describes many approaches to the software test problem. The fourth approach, described in Section 8.1 is discussed here.

The software test program begins in Figure 11-12 by using a presently unused cell level and setting a single specific cell to the dependent state and to an unused level. The controller cell will ask for and expect a response. The cell is set to another level. The accumulator, etc., are moved to a temporary storage area. This procedure gives some confidence that the cell's state and cell address register are functioning correctly. This procedure is repeated for all the other cells. The controller cell now transfers control to the cell that contains the test program. This new cell, now the controller cell, will send out the software test program for all the other cells.

Notice that the given address and data concept introduced in Section 6 are now extremely important. The processor and memory can be tested without having each cell save many constants, addresses, etc. In fact, each cell will need to move very few registers out to make room for the software test program. Some registers to

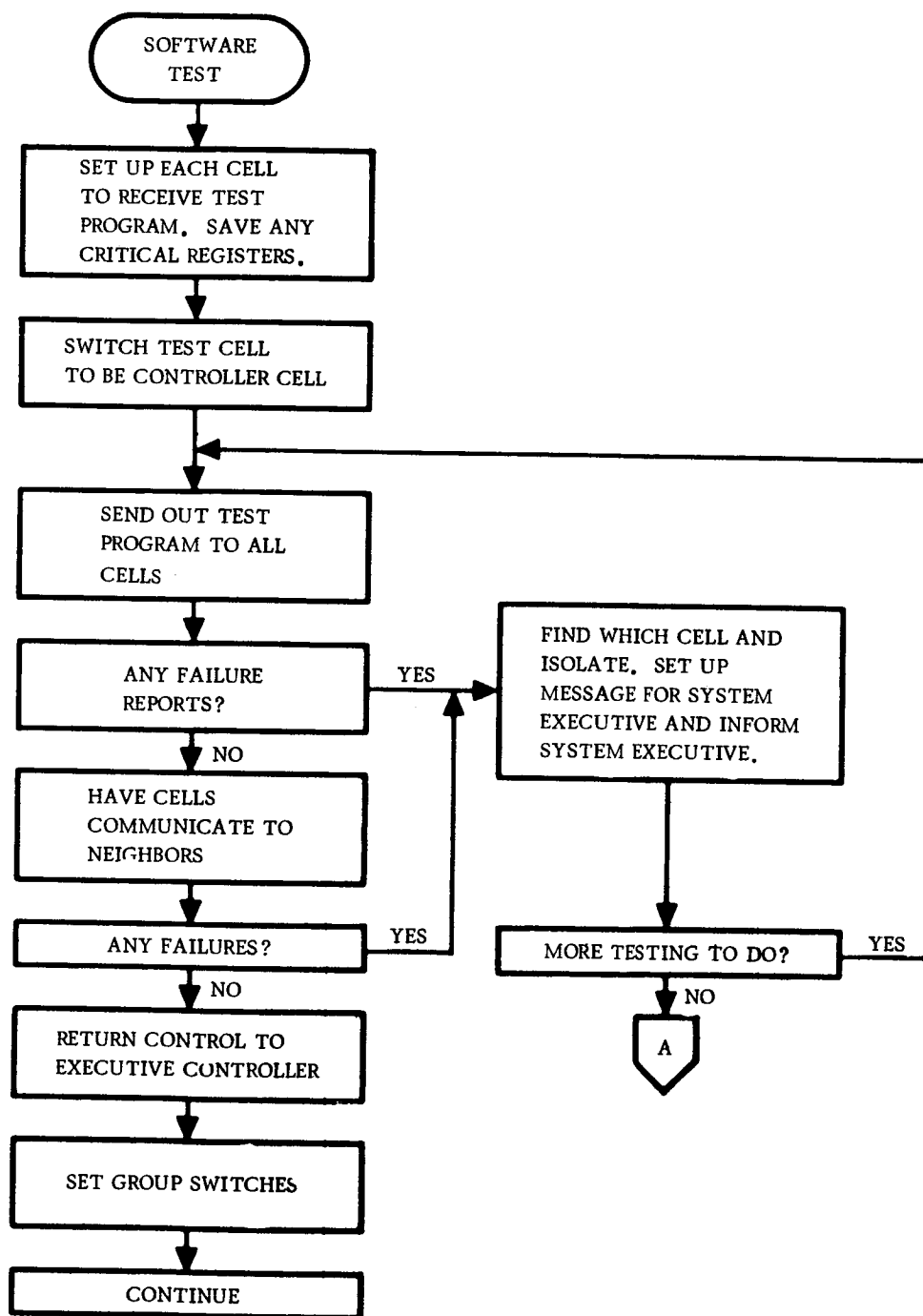


Figure 11-12. Software Test and Reconfiguration Routines
(Sheet 1 of 2)

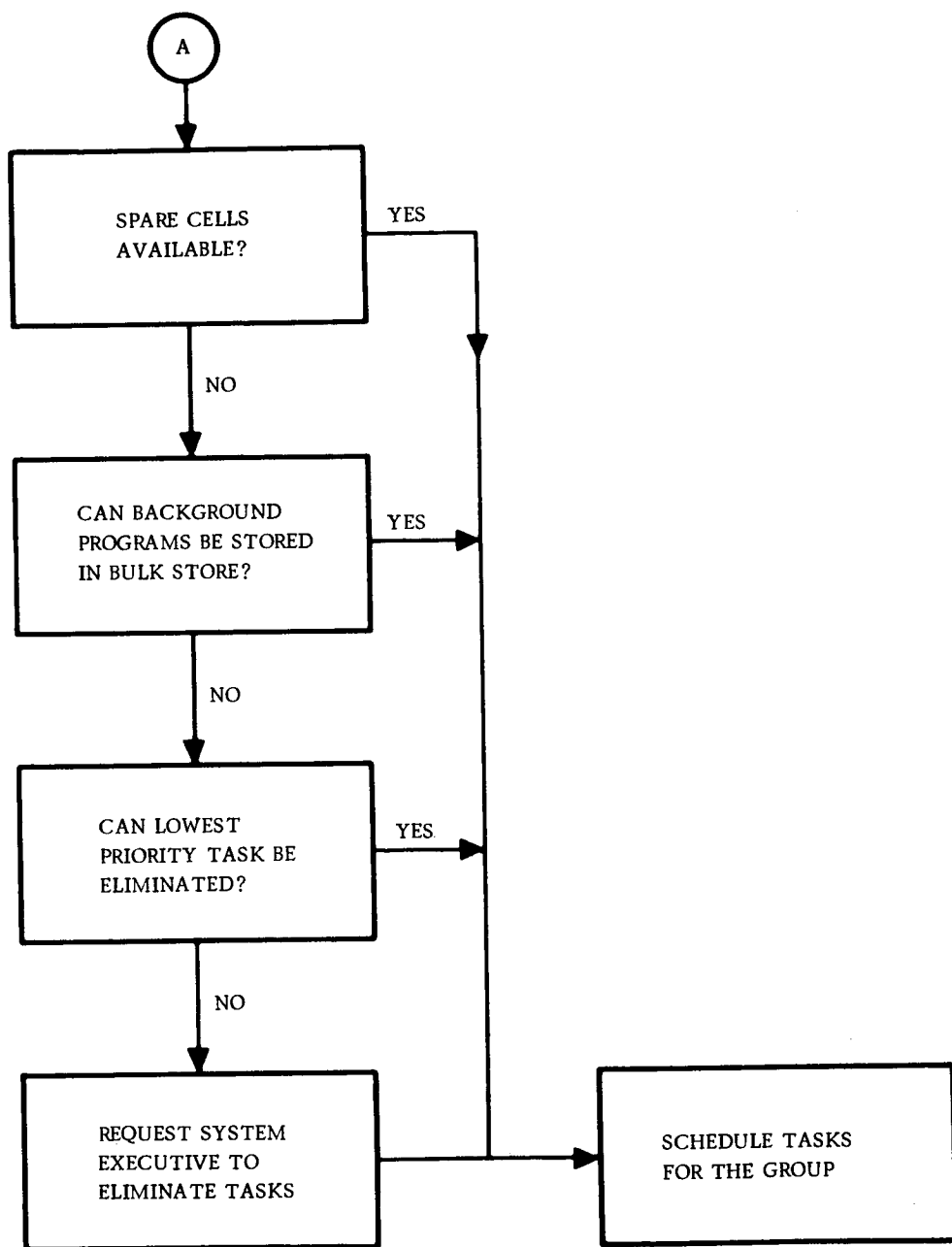


Figure 11-12. Software Test and Reconfiguration Routines
(Sheet 2 of 2)

be moved include the program counter, some index registers, and status registers. The operational program contained in the cell is assumed to be stopped at some point where the accumulator contents may be destroyed by the software self-test program, if not, it will need to be saved.

The software testing program should require less than 500 words. Because there are no words required in the receiving cells, all the constants, etc., are sent as data (D16 or D32) over the inter-cell bus. The last commands would make the cells transmit to their neighbors their final test results. Each cell, testing its results against the neighbors, will report to the controller these 'good - no good' results.

By using the level response command, the controller cell running the tests can obtain a report of (1) no failure reports or (2) one or more failure reports. An individual cell by cell poll can determine which cell or cells had sent in the malfunction report.

At this point, either all the cells test good or there is a failure report or reports. Of course, if all is well, the test controller cell will return control back to the original controller cell. The original controller cell will now send the proper response words to the group switches. (See Section 8 for a discussion of the operation of the failure detection system.)

The test controller cell, upon receiving a failure report, will isolate the cell, and continue testing the other cells. The failure report will be sent to the system executive and the original controller cell. This completes the test controller cell routine; the diagnosis having been completed.

11.2.5 Reconfigure the Group

Reconfiguration is required under two conditions. There are the normal changes due to phase changes in the mission; e.g. new programs are required during the coast phase which are different from the requirements of the Mars orbital phase. Secondly, reconfiguration may be required upon a failure or malfunction of a cell, group, bus, etc.

To perform reconfiguration due to phase changes, there must exist a means of changing the programs by overwriting the programs that are not needed with the new programs that are required. The reconfiguration during normal phase changes has the advantage that the procedures can be simulated on ground based computers to ensure that the software will smoothly change from one phase to another. Thus the interactions between one program and another can be noted and any adverse effects can be corrected.

If the group is not performing periodic programs and the group programs that are to be retained do not keep a data history, the entire group can be loaded using the resources tables. For many groups this can not be done. One means of loading the group is to monitor the programs operation. As the cells complete all their programs, a cell can be loaded with a new program and set to inactive. As tasks are completed, they are removed (if they are not needed or must be moved) and the cells are loaded with the new program. As soon as the new periodic programs are loaded, the old periodic programs are made to average periodic so as not to disturb

the new time schedule. An excellent time to change the periodic programs is to reload cells instead of performing the software tests. After the periodic programs are loaded and are performing correctly, the background programs can be started. The group is now reconfigured with the new programs.

Reconfiguration because of a hardware failure is a more serious and difficult problem. Some simulation of failures and the reconfiguration programming can be done on ground based computers before flight begins. However, there are too many possible combinations of critical programs and hardware configurations to check all possible problems and solve each one. There are some that can be simulated easily, such as a group failure or an inter-group bus failure. There are two solutions to the problem - (1) a general algorithm to reconfigure any hardware - software system (S1) into another hardware - software system (S2), or (2), set up the first system S1, in a form that can tolerate as many varieties of failures as possible and still perform the critical functions.

The first solution is very complex, it is the most sophisticated and is the goal of the failure reconfiguration considerations. The objective is to go from any state S_1 to another state S_2 which will maximize the amount of computer resources that can be used for background programs. In other words, the most efficient assignment of the programs is desired. Some considerations of the general algorithm were given in the section on parallelism (Section 4) in this report. It was seen there that there are many possible solutions to performing tasks in parallel and in general one has to resort to a suboptimal algorithm to assign tasks. In general each assignment will vary the cell storage requirement, speed requirements, inter-cell bus usage requirements, etc. This subject is very complex and deserves much more consideration before any conclusions can be drawn as to the merits of optimal assignment algorithms.

The second solution is to ignore efficiency and configure the system into a very reliable, fault tolerant one. In this approach background and non-critical programs may be executed at lower rates and lower response times expected when reconfiguration occurs with no opportunity to seek out an efficient utilization of the system resources.

The first page of Figure 11-12 presented the general procedures for the software test. The second page of this figure contains the flow chart that is entered if a failure occurs; this is part of the reconfiguration routine. The result of a failure is a change in the hardware resources table. A comparison between the present and new tables will first be made to determine if a simple solution is available such as to start up a spare cell or two. If this cannot be accomplished, background or low priority tasks may be attempted to be eliminated. If these attempts fail the system executive will be called upon to reschedule and allocate the tasks throughout the entire computer system. In general the procedures followed throughout this process are quite similar to those presented in the previous section for the executive scheduler (Section 11.2.4).

During non-critical phases, time is available to reconfigure the system and in general the first solution, that of attempting to obtain an optimum assignment, will be followed. However, during the critical phases; time is generally not available to follow an optimum procedure (0.1 second is allowed to reconfigure the system) for reconfiguration. In this case the second solution will be followed; this will involve

switching over the output signals to a redundant set of critical programs being performed in the computer system.

11.2.6 Interface With System Executive

The overall computer system control is performed by the system executive. To ensure reliability and ensure the mission will be successful despite a system executive failure, the software design approach has been to minimize the system executive functions and distribute its functions where possible. In this way the group will continue to function if the system executive has failed for some reason. The group executive has some interface with the system executive. As stated above, this interface has been designed to be minimal. The functions are outlined here.

1. Task assignments. The tasks each group is to perform will be issued by the system executive. The system executive will have more information concerning the status of the groups and which groups have spare capacity. When a group can not perform a task for some reason, the system executive can be informed. The system executive may be able to re-assign the task to a group that has spare capacity sufficient to do the task.
2. Back-up Data. Critical tasks during the critical phases will be duplicated in separate groups. If one group fails, the remaining group will continue to operate. Usually it is advantageous to have one task check the results of the other tasks. This data will be sent from one group to the other groups that are concerned. Thus various groups can check each other, and all will have the latest data (some groups may not perform a periodic task as frequently as others). Back-up programs might not be performed unless the primary task fails. This back-up program should contain current data used and generated by the prime program. The back-up program should if be needed, is ready to perform using the current data.
3. I/O Conditioners. Many I/O conditioners are connected to two groups. Depending upon the hardware design, the system executive may be called upon to resolve conflicts that arise when the two groups each attempt to control the I/O conditioner. The system executive may require one group to not use a particular I/O conditioner, or to time share the device with the other group.
4. Data Transmission. The system executive will allocate time slots to groups so the groups may transfer data over the inter-group bus. The group executive will monitor its group switches as part of its periodic RTC service routine. It will pick up requests from other groups here. In addition, the group executive will place any requests it has for communications with another group in its group switches. The system executive will control the transfer of these requests between the groups.

11.3 SYSTEM EXECUTIVE

In addition to the group executive functions presented in the previous section, one of the group executives will also contain additional functions giving it the

capability of a system executive. The system executive is designed to coordinate the groups and not really to control them. By this means, the system reliability is increased.

Some of the functions of the system executive were introduced in Section 11.2.6 where the normal group executive interface with the system executive was discussed. This section will discuss those functions plus others that reside in the system executive (hereafter referred to as SE).

11.3.1 Task Assignments

The SE will contain a complete up-to-date list of the total hardware resources in the computer system. It will be responsible for deriving the software requirements table (Table 11-3) and sending it to each cell. This table essentially assigns the tasks throughout the computer system. The SE will retain a copy of this table for each group. This table is formed in the SE by using an executive scheduler similar to that described in the previous section. The prime difference here is that the executive function is more of a gross assignment routine rather than a detailed scheduler routine.

11.3.2 Control Inter-group Bus

The SE is responsible for control of the inter-group bus in much the same manner as the group executive is responsible for control of the inter-cell bus. The SE interfaces with the group switches and essentially this is similar to the SE acting like the group executive and the group switches looking like cells. The SE will follow a routine that is quite similar to the control system resources routine described in Section 11.2.1. It will essentially sample the group switches periodically by a RTC interrupt routine and also in a background manner by a poll routine. The prime difference between this one and the group executive is that more of the service on the inter-group bus is expected to be of background in nature.

The primary difference between the SE function here and the group executive is the sequence of commands that must be sent on the bus to accomplish the requested services. Section 10.2 discusses the inter-group bus operations and how they take place.

11.3.3 Reconfiguration

The SE will be forced to handle many cases of reconfiguration: phase changes, faulty assignment of tasks, and failures. The most difficult reconfigurations are due to the latter two cases, both are handled in a similar manner. The SE will enter its executive scheduler routine (obviously more complex than that described for the group executive) and proceed to re-assign tasks among the groups.

Some of the other SE functions will be mentioned here. The SE will be responsible for monitoring the group switches to check for a Go/No-Go signal. This signal informs the SE of the status of a group. The general functions of the SE if a No-Go signal is received were outlined in Section 8 of this report.

The SE will be responsible for communicating with the group executives to keep track of the hardware resources and the status of assigned tasks (completed, in process, etc.).

The SE will also be responsible for containing initialization routines to start up the groups. There will be basically two types of initialization routines the cold start and the start up after a failure. The first case occurs when a group is inactive such as in the pre Mars orbital phases and is required in a new phase such as Mars orbital. The latter case occurs when a group failure results as indicated by the group switches.

Some of the general concepts in the initialization of a group were introduced in Section 8. Assuming a cold start, one of the cells in the group that will contain the SE will have a connection via its I/O line to the display/control panel. This cell will be initialized by first sending it information to initialize its address (ID) and state (set to the CC state) register. Then the cell may be loaded from the bulk storage unit with the initialization program.

This program will first begin checking out the cells in the group one at a time. (The cell's ID and state registers will be initialized via neighbor lines.) Having determined the hardware resources for this group, the program will load in the system executive from the bulk storage unit. The SE will then notify the astronauts via the display panel that another group is to be started up. This results in a cell in another group being initialized in the same manner via its I/O line as done for the SE group. The SE will then control the loading of this cell in the group from the bulk storage unit. The cell will follow a similar procedure in checking out and initializing the cells in its group. However, all loading will be done under control of the SE group via the inter-group bus. When the group has been completely checked out the group executive of this group will inform the SE of the status of the hardware resources (any failed cells).

Following this report the SE will proceed to the next group in the same manner until all the groups have been initialized and checked out. The SE will then have a complete list of the hardware resources available. It will then proceed to assign the tasks to the groups and send each group its assignment table. The group executives will then control the final loading of the operational programs into the groups.

Start up after failures, in particular group failures, were generally discussed in Section 8 and are quite similar in procedure to that given above. This section should be referred to for some of the specific formalities for starting up after failures.

11.4 CELL EXECUTIVE

Each cell in a group will require a cell executive routine. This routine will vary in size and complexity from cell to cell depending on the specific tasks to be performed in each cell. In any case the cell executive (hereafter called CE) must be a small, minimal program. The reason is that it is repeated so many times in the group and the system would be very inefficient if each cell required for example hundreds of words for an executive.

A way for minimizing CE requirements is to organize programs into sets of cells and have the bulk of the executive functions in one cell of this set and very minimal executive functions in the other cells of the set. This was done in an example program that will be presented in Appendix A of this report. In the example program 3 cells were in the set for the particular program and only 1 of the cells had most of the executive functions such as: all interface with the controller cell, all error checking of intercell bus transmissions, and all data error and program exception processing.

The minimum cell executive requirements will be given below; every cell will be required to contain these functions. The storage requirements for the minimum executive are:

Location 0	Save location for MB register
Location 1	Save location for P
Location 2	Save location for L
Location 3	GC X_2 response request
Location 4	GC X_2 response request

Location 0-2 are required in case the controller cell forces the cell to do some function via a command addressed to this cell, such as during software testing. These locations allow the program that was interrupted to be resumed.

Locations 3 and 4 are required to respond to a GC command sent over the inter-cell bus by the controller cell that requests a communications request word back. These locations are required even if the cell has no requests to be made.

Most cell executives will require a little more in the way of a program to take care of the cell processing problems. Some of these problems may be allowed interrupts by the programmer: The interrupts will be discussed below. The following are possible interrupts in a cell:

1. Machine errors
2. Inter-cell bus I/O interrupt
3. Illegal data such as attempted division by zero
4. Illegal operation such as an invalid operation code, no neighbor response, etc.
5. Real time clock goes to zero

Any of the above interrupts may be masked off. Each interrupt will be briefly discussed below to provide some idea of what is required of the cell executive to provide servicing the interrupts.

Machine error interrupts are handled by a simple routine. An example of such a routine is given in Figure 11-13. Block 1 is the normal hardware saving and changing of the program counter. In Block 2, the two locations that hold the GC X_2 response request words are loaded with the request. The request will be for an output of a fixed number of words to the controller cell as seen by the description of the X_2 storage area. The location following the second X_2 word is loaded with the address in this cell where the words will be output from. The cell will then mask off all interrupts and wait for the controller cell to send this cell a GC command requesting the X_2 response request words.

One of the prime functions of the cell executive will be handling the inter-cell bus communications. The inter-cell bus I/O routine is shown in Figure 11-14. As may be seen part of the functions are accomplished by hardware and part by software. When I/O over the inter-cell bus is desired, the two dedicated locations that store the X_2 response request words are loaded with the desired communication information. The program will then return to its problem or execute a wait instruction if the problem cannot proceed until the communication is accomplished (this would normally be inputting some data from another cell).

When the controller cell sends this cell a GC X_2 command (described in Section 11.2.1), the cell will respond as shown by the hardware blocks. The detailed actions that occur to implement the functions described in these blocks was given in Section 10 of this report. However, Figure 11-14 shows the general functions and the interaction between the software and hardware. The interrupt when the GC X_2 command is received is not really a true interrupt. It simply halts the execution of

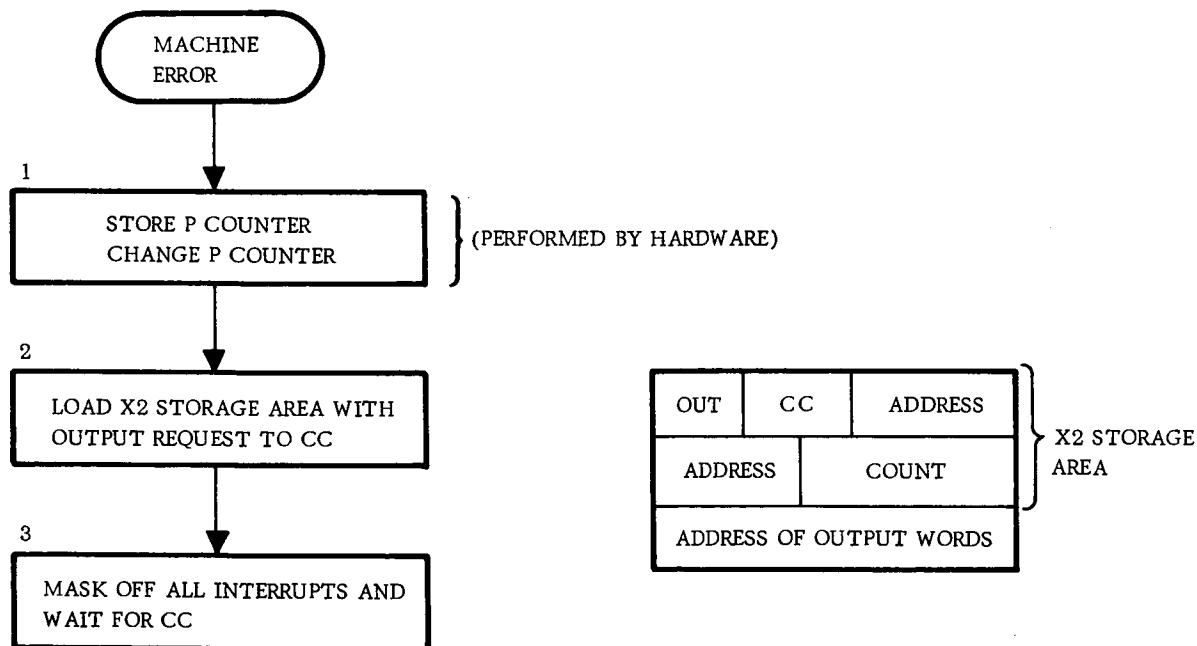


Figure 11-13. Machine Error Routine

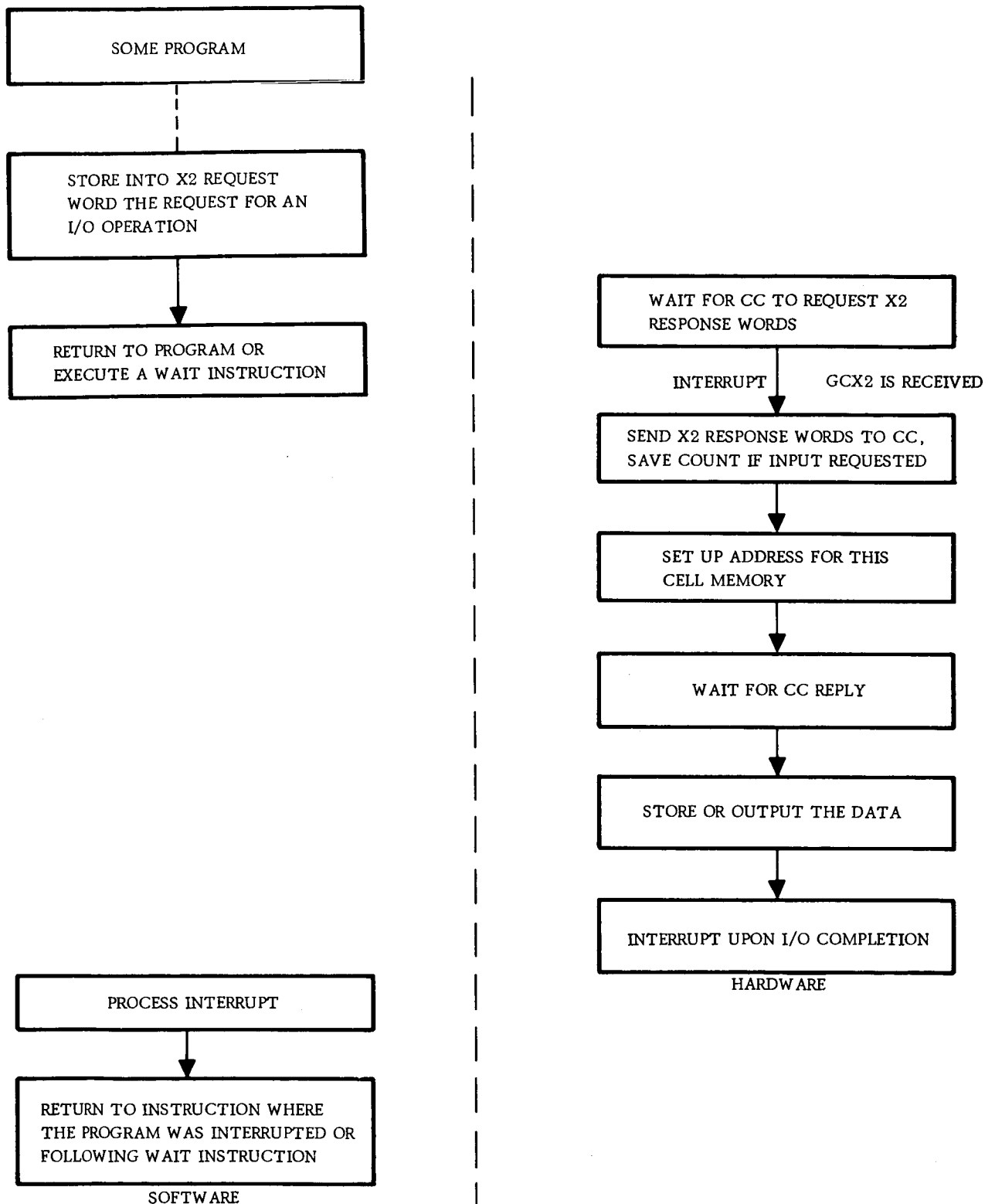


Figure 11-14. Inter-Cell Bus I/O Routine

the present instruction and forces the hardware actions to be completed. It will essentially release the processor to complete the present instruction and continue with the halted program in the block "Wait for CC reply." When the CC reply is received a similar halt interrupt occurs until the communication is completed. The communication is completed when: the count in the accumulator equals 0, the time interval between data words coming in from the inter-cell bus exceeds some value, or the transmitting cell sends out an end of transmission command. Then an I/O complete interrupt occurs. The block "process interrupt" is expanded upon in Figure 11-15.

The interrupt will enter block 1 in Figure 11-15 by executing the instruction in the memory location dedicated to processing this interrupt. This interrupt will be entered when the accumulator is counted down to 0, the time interval between data words has exceeded that allowed by a hardware timer or a command "end of transmission" is received and decoded. If there are errors such as parity, then the interrupt will occur early, i. e. before the count has reached 0 or the timer has signaled end of data.

The count is checked to see if it is 0. This is first accomplished by seeing if the accumulator equals 0. If the operation was to input words and the accumulator is 0 then all the words expected were received. (Note that the controller cell could tell another cell to communicate less data than requested by the requesting cell as explained in Section 11.2.1.) For an input operation this is all that is needed, to check for 0 in the accumulator, since the cell will save the count it requested in the accumulator and count it down each time a data word is received. If it is not 0, then the address in the program counter is compared to the initial address (given in the location following the second X_2 response word) and the difference noted. This will indicate how many words were received and the cell can then form a new X_2 response word to try to get the remainder of the data on the next GC X_2 received from the controller cell.

If an output operation was being performed, the procedure is slightly different. In this case the count that is placed in the accumulator is received from the controller cell. This should be zero at the end of transmission otherwise an error exists. Therefore the program counter will be compared with the initial address given in the same location as discussed above and the difference noted. This difference will be compared to the count in the X_2 response word location and any discrepancies noted. If they compare, all the words desired were output. If not, then a new X_2 response word is formed as above.

If all the words were transferred, Block 2 is entered and the response word location set to zeros. The X_2 response word is set to zeros so this request is not repeated. The branch to the requestor or a return to the interrupted location may be made. The X_2 response word being zero may be tested by other software routines to verify the data transfer was completed and the interrupt processing routine was entered and executed correctly. Block 4 will try to get the data again if there were errors. Since the X_2 is still in the same location, the next time the controller cell polls the cells, the same request will be made. If some words were transferred and not others, the address would be modified and the request made only for the data words that are missing. Block 6 will be entered if the retry was not successful.

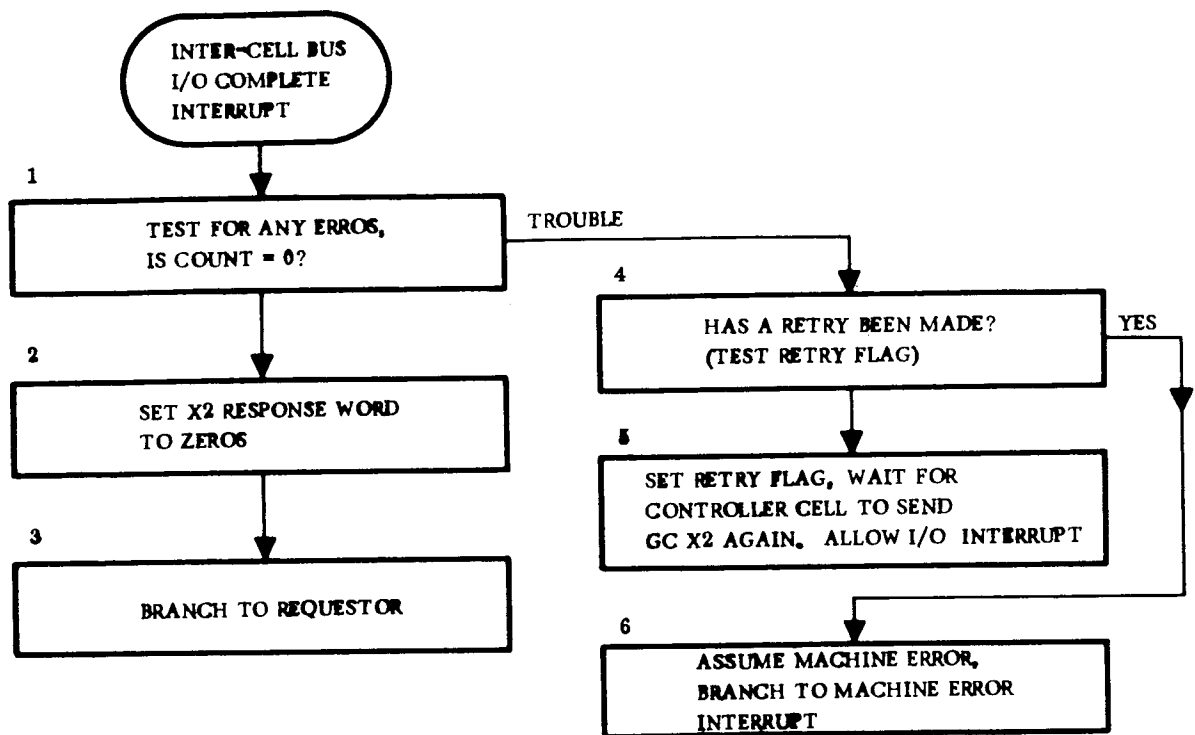


Figure 11-15. Inter-Cell Bus I/O Complete Routine

A machine (hardware) error will be assumed. The next transmission to the controller cell will request the machine error routine.

Finally for another program within the cell to request some I/O on the inter-cell bus, the program will test the X₂ location, if it is zero, the program will load the X₂ request locations with the desired address and count. The branch address is stored in a location known to the interrupt processing routine.

Interrupts three and four are concerned with the operation of the program. In correctly operating software, these errors may not exist. They can be handled like the machine error, sharing most of the instructions. Thus only three additional instructions and a data word will be needed.

Sometimes the data values may vary radically and overflows may be expected. In this case, the cell should contain its own interrupt processing routine. Conversely, the cell could send its problem to a neighbor cell or even to the controller cell. The time and space requirements all play a roll in defining how the software should be developed.

The routine to process the data will be similar to present programming for large computers. Essentially the programmer tells the routine what to do whenever an error occurs. Thus on an overflow error, the programmer may cause the interrupt routine to modify shift instructions and repeat the operation.

Illegal operations are usually assumed to be caused by a software error. Usually these are treated as a machine error, i.e., the controller cell error correcting routine is called to attempt to correct the bad program. The Flow diagram in Figure 11-13 will perform this operation.

The last interrupt, the real time clock going to zero, is either expected and the clock is set correctly, or the clock is not used and the interrupt is always masked off. When the clock is wanted, the flow diagram in Figure 11-16 shows a real time interrupting routine. Block 1 of Figure 11-16 stores the program counter and branches to the dedicated memory location for the real time clock.

Block 2 resets the clock, calculating the time interval from the present contents of the RTC, the time the clock was set and the desired interval to the next interrupt time. The branch address is updated so the next requestor may be branched to.

If there are more than 2 times, a pointer to indicate which is the next time for the clock needs to be updated. The complexity of the list depends upon the system software requirements. A scheduling of "random" time intervals will require a push-down list technique or similar software programming technique.

In all cases, the interrupt has been processed, the time for the clock when the next RTC interrupt occurs is ready. The branch to the proper routine that needs processing at this time is made. If the RTC interrupt has not yet been allowed, it is allowed at this time.

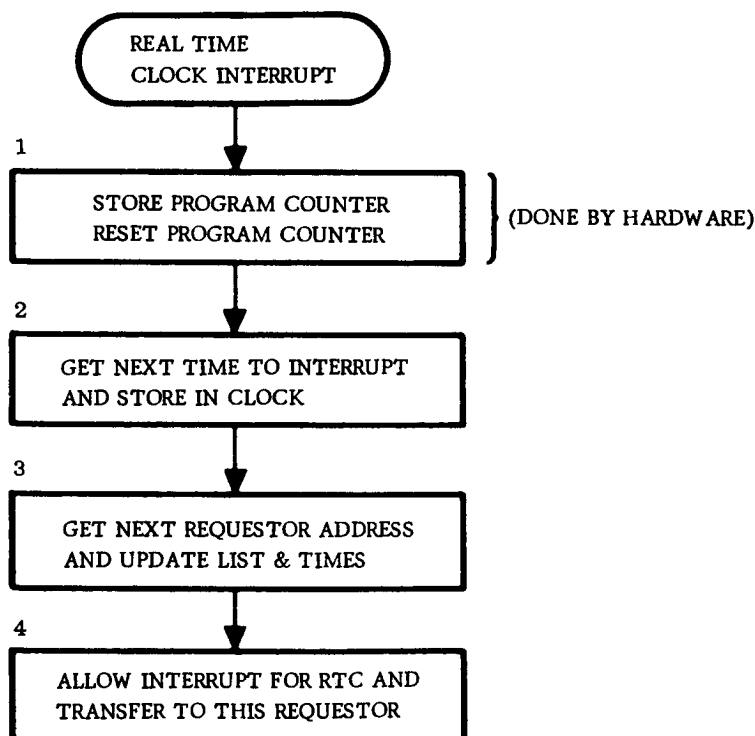


Figure 11-16. Cell Executive Real Time Clock Routine

The cell executive is seen to be as small as possible. Only the common, frequently used routines are in the memory. Error correcting procedures and complicated testing programs that will be used infrequently are all kept in the controller cell. The programmer will make trade-offs between the amount of executive required in the cell and the speed requirements. Obviously, it is slower to ask the controller cell to perform tasks than to do the same task by having the program in the same cell.

One point that should be brought out here is that there will exist two types of cells that will require quite different executive routines, these are the operational cells and the I/O cells. Operational cells are those actually performing the tasks and I/O cells have the connections to the sensor/conditioners via their I/O lines. The I/O cells will be primarily concerned with inputting/outputting information over the inter-cell bus and the I/O line.

Basically both cells will contain the same fundamental executive ideas introduced in this section. However, the I/O cell will contain considerably larger lists of tables and data handling routines. Its principal program in fact will be that of communicating over the inter-cell bus whereas in an operational cell this is more of an executive routine to handle the inter-cell bus communications.

12. RELIABILITY SIMULATION

12.1 MONTE CARLO METHOD

12.1.1 Introduction

The distributed processor organization was subject to a reliability simulation for a manned Mars mission application. A Monte Carlo reliability analysis program was used for the simulation. This program generates reliability and availability data based on simulating the organization.

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.

$P_f(t) = 1 - e^{-\lambda t}$ where λ 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.

12.1.2 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 susceptible 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:

$$P_f(t) = 1 - e^{\left(\frac{-\lambda t}{\alpha}\right)}$$

where α is the constant of proportionality.

A block diagram of the Monte Carlo program is given in Figure 12-1. The program first generates the time to failure for each piece of equipment. Equipment is divided up into two basic types, prime and secondary, prime equipment is that required to keep the computer operating while secondary equipment is that required to replace prime equipment in the event of failure. Prime equipment will have its statistics generated by an active failure rate while secondary equipment may be in the active or idle failure rate mode. Equipment may also change between active and idle modes depending on which phase of the mission is being considered. This will become clearer when the model for each phase of the mission is presented below.

Continuing with the operation of the program, it then checks if any of the times to failure of prime 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. An undetected failure can result in a mission failure if it occurs during a critical phase. Undetected failures that occur in other phases are simply disregarded here and assumed to be detected. This is a simplification of the actual case, namely that the probability of detection will approach 1.0 as time increases from the occurrence of the undetected failure. The effects of this during mission phases that are non critical will be to lower the computer system availability. The error in assuming the undetected failure is immediately detected will be trivial since these mission phases are on the order of thousands of hours and the undetected failure should be detected in less than approximately 10 hours.

The failed prime equipment will be replaced only if a suitable secondary piece of equipment is available. This requires generating a time to failure for the secondary equipment and also a detection probability if this time to failure is less than the length of the first phase of the mission.

The program then changes to the next phase of the mission and follows the same procedure for the computer configuration for the new phase. Of course if a mission failure occurs in a critical phase the program is terminated for this mission.

The program collects the statistics for each phase. Critical mission phases are primarily concerned with mission failure (probability of success) and the program will record these failures. Other phases of the mission are concerned with availability of the computer system. For these phases the percentage of the prime equipment that is operating will be recorded.

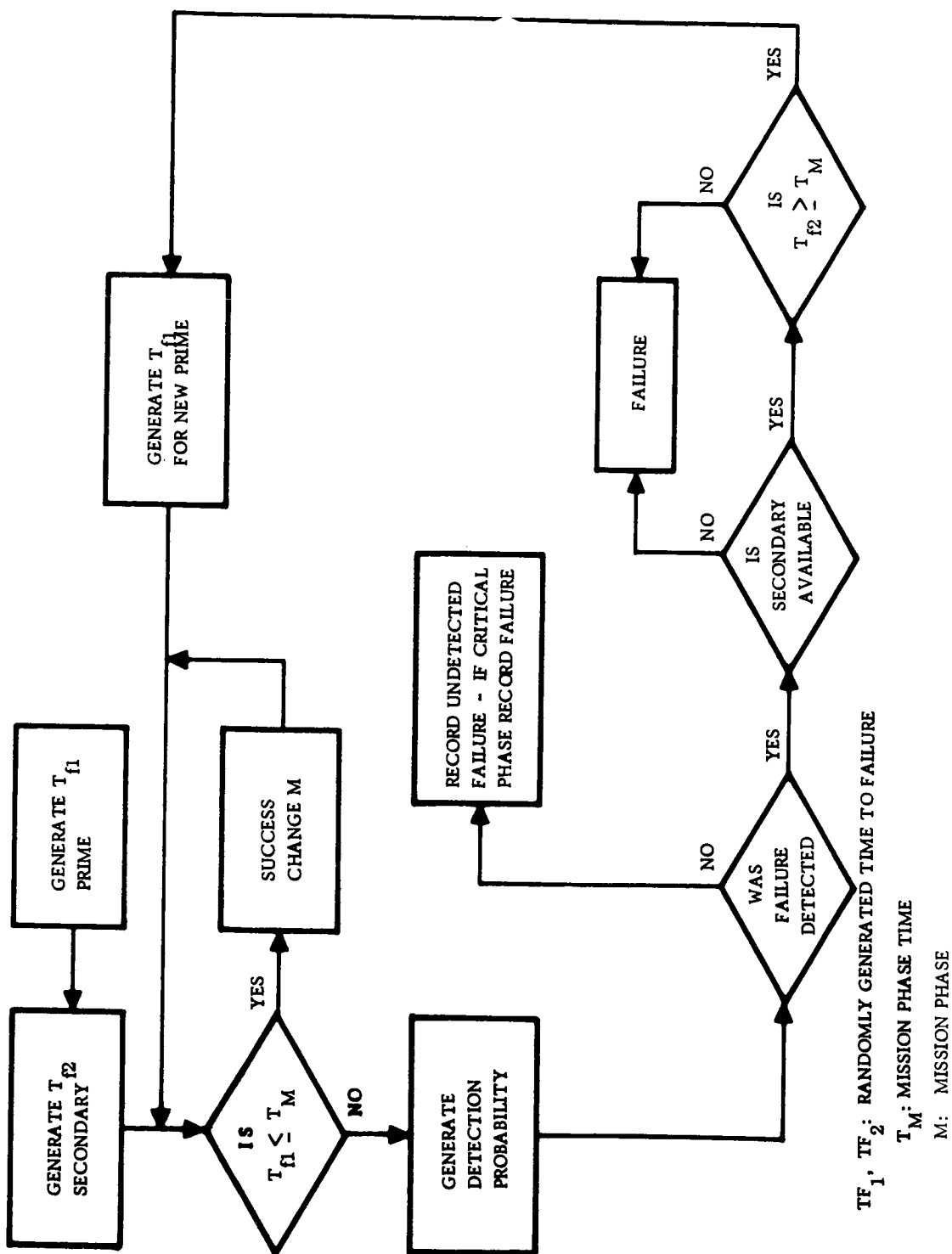


Figure 12-1. Block Diagram of Monte Carlo Simulation

When all the phases of the mission have been completed, the process is again repeated. By repeating this process, statistics are collected for probability of success and availability. As more and more runs are made the error in the results decreases, appendix B gives curves of the error in Monte Carlo simulations.

12.2 SIMULATION MODELS

This section will present the actual models used in the Monte Carlo program. The mission was divided into five basic phases from the twenty given in reference 1. This was done to reduce the running time of the program. These phases are given below:

Table 12-1. Mission Phases

Phase	Hours	Mission Time (hours from t_0)
Non Critical	2962.8	2962.8
Critical	1	2963.8
Mars Orbital	974.3	3938.1
Non Critical	6242.6	10,180.7
Critical	0.2	10,180.9

The first Non Critical phase consists of the first 9 mission phases listed in Section 2 namely Atmospheric Ascent through Mars Approach Correction; the first critical phase consists of phases 10 and 11, namely Mars Aerobraking and Mars Orbit Injection; the Mars Orbital phase is simply phase 12; the last non critical phase consists of phases 13 thru 19, namely Trans Earth Injection thru Earth Approach Correction; and the last critical phase is phase 20, Earth Re-Entry. Each of the three basic types of phases, Non Critical, Critical, and Mars Orbital results in a different computer configuration as will be presented below.

12.2.1 Mars Orbital Phase

The configuration for this phase consists of 4 groups turned on with 18 cells required per group as shown below in Figure 12-2. Spare cells, group switches and cell busses are provided for in each group. Spare groups are also provided for. One identical failure rate will be assumed for each of the cells and cells which are in groups that are off will have the failure rate of the cells in groups that are on, multiplied by a factor ($\lambda_{ON}/\lambda_{OFF}$). If a group switch or a cell bus is not available in a group then this group is considered as having totally failed. Assuming a group switch and a cell bus are operating in a group, the number of cells that are operating in a group is the primary statistic of interest. This will give an indication of the availability of the computer system when all the groups are considered. The group bus is quite important to efficient operation of the groups. However, a group could still continue to function in the absence of a group bus. Therefore, if no group bus is

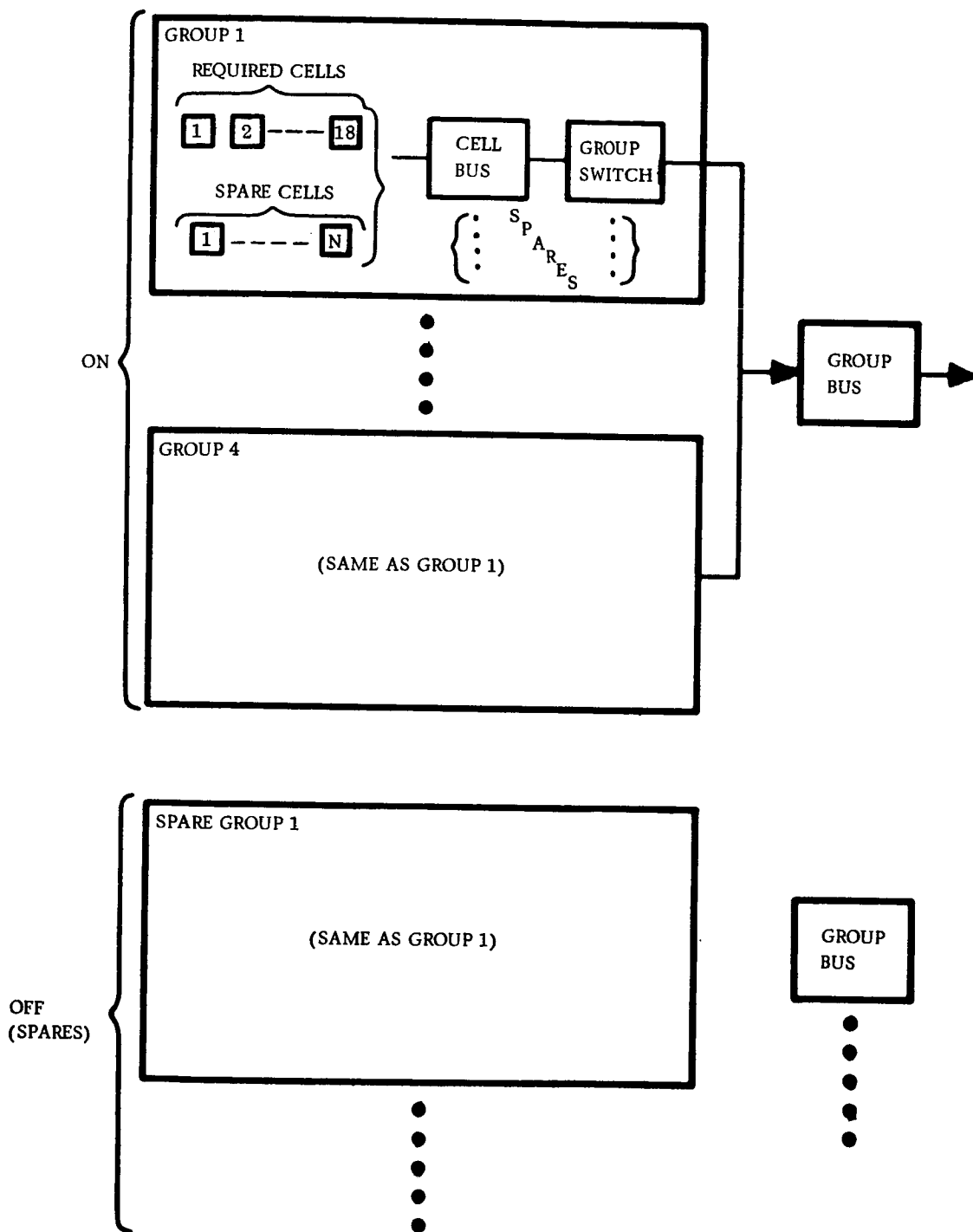


Figure 12-2. Mars Orbital Configuration

available the total percentage of cells operating will simply be reduced by some fraction. This effectively degrades the availability of the computer in the absence of a group bus as would be the case.

12.2.2 Non-Critical Phase

The configuration for this phase is identical to that for the Mars orbital phase except that only three groups will be required as shown in Figure 12-3. It should be noted that in this phase group 4 is considered a spare group. Availability is the prime concern here and the same comments as for the Mars-Orbital Phase apply here.

12.2.3 Critical Phase

The critical phase is primarily concerned with mission success or failure. This is determined by whether or not a critical navigation and guidance program can be successfully run on the computer and correct outputs sent to the critical sensors. The critical phases are primarily the planetary entry phases. It is during these phases that failure of the computer may result in destruction of the vehicle and a resultant mission failure. There are other phases during which one requires navigation and guidance computations that may be considered very important such as midcourse maneuver phases. However, failures here may simply result in the full mission not being accomplished due to the expenditure of too much fuel, etc. These effects can not be classified as availability or probability of success. However, they do affect the probability that the total information desired from the complete mission will be obtained. This is a systems analysis type of effort that is relatively important, however it is beyond the present scope of the study. (The availability determined in this study will also have affects on this probability of achieving the total information desired.)

The distributed processor offers many internal paths around which it may be reconfigured due to failures as pointed out in Section 8. Therefore, the configurations to determine whether or not the computer has failed in a critical phase are very complex. Three configurations have been defined for a critical phase for this simulation as shown in Figure 12-4. The first configuration is the one used if a group bus is available during the critical phase. Spare cells can replace failed cells in the groups and likewise for the other components. It should be noted that no spare groups are applicable in this phase. This is due to the fact that one would generally not have time to load up and get these groups running during the relatively short critical phase. Another point to be noted is that three of the operating cells in each group, say cells 1, 2 and 3, have been selected as non replaceable in case of failure by a spare cell. This is due to the fact that these cells may require I/O connections etc. to the outside world and it is assumed these would not be altered during the critical phase. A group is defined as operating if and only if it has 16 operating cells, a cell bus, and a group switch. Only one group is required to be operating during the critical phase for a successful critical phase (the critical navigation and guidance function should fit into one group). To take advantage of all the sparing outlined above, the group bus is needed.

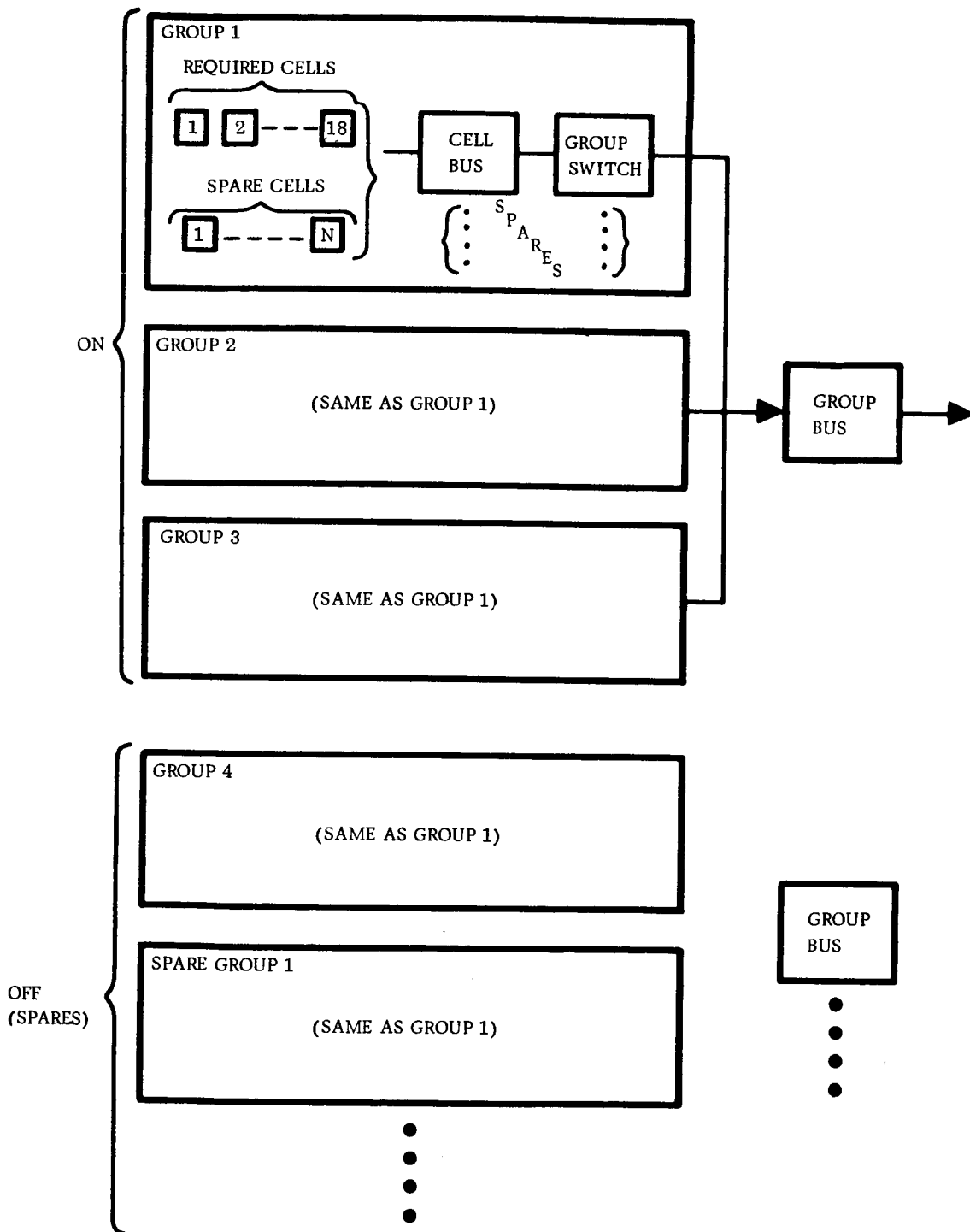
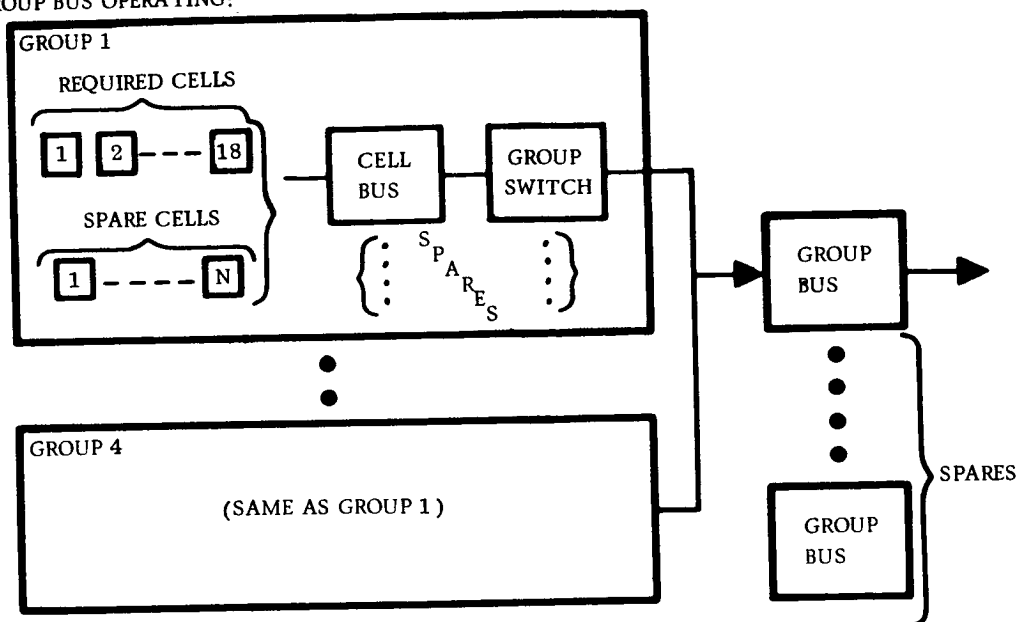
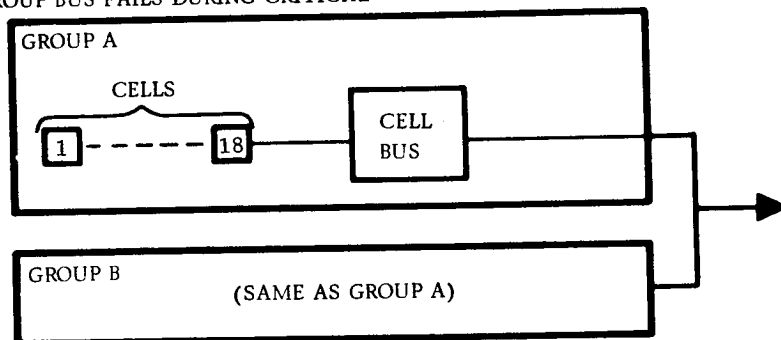


Figure 12-3. Non-Critical Phase Configuration

A) GROUP BUS OPERATING:



B) GROUP BUS FAILS DURING CRITICAL PHASE:



C) GROUP BUS FAILS BEFORE CRITICAL PHASE

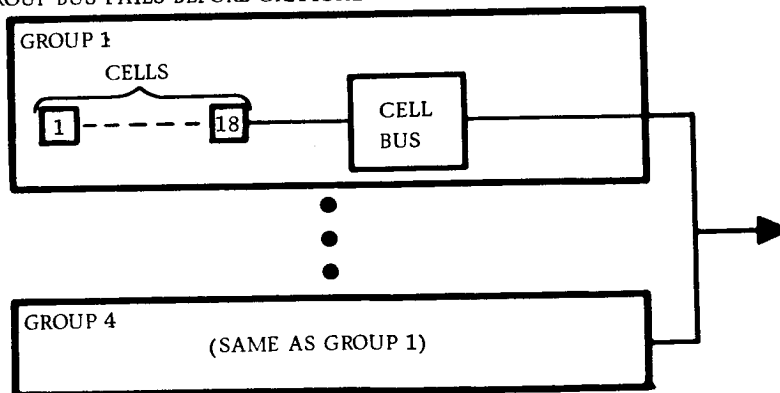


Figure 12-4. Critical Phase Configuration

If a group bus is not available a new configuration is assumed. Two cases can occur here. The bus fails during the critical phase or before it. If a bus is not available during the phase and was available before the phase was entered (degeneration of configuration a in Figure 12-4) then the configuration shown in Figure 12-4-b is assumed. As can be seen from this figure, one of the two groups needs to be working for a successful critical phase. However, note that no replacement of failed cells is possible by spare cells in a group.

Finally, if before entering a critical phase, no group bus is available, the configuration in Figure 12-4-C would be assumed. This is similar to Figure 12-4-b in that only one group is needed to have a successful critical phase and no replacement of failed cells is possible. However in this case 4 groups are used. This implies that all 4 groups are loaded with the same programs and the outputs switched.

It should again be noted that undetected failures are also considered in the critical phases. An undetected failure in the group that is being used for output to the critical sensors will result in immediate failure of this critical phase. However, an undetected failure in other groups will not result in a failure of the critical phase unless this group is brought in to replace the group outputting to the critical sensors.

12.3 SIMULATION RESULTS

The results of the Monte Carlo simulation will be presented in this section. Numerous cases were simulated varying parameters such as, cell and group switch failure rate, number of spare cells per group, number of spare inter-group busses and group switches, number of spare inter-cell busses, on/off failure rate ratio, and percent of failures affecting the busses. A listing and definition of the parameters that could be varied in the Monte Carlo program is given below:

MTBF	:	Each cell and each group switch failure rate
N	:	Number of spare cells in each group (number of cells exceeding the 18 required)
M	:	Number of spare groups
K	:	Number of spare inter-group busses (also the number of spare group switches per group)
Y	:	Number of spare inter-cell busses per group
PD	:	Probability of Detecting a Failure
PB	:	Probability that a cell or group switch failure results in a failure of a bus
O/O	:	MTBF ON/MTBF OFF Ratio

A total of 23 different cases were simulated. An example of the print out of one case is shown in Figure 12-5. Examination of this figure will give more understanding as to the operation of the program. The first two columns contain the phase and time at which the statistics are accumulated. The program starts at $t = 0$ hours and

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING (M=0, N=1, K=1, Y=1, PD=.95, MTBF=200,000)

CASE 8 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99913		0.159	0.987	0.114	0.010	0.001	
CR-2	2963.8		1.00000	0.001	0.001	0.0	0.0	0.0	0.0
MO-3	3600.0	0.99795		0.038	0.244	0.024	0.002	0.0	
MO-4	3938.1	0.99730		0.031	0.130	0.014	0.003	0.0	
NC-5	4500.0	0.99603		0.032	0.172	0.018	0.0	0.0	
NC-6	6500.0	0.99349		0.106	0.639	0.061	0.008	0.002	
NC-7	8500.0	0.98911		0.106	0.609	0.055	0.009	0.001	
NC-8	10180.7	0.98445		0.081	0.534	0.057	0.007	0.001	
CR-9	10180.9		1.00000	0.0	0.001	0.0	0.0	0.0	0.0

Figure 12-5. Example of Computer Print Out

examines the computer configuration at $t = 2962.8$ hours, 2963.8 hours, etc. This procedure was described in Section 12.1. The next column contains the cell availability. This is the average number of cells available (good) in the computer system (Number Available/Number actually required to meet the requirements of the particular phase) for the time interval just evaluated. For example, at $t = 8500$ hours, a cell availability of 0.98911 is the average availability from $t = 6500$ to $t = 8500$ hours.

Cell availability is not of interest during a critical phase and is not calculated at phase CR-2 and CR-9. In its place probability of success is calculated. The conditions on probability of success were defined in section 12.2.

In addition, statistics on the cells and bus failures were printed out. Cells 1, 2, and 3 were defined as having external connections and were therefore considered apart from the others in a group. The average number of these cells failing during each time interval is printed out in column 5. It should be pointed out here that a total of 1000 runs were made for each case and the number printed out in this column is the average of the 1000 runs. For example, for the first time interval, $t = 0$ hours to $t = 2962.8$ hours a total of 159 of these cells failed. Therefore, the average number of cells 1, 2, and 3 failing is 0.159. It should be noted that this is the total number of cells 1, 2, and 3 in the computer system, each group contains 1 cell 1, 1 cell 2, and 1 cell 3.

Columns 6 and 7 give the total number of cell and group switch failures respectively. Columns 8 and 9 contain the statistics for the inter-cell (IC) and inter-group (IG) bus failures respectively. Undetected failures are considered in the critical phases and the last column gives the total number of these occurring.

The results from the simulation are summarized below in Table 12-2. The print out for each of the cases may be found in appendix C.

It should be noted that the statistics for cell, group switch, and bus failures are tabulated for the entire mission and not each phase as in the print outs. Also availability is the average over the total mission time. A word of caution should be mentioned when reading the table; errors are inherent in Monte Carlo simulations and the curve in Appendix B should be referred to. This will give one an indication how far to extrapolate the results.

Effects of N - Number of Spare Cells Per Group

Figure 12-6 shows a curve of the average availability over the entire mission as a function of the number of spare cells in each group. The conditions as shown are for an MTBF of 200,000 hours for each cell and group switch, 1 group bus, and 1 cell bus per group. The average availability is 97.25% with no spares and rises to 99.6% with two spares per group.

It should be noted that for this and all the curves in this section the following conditions hold unless otherwise specified: M (Spare groups) = 0, $MTBF-OFF/MTBF-ON = 2.0$, PB (probability of a failure affecting a bus) = 0.5 percent.

Table 12-2. Simulation Results

Case	MTBF (Hours) (On)	N	K	Y	Avg. Avail.	Prob. of Success	Cell 1,2,3 Failures	Total Cell Failures	Group Switch Failures	IC Bus Failures	IG Bus Failures
All Cases: Probability of Detecting a Failure = 95%, M = 0											
Cases 2 - 17: MTBF ON/MTBF OFF = 2.0, PB = 0.5%											
2	75,000	1	1	0	0.9666	0.999	1.451	8.509	0.868	0.049	0.012
3	200,000	1	1	0	0.9937	1.000	0.559	3.296	0.335	0.020	0.007
4	500,000	1	1	0	0.9986	1.000	0.215	1.381	0.136	0.011	0.001
5	75,000	1	0	0	0.9580	1.000	1.327	8.236	0.448	0.038	0.006
6	200,000	1	0	0	0.9917	1.000	0.549	3.320	0.182	0.011	0.001
7	500,000	1	0	0	0.9981	1.000	0.219	1.327	0.073	0.006	0.001
8	200,000	1	1	1	0.9933	1.000	0.554	3.317	0.343	0.039	0.005
10	75,000	0	0	0	0.9276	0.990	1.403	7.778	0.475	0.032	0.002
11	200,000	0	0	0	0.9727	1.000	0.548	3.154	0.168	0.017	0.000
12	500,000	0	0	0	0.9893	1.000	0.231	1.269	0.075	0.011	0.000
13	75,000	0	0	1	0.9298	0.997	1.382	7.820	0.453	0.082	0.004
9	200,000	0	0	1	0.9737	1.000	0.535	3.140	0.177	0.032	0.002
14	500,000	0	0	1	0.9900	1.000	0.196	1.212	0.081	0.014	0.001
15	75,000	0	1	0	0.9345	0.988	1.542	8.131	0.890	0.054	0.010
16	200,000	0	1	0	0.9750	1.000	0.537	3.171	0.328	0.022	0.004
17	500,000	0	1	0	0.9901	1.000	0.208	1.274	0.123	0.007	0.000

Table 12-2. (Cont)

Case	MTBF (Hours) (On)	N	K	Y	Avg. Avail.	Prob. Of Success	Cell 1, 2, 3 Failures	Total Cell Failures	Group Switch Failures	IC Bus Failures	IG Bus Failures
Cases 18 - 21 MTBF ON/MTBF OFF Varied											
18 ON/OFF=1	75,000	0	1	0	0.9347	0.967	1.571	8.976	0.978	0.053	0.014
15 ON/OFF=2	75,000	0	1	0	0.9345	0.988	1.542	8.131	0.890	0.054	0.010
20 ON/OFF=5	75,000	0	1	0	0.9365	0.998	1.273	7.426	0.822	0.034	0.013
19 ON/OFF=1	200,000	0	1	0	0.9746	1.000	0.638	3.649	0.394	0.016	0.006
16	200,000	0	1	0	0.9750	1.000	0.537	3.171	0.328	0.022	0.004
21	200,000	0	1	0	0.9742	1.000	0.485	3.009	0.326	0.014	0.005
Case 22 & 23 PB Varied, MTBF ON/MTBF OFF = 2.0											
16 PB=0.5%	200,000	0	1	0	0.9750	1.000					
22 PB=1.0%	200,000	0	1	0	0.9752	1.000	0.527	3.145	0.342	0.040	0.005
23 PB=2.0%	200,000	0	1	0	0.9747	1.000	0.573	3.251	0.366	0.059	0.012
11	200,000	0	0	0	0.9727	1.000					
6	200,000	1	0	0	0.9917	1.000					
24	200,000	2	0	0	0.9962	1.000	0.526	3.528	0.178	0.026	0.001

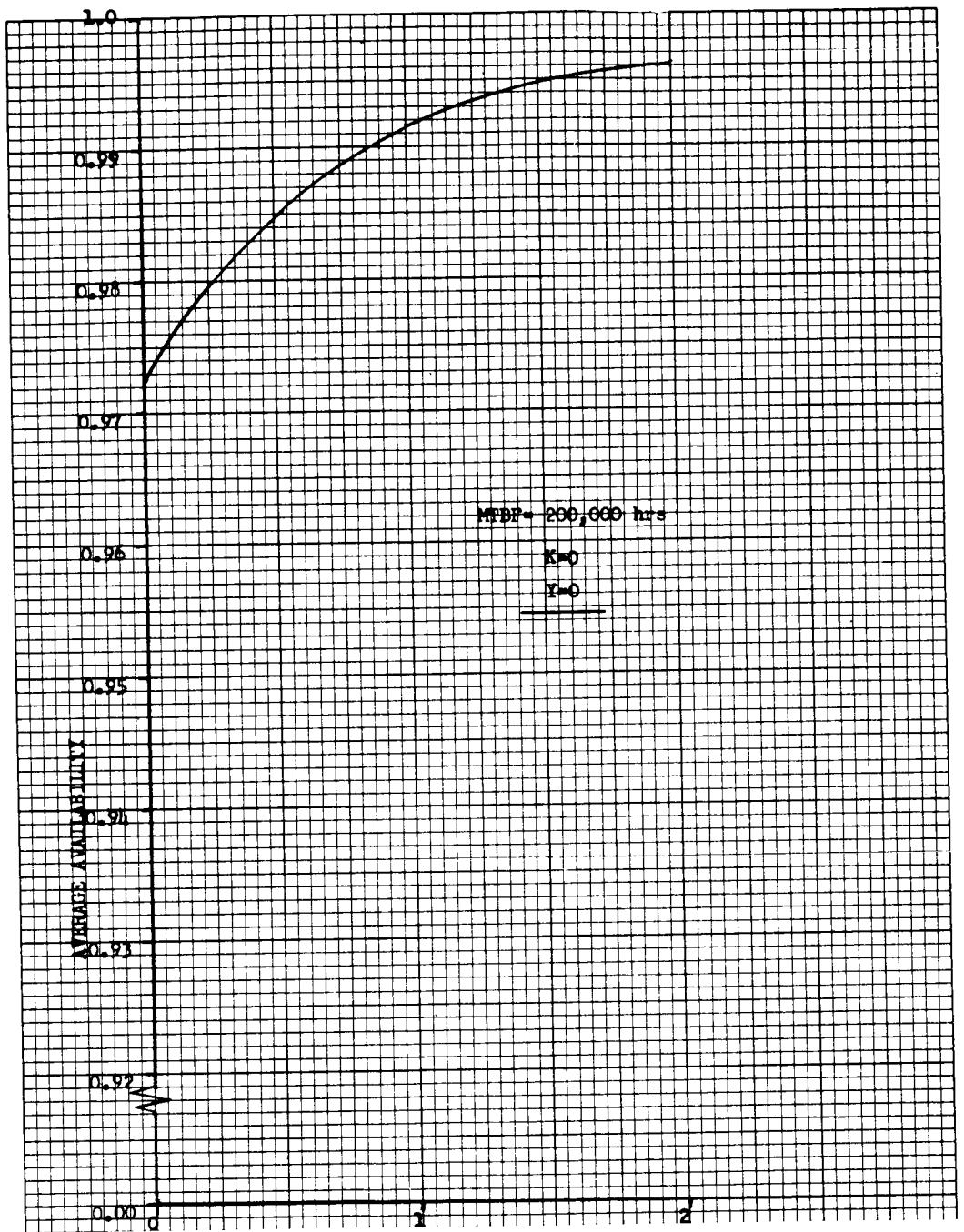


Figure 12-6. Average Availability vs Number of Spare Cells Per Group

Effects of MTBF

The effects of cell and group switch MTBF (on) on average availability are shown in Figure 12-7. These effects are shown parametrically as a function of the number of spare cells per group. The conditions on the curves are no spare busses in the system. The curve for $N=2$ is dotted between 75,000 hours and 200,000 hours since no point was actually obtained at 75,000 hours. As may be expected, MTBF has a significant effect on availability and is most pronounced for values up to 200,000 - 300,000 hours. In addition, the improvements due to adding spare cells are most pronounced when adding the first spare cell.

Effects of K - Inter-Group Busses

Average availability as a function of MTBF is shown in Figure 12-8. Curves are shown for $K=0$ and $K=1$ with both $N=0$ and $N=1$. In both cases a slight improvement in availability results by having one spare intergroup bus.

Effects of Y - Spare Inter-Cell Busses

Figure 12-9 shows the average availability as a function of MTBF when the number of inter-cell busses per group is varied. It can be seen that only a slight improvement in availability results by adding a spare inter-cell bus.

Effects of ON/OFF Failure Rate Ratio

The MTBF-ON ratio was varied from 1 to 5 and the effects on average availability is shown in Figure 12-10 for two values of MTBF-ON. As may be seen the availability did vary significantly. One of the factors that probably produced such a result is that three out of the four groups are always on and the fourth one on during the Mars Orbital phase therefore, most of the equipment is always on. However, it will be shown later that the ratio has an important effect on probability of success.

Effects of PB- Percent of Failures Effecting Busses

The percent of failures of a cell or group switch bringing down a bus was varied from 0.5 to 2.0 percent and the effects on average availability observed. The changes in availability were insignificant. It is felt that somewhat higher values of PB would have to be used before degradations in availability result.

Availability as a Function of Mission Time

It should be recalled that availability is a function of time during the mission and the curves presented previously only indicate the average availability. Figure 12-12 through 12-15 show the variation with time for four different conditions: (1) $N=0$, $K=0$, $Y=0$, (2) $N=0$, $K=1$, $Y=0$, (3) $N=1$, $K=0$, $Y=0$, (4) $N=1$, $K=1$, $Y=0$. In addition the curves for three values of MTBF are shown for each condition. It should be noted that the curves are plotted as straight line segments since the values represent average availability over fixed time intervals of the mission.

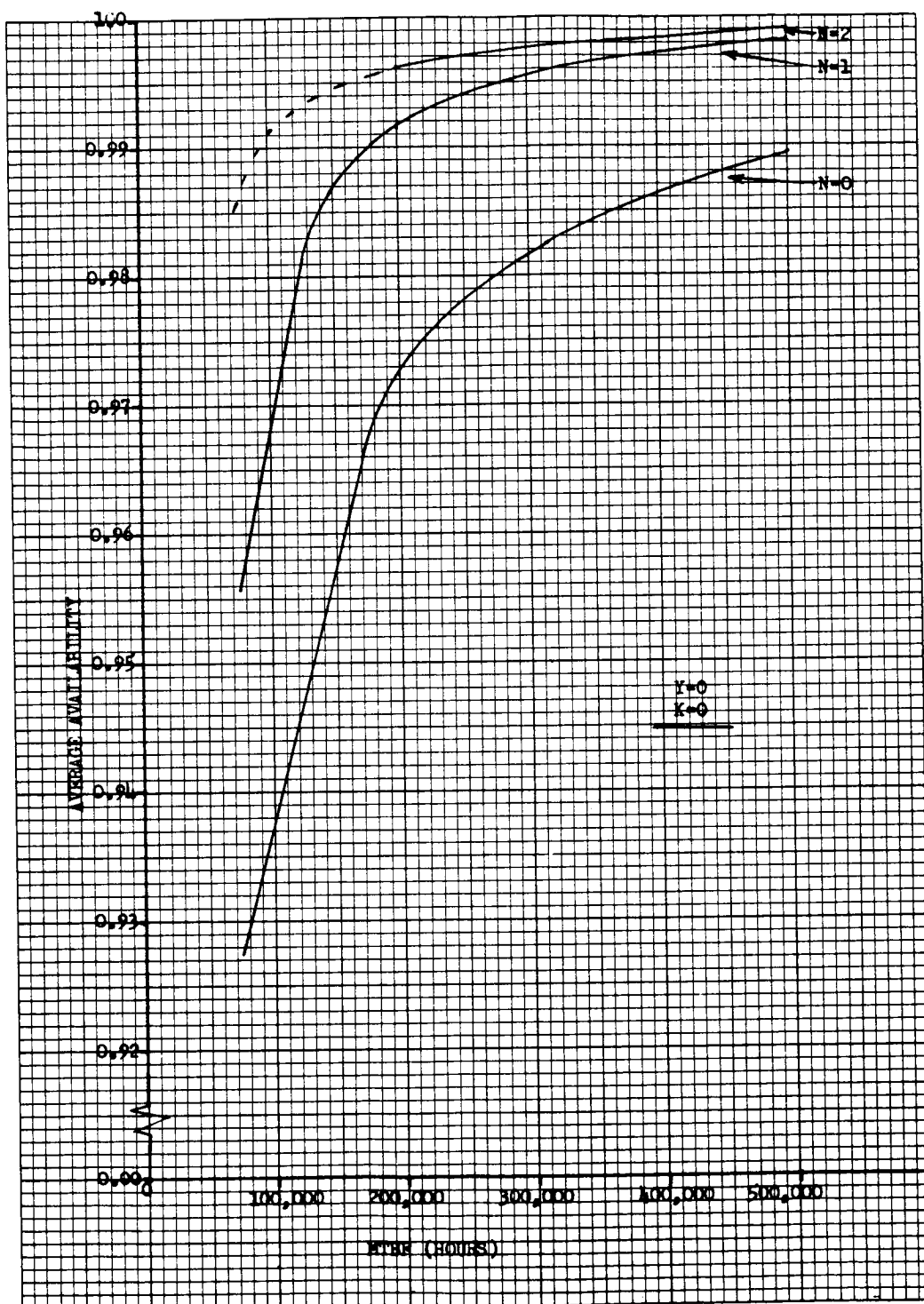


Figure 12-7. Average Availability vs Cell MTBF Varying Number of Spare Cells

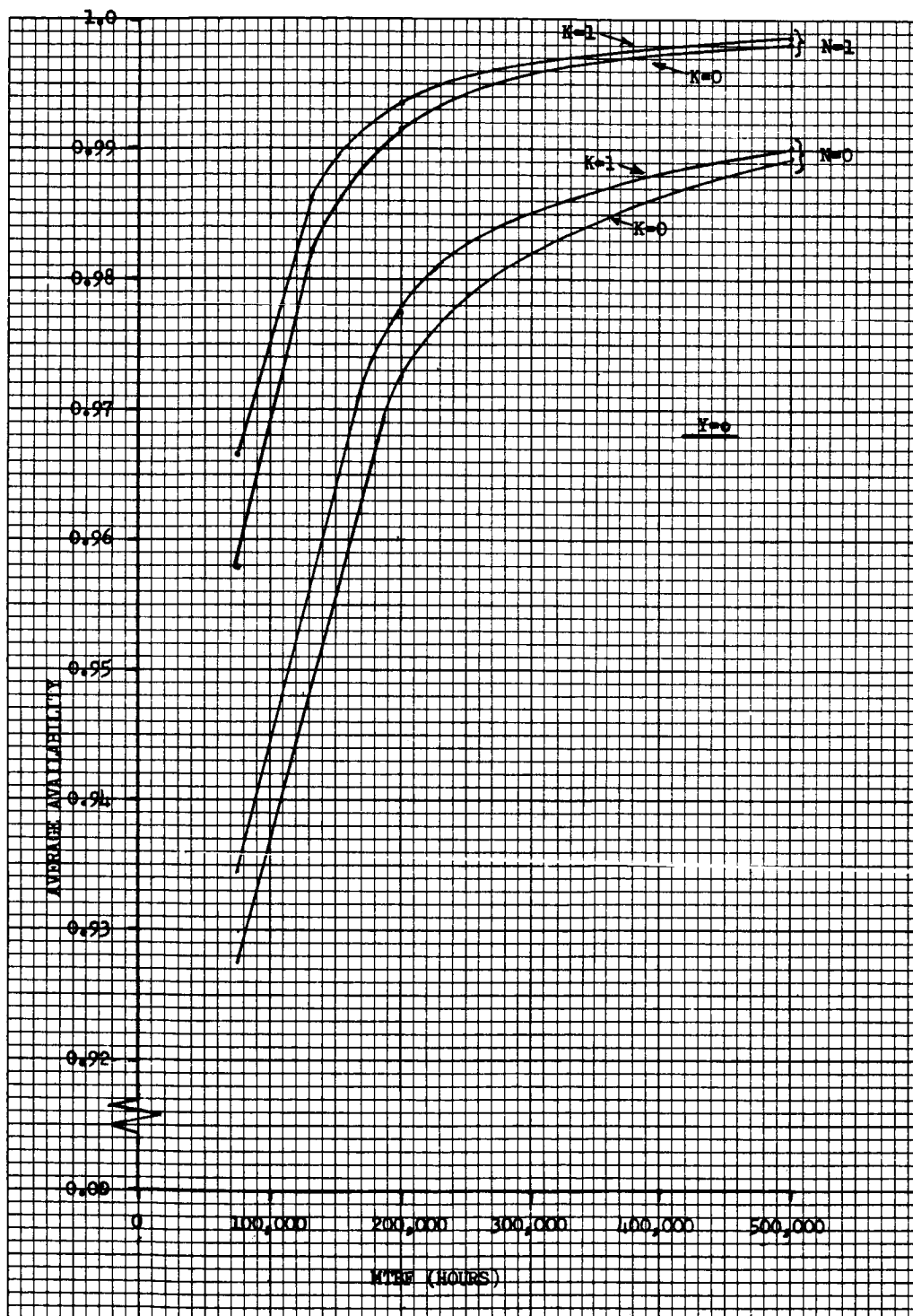


Figure 12-8. Average Availability vs Cell MTBF Varying Number of Inter-Group Busses

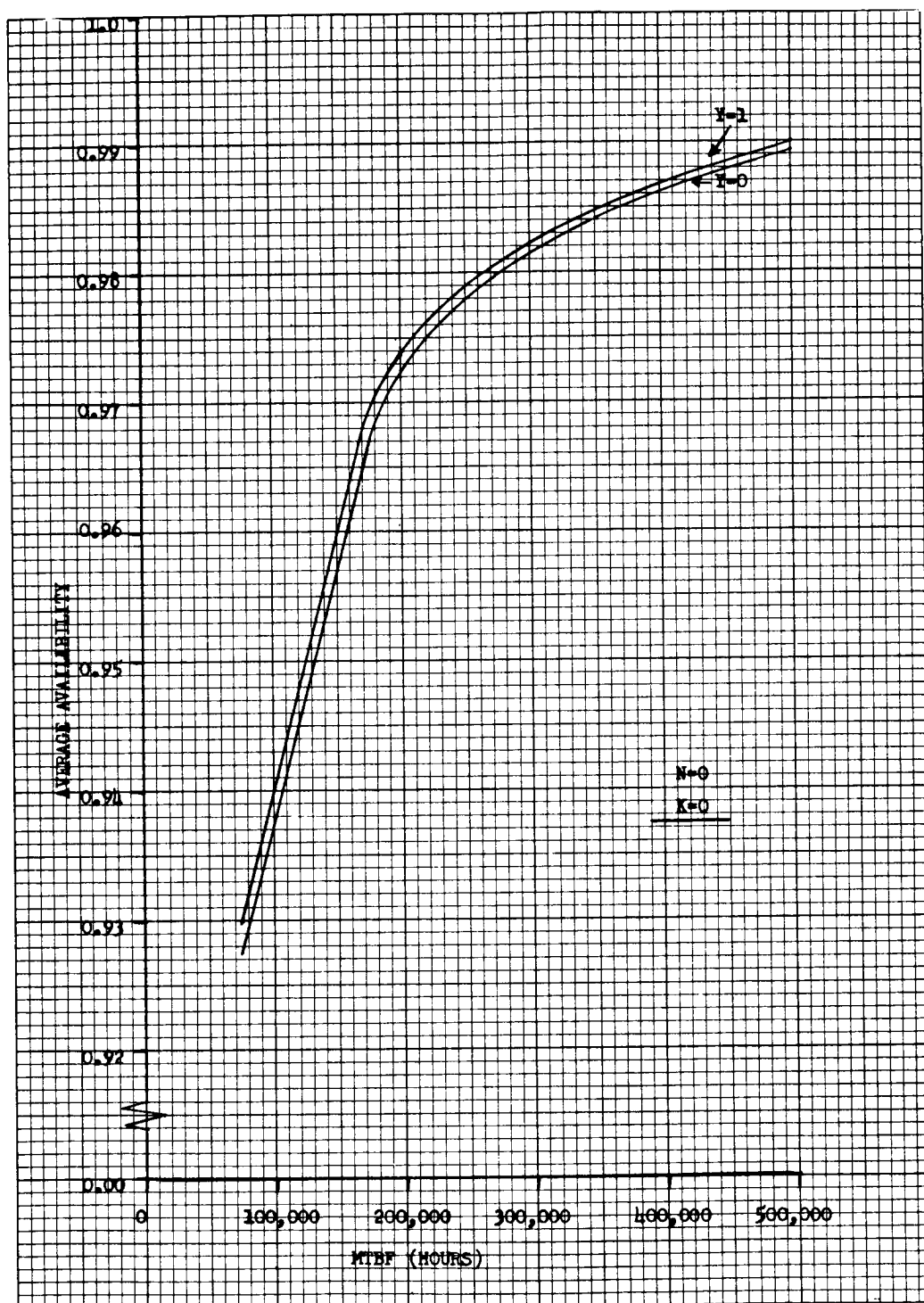


Figure 12-9. Average Availability vs Cell MTBF Varying Number of Inter-Cell Busses

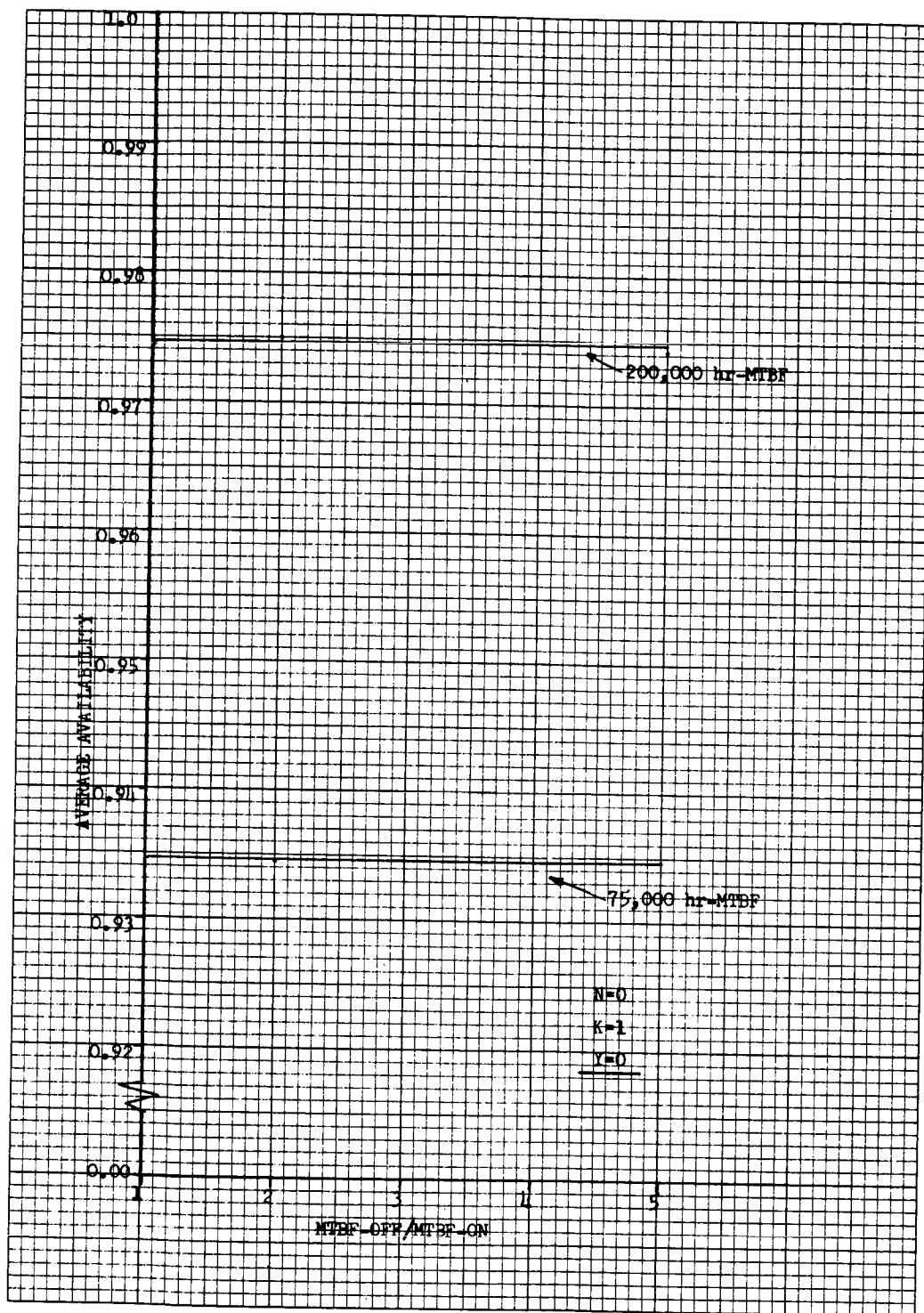


Figure 12-10. Average Availability vs MTFB-OFF/MTBF-ON

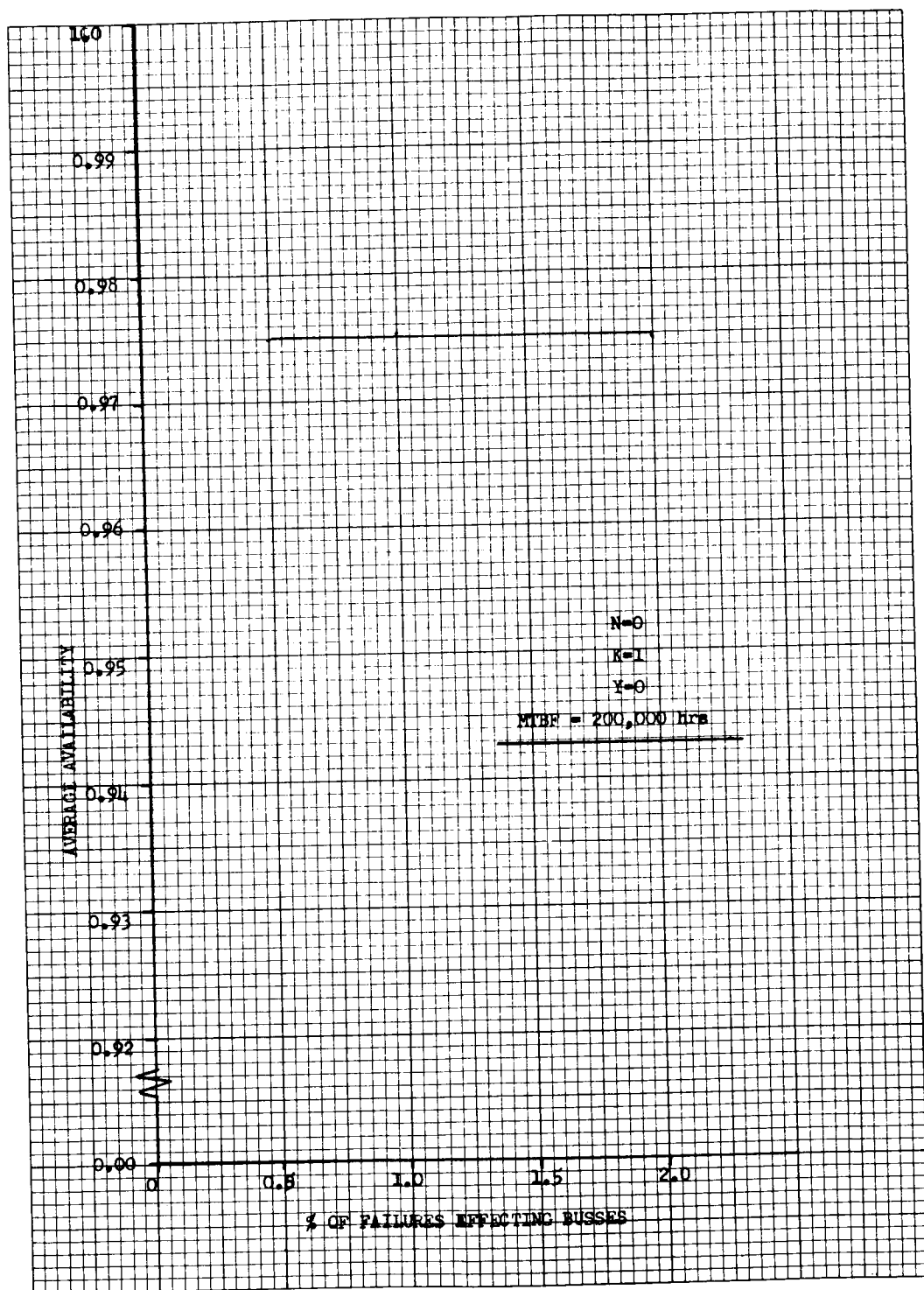


Figure 12-11. Average Availability vs % of Failures Affecting Busses

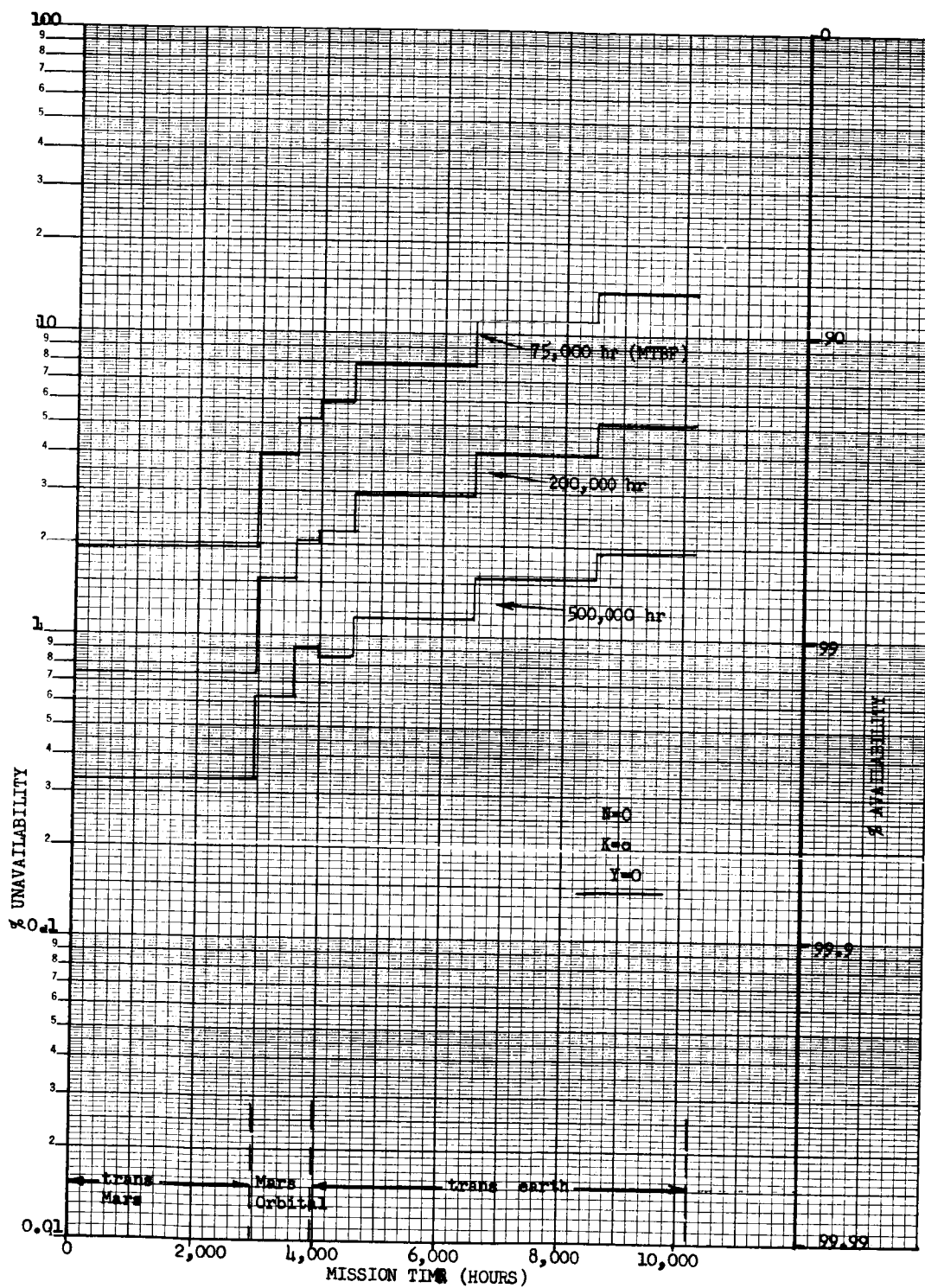


Figure 12-12. Percent Unavailability vs Mission Time for $N=0$, $K=0$, $Y=0$

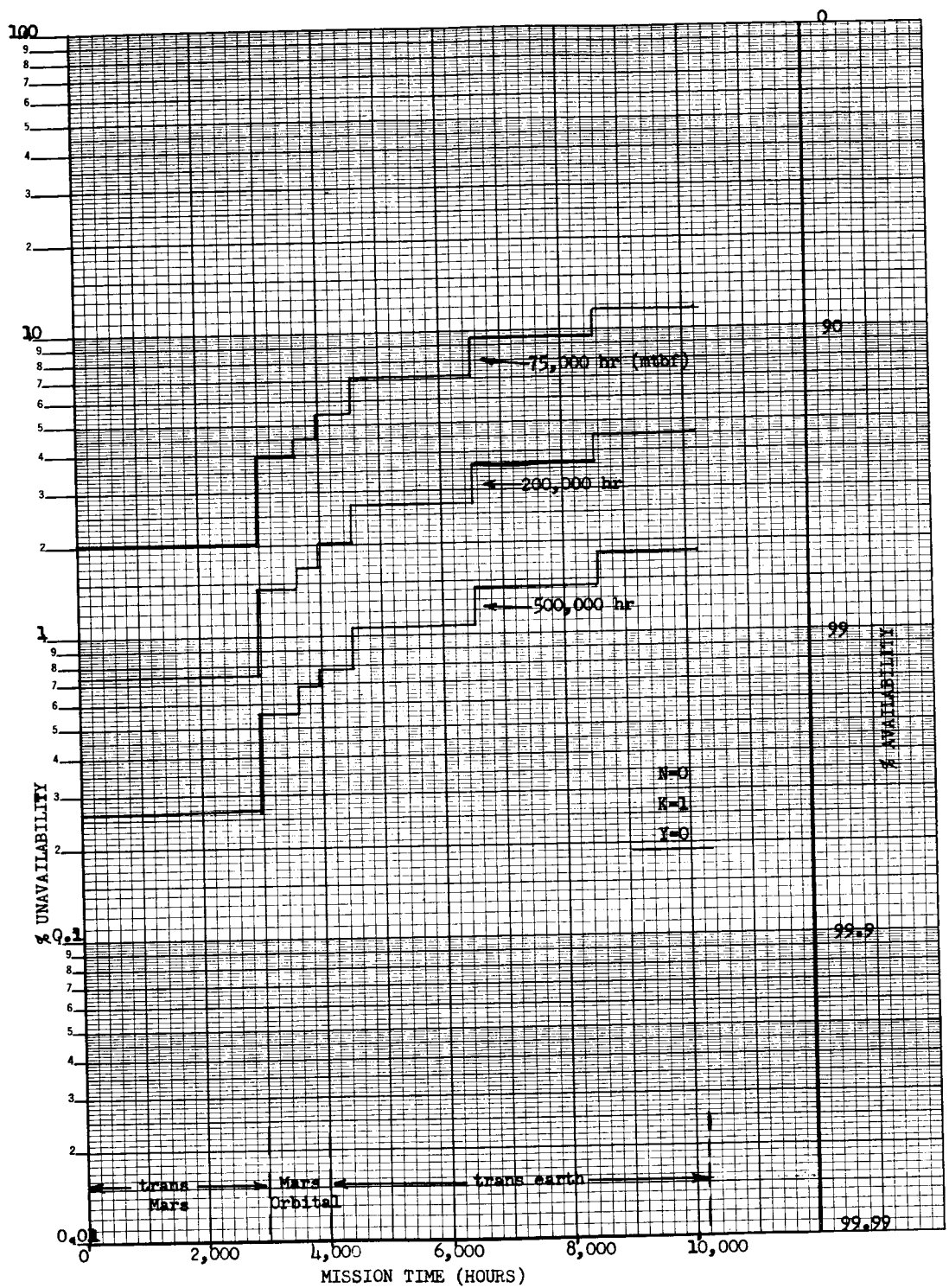


Figure 12-13. Percent Unavailability vs Mission time for $N=0$, $K=1$, $Y=0$

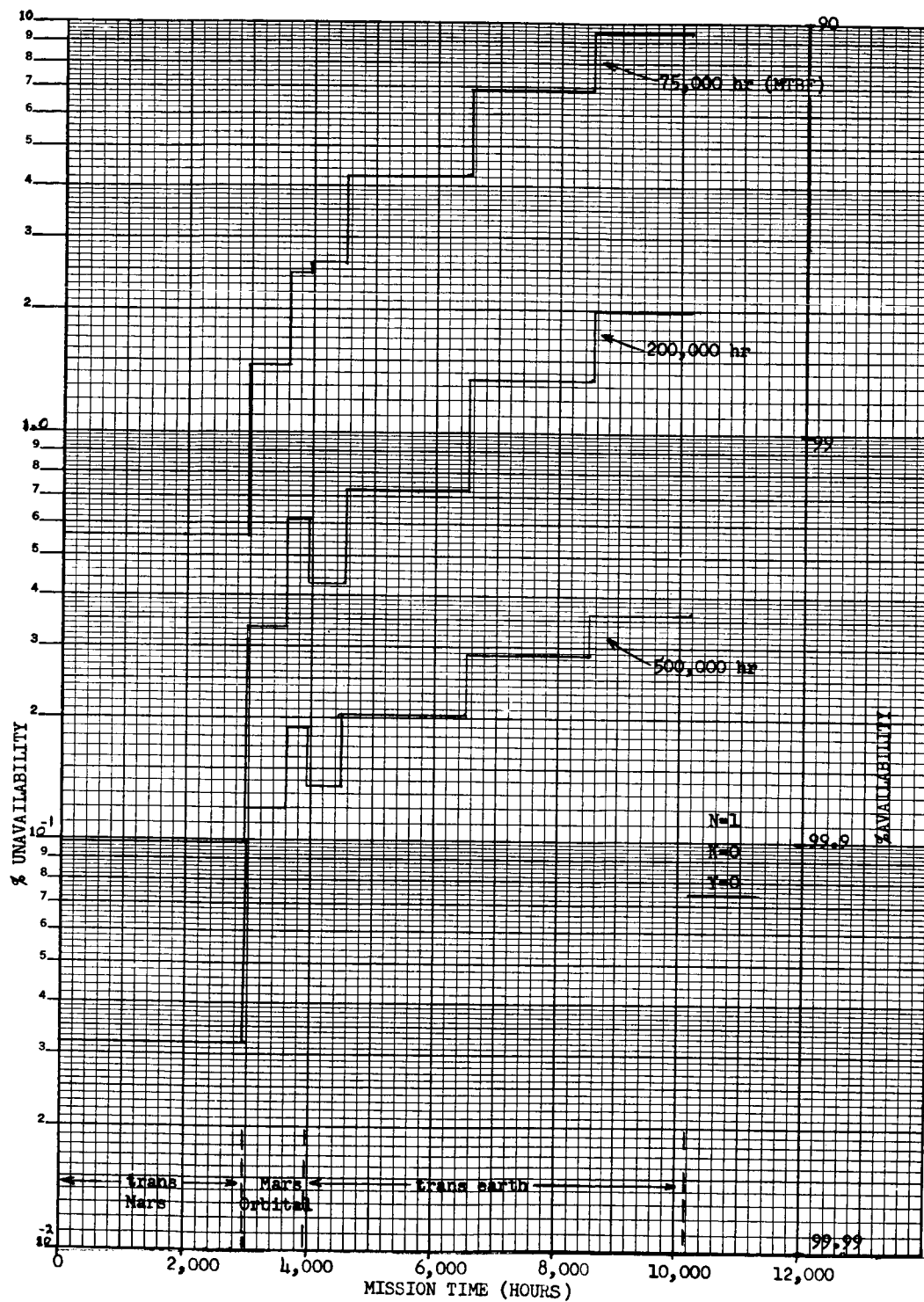


Figure 12-14. Percent Unavailability vs Mission Time for $N=1$, $K=0$, $Y=0$

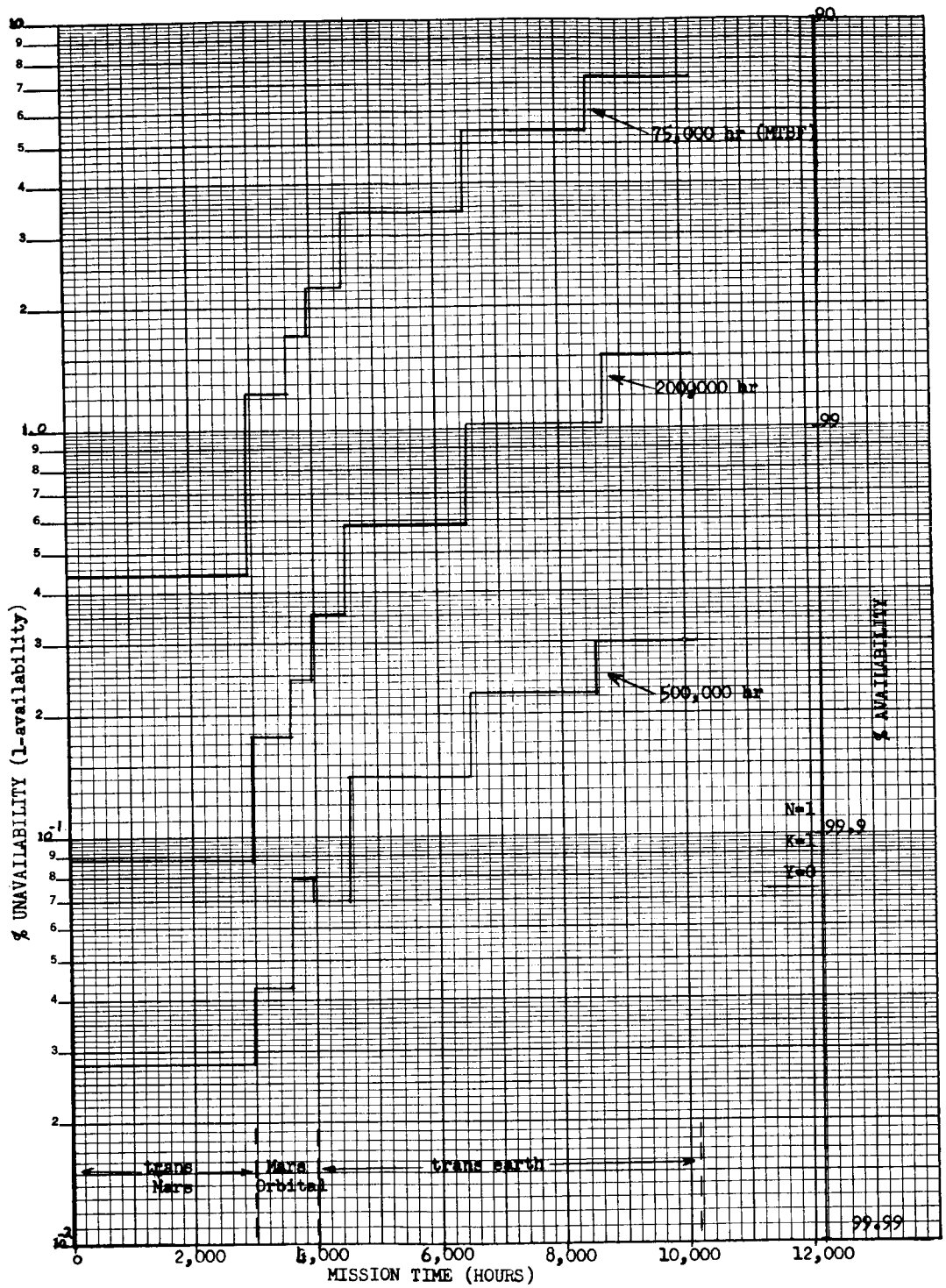


Figure 12-15. Percent Unavailability vs Mission Time for
 $N=1$, $K=1$, $Y=0$

Both percent availability and % unavailability are indicated in the figures. Comparing figure 12-12 ($N=0$, $K=0$, $Y=0$) with figure 12-15 ($n=1$, $K=1$, $Y=0$) one can see the improvements as a function of time by adding spares. Note also that in many of the curves the availability at the end of the Mars orbital phase is lower than at the start of the trans earth phase. This is due to the lower requirements in the trans earth phase.

Probability of Success

It should be recalled that 1000 runs were made for each case and therefore the results can only be extrapolated to the third decimal point. In all the cases run with MTBF's of 200,000 and 500,000 hours no mission failures resulted. Therefore a flat curve at $P_S = 1.0$ results for these values. With MTBF's of 75,000 hours there were a number of mission failures and the values for P_S ranged from 0.967 to 1.000.

Figure 12-16 shows the effects of ON/OFF failure rate ratio on P_S . As may be seen, the only effects were with an ON MTBF of 75,000 hours. The high probability of success of this system is due to the many logical paths available for reconfiguration as discussed in Section 12-2. It should be recalled that availability was not significantly affected by ON/Off failure rate ratios, however, P_S is affected at lower values of MTBF (on) as can be seen in Figure 12-16. The reason behind this appears to be due to the fact that only 1 group is necessary for a successful critical phase and the fact that one group will be off during a good part of the mission means that it will have considerably fewer failures as the ON/Off failure rate ratio is increased.

One other point should be mentioned with regards P_S . The probability of detection was 95 percent for all cases run. One case, case 13, had an undetected failure in a critical phase in one of the 1000 runs. Also one of the runs resulted in a mission failure during that critical phase. It appears that this mission failure was due to the undetected failure since the failure occurred with an intergroup bus available as may be seen by investigating the print out for case 13 in Appendix C. Due to the low number of undetected failures it was felt unnecessary to vary the probability of detection parameter.

Number of Cell, Group Switch, and Bus Failures

The average number of failures of specific components were plotted in Figure 12-17 through 12-19. Figure 12-17 shows the average number of cell failures as a function of MTBF. One may expect from 8.1 to 1.3 failures per mission as MTBF varies from 75,000 hours to 500,000 hours; this is for the entire computer system. Also shown in this figure is the average number of group switch failures per mission with one and 1 spare group switch per group. In all cases one can expect less than 1 of these failing.

Cells 1, 2 and 3 in each group were arbitrarily assigned as having external connections (e.g. used with I/O conditioners). The average number of these types of cells failing over the mission is shown in Figure 12-18. This gives an indication of the number of times a connection may have to be changed due to a failure, providing no spare connections exist. This number varied from 1.43 to 0.22 with cell MTBF.

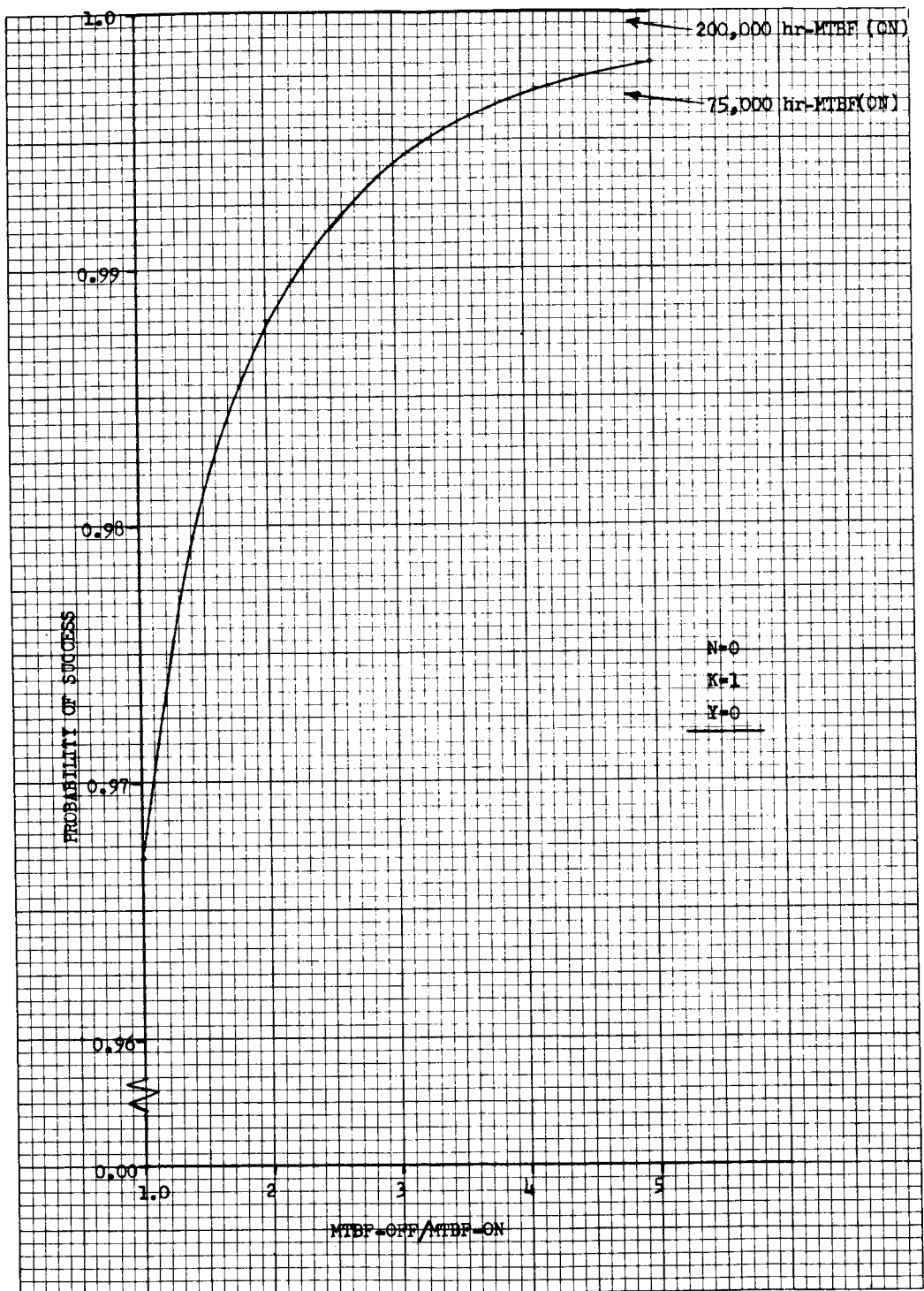


Figure 12-16. Probability of Success vs MTBF-OFF/MTBF-ON

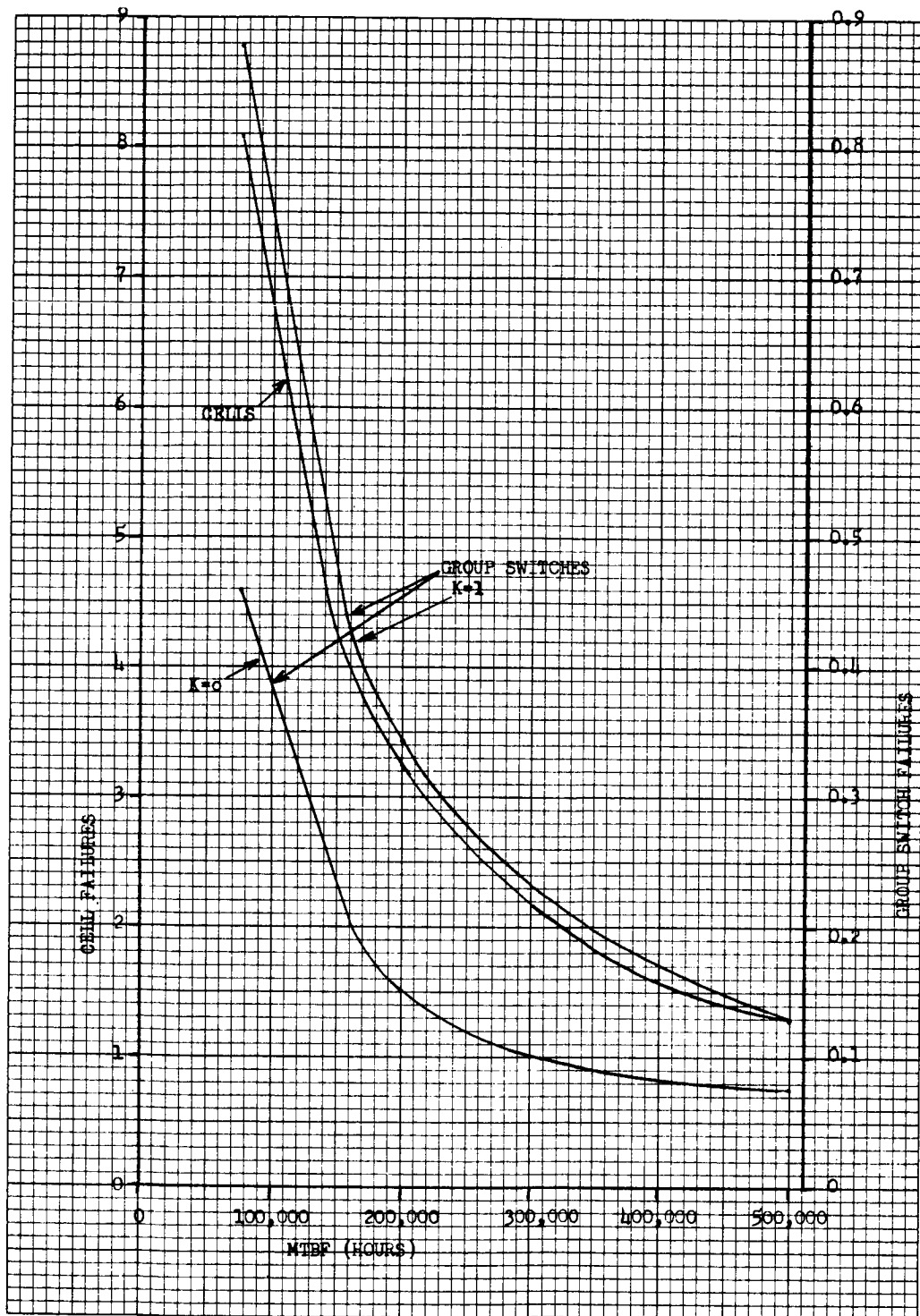


Figure 12-17. Cell and Group Switch Failures vs Cell MTBF

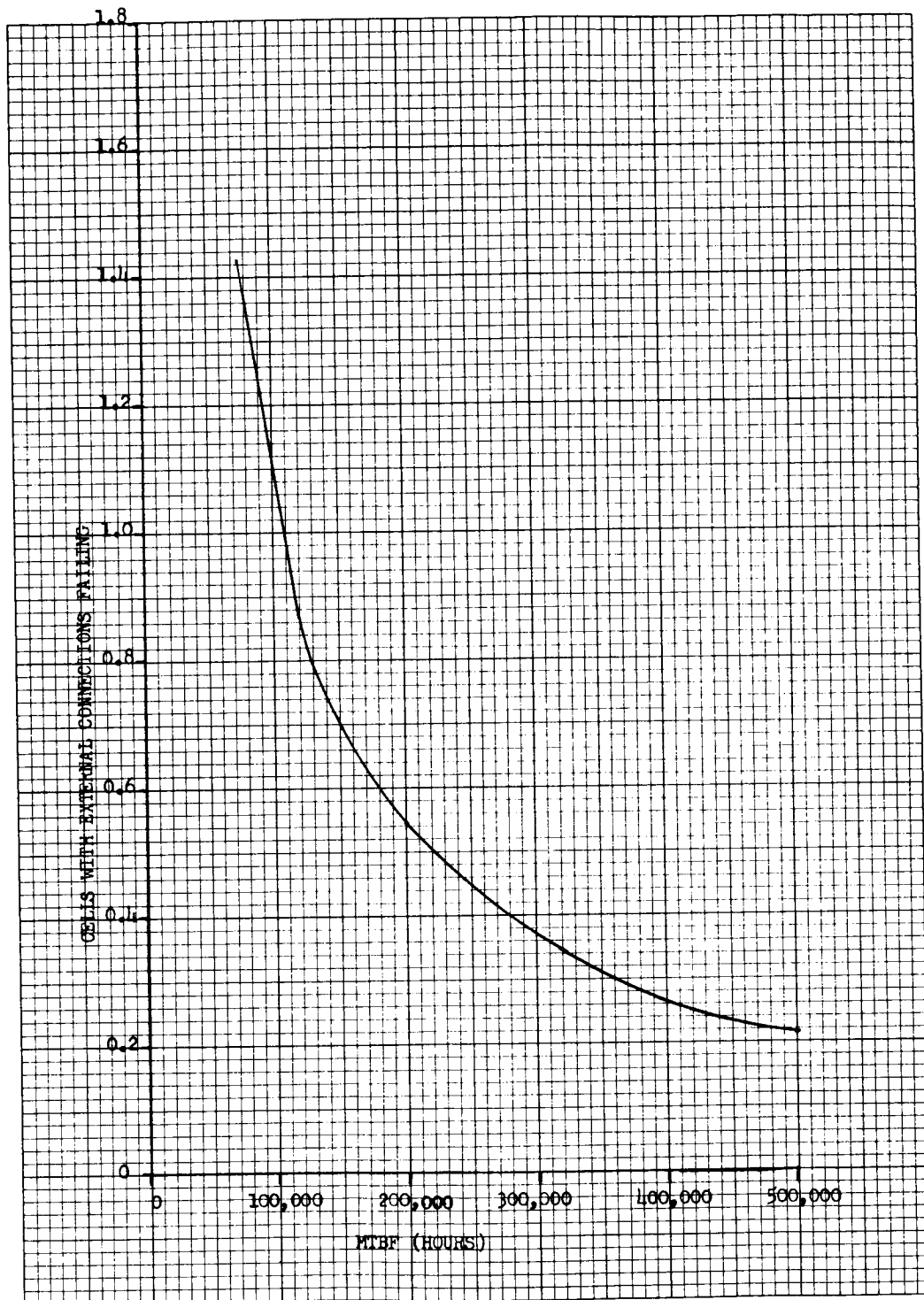


Figure 12-18. Cells with External Connections Failing vs Cell MTBF

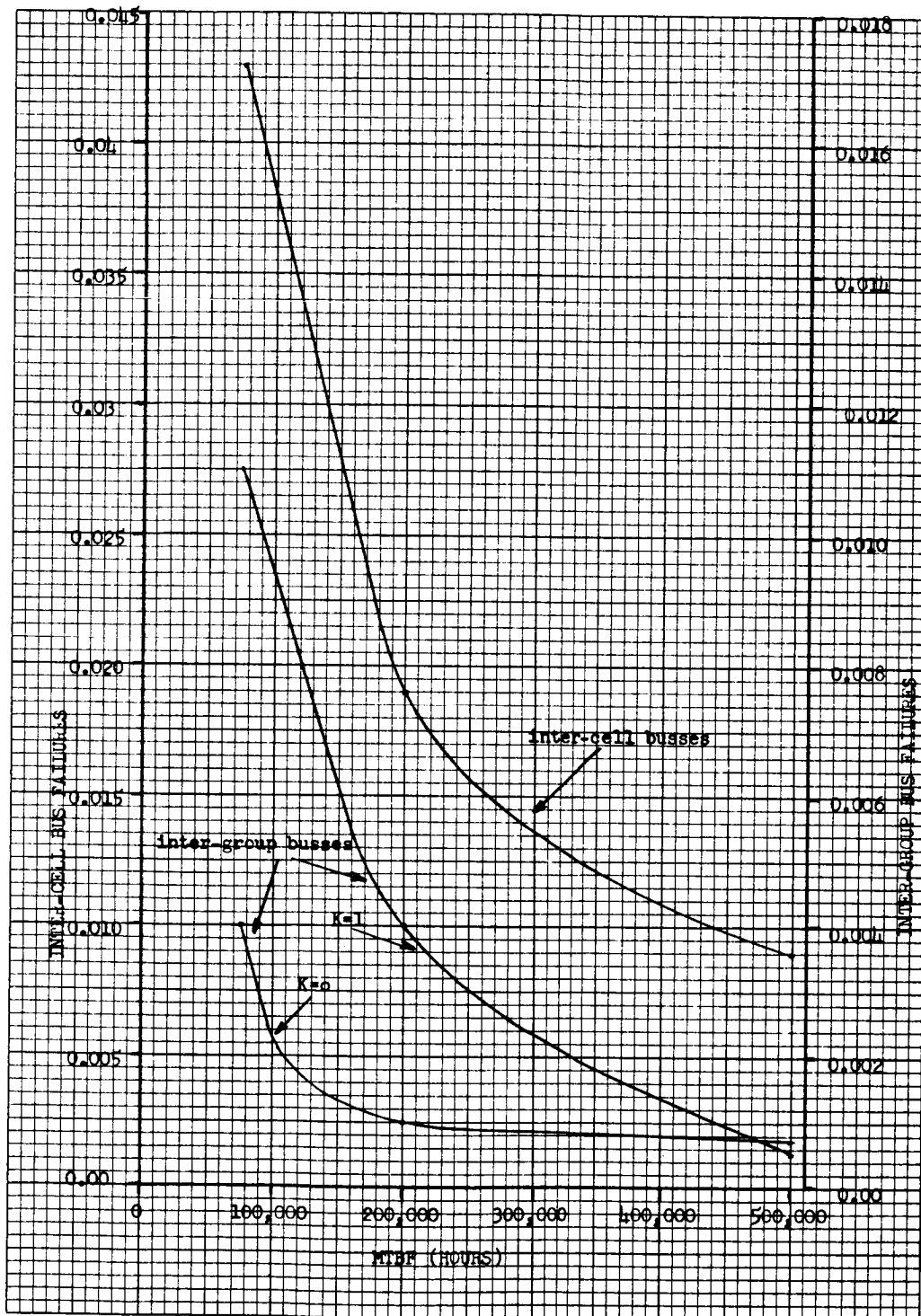


Figure 12-19. Inter-Cell and Inter-Group Bus Failures vs Cell MTBF

Bus failures were determined as a percent of cell and group switch failures. The curves in Figure 12-19 show the average number of bus failures assuming $P_B = 0.5$ percent. In general the numbers here are fairly low.

12.4 SUMMARY AND CONCLUSIONS

More accuracy could be obtained in the results by having made more runs for each case. However, the results appear accurate enough to derive some general conclusions. Availability is influenced very heavily by the number of spare cells and the MTBF. Alloting 1 or 2 spare cells per group results in a fairly high average availability. MTBF's of 200,000 hours or higher provide very good results. A 200,000 hour MTBF is equivalent to a 0.5 percent/1000 hour failure rate which appears a conservative estimate for a single MOS-SOS LSI wafer in the future. Nevertheless, even with conservative estimates, very good results are obtained.

Spare group switches had a slight affect on improving availability. However, spare cell busses had very little effect. The conclusion is that 1 inter-cell bus is all that would be recommended.

Probability of success appears to be very high with this system. However, if the cell MTBF's are quite low; one can obtain a significant improvement in P_g if the MTBF-OFF/MTBF-ON ratio is high. In addition, probability of detection does not appear to be a very significant parameter here.

Finally the number of cell failures do not appear very high. In fact one can expect to have to change a connection less than once during the entire mission due to a failure in the computer.

Considering the above results the recommended organization would be one with 2 spare cells per group and 1 spare group switch (2 inter-group busses) per group. For the manned Mars mission considered, this results in 4 groups of 20 cells per group.

13. SUMMARY AND RECOMMENDATIONS

A summary of the work accomplished during this phase of the 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: (1) the interface between the computer and the sensors, information is lacking as to the nature of the signals expected, (2) 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 function should be decided, (3) the structure of a reliable bulk storage unit for the time period of interest should be investigated and (4) the investigation and design of ultra reliable switches should be carried out. Any future requirements studies should take into account the above points.

It was shown that future LSI technology is difficult to extrapolate. However, it is felt that the technology predicted should be attainable in 10 years. Some important areas of development for LSI techniques were outlined in section 3.

Parallelism within computations is an important area of consideration for highly parallel computer structures such as the distributed processor. In particular, methods for measuring the effectiveness of parallel structures need to be developed and most important of all, the procedures for optimally assigning parallel computations need to be developed.

The organization and its architecture were developed. There are many areas where alternate decisions could have been taken as pointed out in numerous places in chapters 5 and 6, resulting in a slightly different or radically different architecture. Section 6.9 contains a number of such trade-offs.

The design of the cell and group switch may be considered as "preliminary functional." The next logical step in the design would be the detailed logical layout including logic equations and timing diagrams. Such a design was beyond the scope of the study; however, the design carried out showed that no major difficulties should be encountered in a detailed design effort.

Software and executive methods were investigated in considerable depth. However, this area is quite complex for this organization and considerable further study is required in this topic. Some of the more important areas are the reconfiguration executive routines and the assignment executive routines.

The reliability simulations showed that the organization exhibits very impressive probability of success and availability features. A most important result is that even with conservative estimates of a cell (1 wafer) MTBF, 200,000 hours, the reliability results are very good.

Finally, since the technology being considered is considerably away in terms of time, an approach using microwave circuits on the LSI wafer and communicating between cells at RF frequencies was considered. (This is presented in Appendix D.) It was concluded that this approach may very well prove feasible in the time period of interest; however, to make the design perhaps somewhat more near term this approach was not selected. It was shown that with an RF communication system, the organization could be considerably enhanced in terms of reliability and performance.

APPENDIX A. EXAMPLE PROGRAM

A.1 INTRODUCTION

The distributed processor system, being a new concept of computer design, requires some new software to make the best use of the hardware. The example program that is described in this section will show how a typical program is coded and some of the considerations used in coding the problem. Two examples of coding the problem are shown, the second uses global programming, the first does not.

One major part of the software is the compiler that takes the equations and statements from the programmer and converts them into instructions, cell assignments, input/output commands, etc. This compiler should appear to the programmer similar to present advanced languages, such as PL/I or JOVIAL. There will be added statements for the programmer to control the amount of applied parallelism, cell assignments and intercell bus usage. This compiler for the distributed processor system will require development. In the past 10 years the computer industry has progressed from absolute octal and simple assemblers to the powerful languages of today. There is no reason to believe that compiler techniques cannot be developed to compile programs for the distributed processor system. For example, the following statements can be added to the language and used by the programmer to control the compilation.

1. Data Description. To aid the compiler in finding all the places where parallel processing could be used, the programmer could specify how the data is to be arranged. An example is a vector notation $X(3)$. The compiler could place the three components of X in one cell, or put one component in each of three cells.
2. Schedule. The programmer may wish to force data to be processed in parallel, or conversely, he may wish an array to be processed serially by a single cell.

There are several areas where more work needs to be done; some of these are given here.

1. Program interactions. This occurs whenever several programs share, in time, a cell or the use of a subroutine. Several programs may request, simultaneously, the use of a common subroutine. The programmer needs tools to force the compiler to generate programs with the desired interactions.
2. Data allocation. Data may be in several cells or may be in one cell. Data may be placed in one cell and distributed to several cells as needed. The programmer needs ways of controlling how the data is to be arranged in the computer.

3. Data Dependent Problems. An example illustrates this problem best. Assume the problem is to invert an $N \times N$ matrix. When $N = 3$, a 3×3 matrix can easily be inverted within a single cell. To invert a 30×30 matrix ($N = 30$) requires a different program and many cells. Applied parallelism can be used to do the inversion in a reasonable time.

These problem areas are being studied by many people. By the time the manned missions to Mars are ready to fly, these problems should be solved. References 34 thru 36 indicate work that is being done now.

The example chosen for this program is the navigation equations given on pages 50 - 54 of the first quarterly report of Phase I of this study, reference 2. These equations are not so complex as to require many words of explanation, but are of sufficient complexity to illustrate the techniques used to program the distributed processor.

There is some opportunity to use applied parallelism. Many of the vectors have three components that may be processed in parallel. Example 2 will show how applied parallelism can be used. Because of the nature of the navigation problem applied parallelism does not give a dramatic time reduction. A data reduction program is a better example of how applied parallelism reduces time. However only the navigation problem was coded due to time limitations.

The distributed processor has many ways of performing a task. The method selected for this example is not the only way the computer system can be programmed, nor is it expected to represent the best way. The method was selected to best illustrate the software.

The general flow diagram is given in Figure A-1. This diagram and the details of the program will be described in the following sections.

Block 1 is the interface with the controller cell. An assumption made was that the controller cell will contain the current values of the navigation parameters. A Kalman filter would process the various parameters, making the best estimate by using data from the various sensors. The output is the best estimate of the present state, (the state vector). This state vector would be saved in the controller cell and sent to the navigation program. The controller cell will send the time and state vector to the navigation program via the inter-cell bus.

The navigation cell will use the time for a check, as each cell has its own clock. Thus the time parameter is for checking the clocks of the two sets of cells. The navigation cell will keep a copy of the last values it sent to the controller cell. The navigation cell will compare these past values with the ones sent from the controller cell. The values should be in very close agreement. If all is acceptable and the data checks, the navigation calculations will begin with block 2.

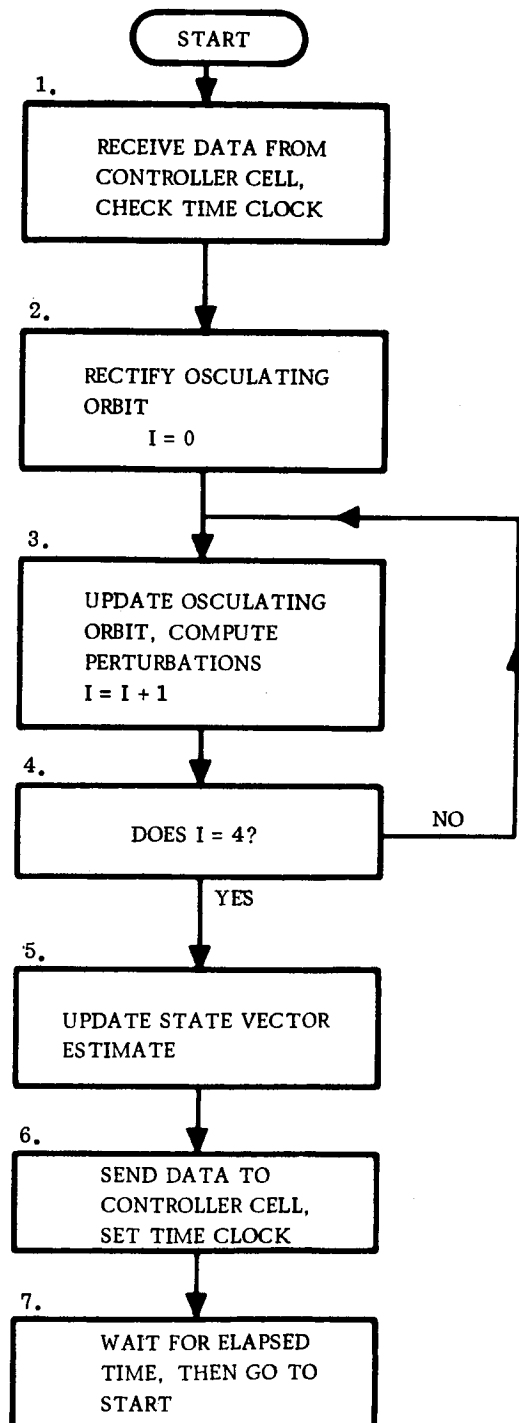


Figure A-1. Navigation Program Flow Diagram

One difference between this computer system and a conventional computer should be noted here. The purpose of block 1 is to use a more accurate state vector value and to start the calculations at the stated time period. However, the cell can function without this block. Each cell contains a clock, so the navigation cell can be interrupted as required to set the proper time period. The past values calculated will be very close to those best estimates from the filter. Thus, if the controller has a reconfiguration or a more important task, the navigation cell can still function and begin its calculations. If the controller should fail to inform the navigation cell that it should begin calculating, perhaps because of a malfunction, the navigation cell's clock will interrupt and the navigation cell will begin anyway. Thus the system is more reliable because it is not depending upon a single processor and a single clock for all its control.

Block 2 rectifies the osculating orbit, if required. As will be seen, there is much processor capability remaining unused, so performing the rectification each time will not cost any additional processor time. However, the inter-cell bus may be needed more. No attempt was made to modify any equations, they were coded as they were given.

Block 3 calculates perturbations and evaluates the derivatives. This is done four times. Block 5 does the numerical integration and updates the state vector.

Block 6 sends back the updated state vector parameters to the controller cell. Block 7 is a wait until it is time to go back and start again.

This navigation program will be described in more detail. Example 1 does not use any global control; it will be described first using the neighbor cell for subroutines and not using the inter-cell bus except when necessary. Then a variation of the example is studied, using the inter-cell bus for passing necessary subroutine data. The second example shows how global programming, using three dependent cells, can be used to solve the program. Initialization is not considered in the examples and would be done earlier in the mission by some other program.

A.2 EXAMPLE 1

The example program and storage estimates are given in Table A-1. In cell 1 is placed the controller cell interface subroutines and two parts of the navigation program that did not fit in cell 2. Cell 2 contains most of the operational navigation program. These two cells make up a self-contained set that is nearly independent of the rest of the group.

The word counts in Table A-1 were obtained by trial programming the equations. All data is assumed to be 32 bits in length, thus most operations were done in double precision (32 bits). It was felt the only exceptions were so few in number as not to affect the results or any conclusions.

Table A-1. Example 1 Storage Estimates

<u>Cell 1</u>	<u>Words</u>
Navigation Program Executive, I/O, Error checking	102
Rectify Osculating Orbit	56
Update State Vector	31
Temporary Storage	40
Subroutine Storage for Square Root, Sin, Cos, Exp.	<u>228</u>
<u>Total Cell 1</u>	457
<u>Cell 2</u>	
Block 3	
Calculate Delta E	73
Update Osculating Orbit	72
Compute Perturbations and evaluate Derivatives	164
Block 4	
Test I, Update Delta	17
Permanent Data Storage and Constants	<u>98</u>
<u>Total Cell 2</u>	424
<u>Total Storage</u>	861

To obtain the inter-cell bus usage, an estimate of the execution time is required. Table A-2 lists an estimate of time required to perform different operations. This set of assumed times makes no claim of absolute accuracy, but is only an estimate of the relative times that can be obtained with a reasonable quantity of hardware.

The inter-cell bus usage can now be estimated by using the times given in Table A-2. The functions as illustrated in Figure A-2 were estimated to require the following times. These times include both hardware and software overhead.

Controller cell send X3 command	20 cycles
Receiving cell set-up time	18 cycles
Send first word of data	6 cycles
Send next 13 words	<u>26 cycles</u>
Total transmit time	70 cycles

Note that only the top half of figure A-2 is applicable here. The lower half applies to the variation of example 1 that will be discussed later (subroutines not in neighbor cell).

Table A-2. Relative Times Used in Programming Examples

Time to transfer one byte on the inter-cell bus	1 cycle
Time for one memory cycle	2 cycles
Load or store an accumulator	6 cycles
Add or subtract, single precision	6 cycles
Add or subtract, double precision	10 cycles
Multiply, double precision	42 cycles
Perform SIN/COS, square root, double precision	600 cycles
Perform SIN and COS with one call, double precision	1000 cycles
Perform exponential, double precision	1000 cycles

The fourteen data words are the six state vector parameters and the time. The inter-cell bus is not needed again until the computed parameters are to be returned to the controller cell. This is assumed to require the same time as sending the data to the navigation cell. This assumes that the controller cell will request the navigation cell to return its computed parameters. The navigation program requires 140 cycles of inter-cell bus time.

The program execution time is estimated below.

Block 1	
Start	16 cycles
Input start parameters, check	214 cycles
Block 2	2200 cycles
Block 3 and 4	51600 cycles
Block 5	1000 cycles
Block 6	90 cycles
Block 7	<u>remaining time</u>
Total	55120 cycles

There are, depending upon the hardware capabilities, between 10^6 to 10^7 cycles per second of real time. Thus between 5.5 percent to 0.55 percent of the cycles available to the cell in a second are being used. The cell is in block 7 (idle) during the remaining time as far as this navigation program is concerned. (Of course during the idle time the cell could be used for global operations from another program etc.)

The inter-cell bus usage for this task is only 0.014 percent to 0.0014 percent. The navigation problem, as programmed here, has used more storage to decrease dependence upon the inter-cell bus.

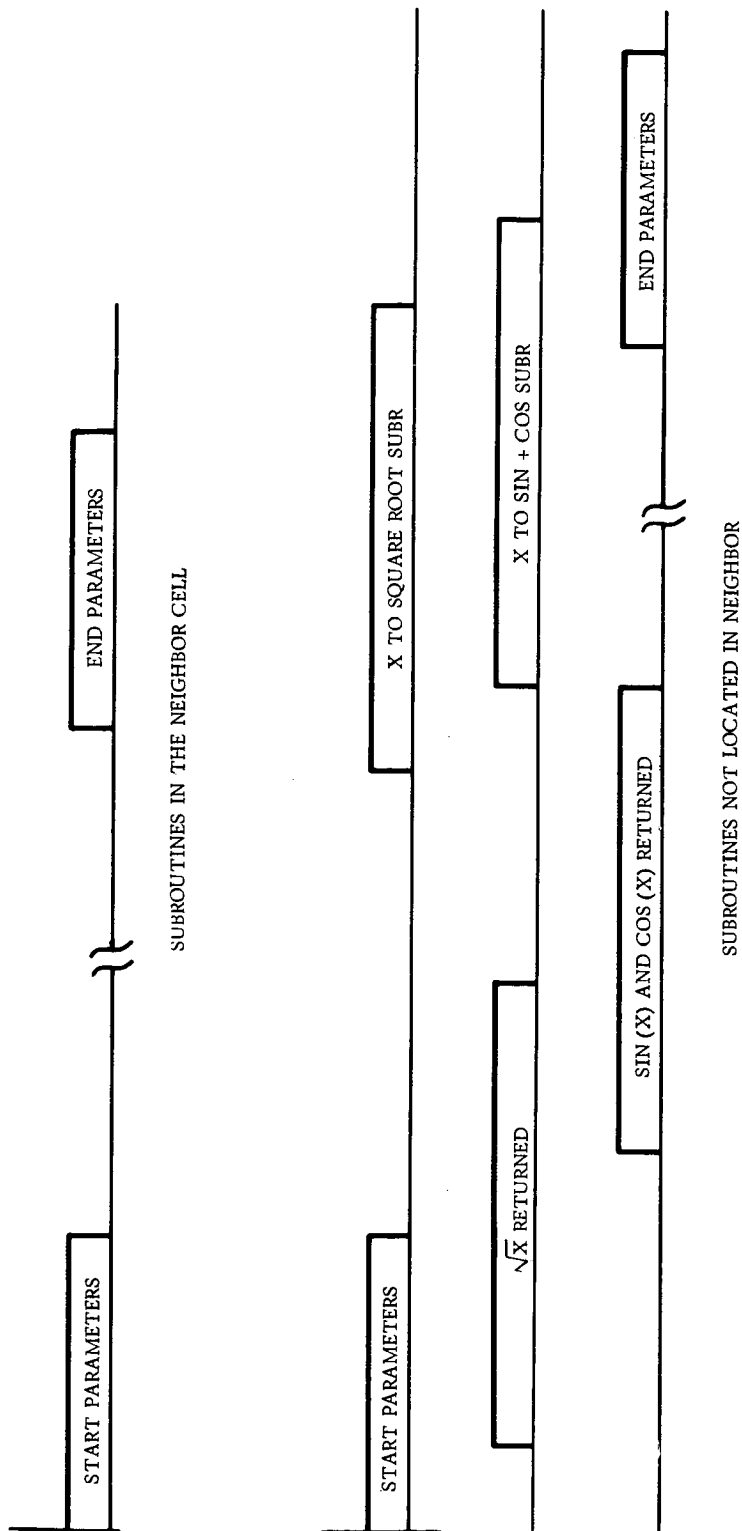


Figure A-2. Example 1, Inter-Cell Bus Requirements

Taking another look at the problem, some storage could be saved by using mathematical subroutines located in another cell. Again, there are many choices that can be made. The square root, because it is used so much, could be left in cell 1. The other subroutines could be moved into a storage area in cell 1 whenever they were needed. Or the variable could be transmitted on the cell bus to a subroutine cell, and the subroutine cell will return the desired function. The latter case was examined in more detail.

For one cell to communicate to another requires a sequence of commands to and from the controller cell and the communicating cells. This sequence is described in section 6 of this report.

The estimated time required to send a double precision word to a subroutine was calculated as follows. C1 is the navigation cell set, CC the controller cell, CS the subroutine cell as shown in Table A-3.

Table A-3. Communication Sequence Timing

Sender	Receiver	Command/Data	Cycles required
CC	C1	X2 - Report communication requests	2
C1	CC	I/O is desired	8
CC	CS	X3 - Prepare to input data	30
CC	C1	X7 - Output data	10
C1	CS	Data Words transmission time	4
Total cycles required			54

To return the data from the subroutine cell to the navigation cell will require the same amount of time. Thus 108 cycles are required for each subroutine call. This is neglecting the time the sending cell waits for the controller cell to ask if it has any I/O. This time to poll the cells is variable, and is a function of the rest of the tasks contained in the group. This time will be ignored in this discussion, not because it is unimportant but because there is no way to analyze the amount of time used by other tasks in this group. When they are all known this delay can be estimated.

There are 42 subroutine calls in the navigation program, which require the times on the bus given in Table A-3. Table A-4 shows the time for execution of the subroutines using this new approach.

Table A-4. Subroutine Timing Requirements

Subroutine	Execution Time Including Bus Time	Times Required By the Program	Total Time Including Bus Time
Square Root	708	18	12744 cycles
SIN	708	4	2832
COS	708	4	2832
SIN and COS	1112 *	12	13344
EXP	1108	4	4432
Total		42	36184
*Both sine and cosine are returned, requiring 4 more cycles than if only one function is returned.			

This time is added to the previous execution time and the previous inter-cell bus usage. The previous execution time estimate was 55120. Adding the additional inter-cell bus time requirements gives a total of 59660 cycles required to calculate the new state vector. It is seen that the total execution time required in the cell is not increased greatly.

The inter-cell bus usage will increase as shown in part of Figure A-2. The addition of 4540 cycles to 140 cycles gives a total of 4680 cycles. If there are 10^6 cycles available in a second of real time, 0.47 percent of the inter-cell bus is used by the navigation program. The saving of subroutine storage in the navigation cell set requires increased inter-cell bus time. The saving in cell 1 reduces the storage requirements therein from 457 words to 239 words, this gain is of course at the expense of increased bus usage.

These two variations of example 1 show how the distributed processor system gives great flexibility in the use of the system resources. System availability in event of a malfunction is one important reason for having a flexible system. The programmer can make trade-offs to give the resultant system the best chance of meeting system objectives.

A.3 EXAMPLE 2

The navigation program was programmed a second time, but this time global programming was used to show the effects of global programming on the software. Example 1 used conventional coding. Example 2 will show how global programming is used. This section will go into much more detail on the coding. Because most of the navigation vectors contain three components, three cells are used to process the data. Usually X is calculated in cell CX, Y in cell CY, and Z in cell CZ. Cell CY is the center and main processing cell. Neighbor lines connect CY to CX and CZ, as shown in Figure A-3.

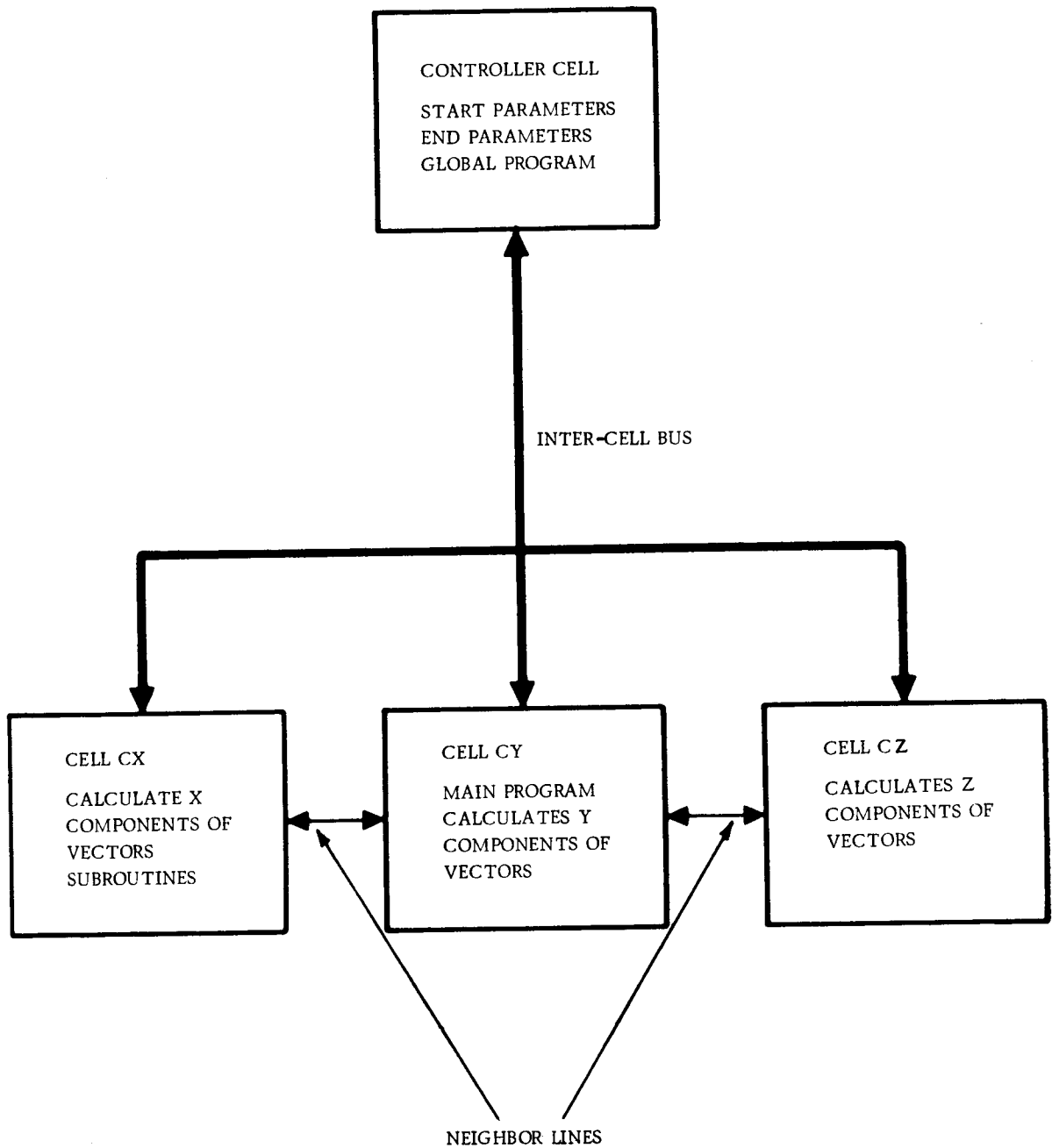


Figure A-3. Example 2, Program Allocation to Cells

The equations were analyzed for places where applied parallelism could be used effectively. The resulting dependent program sequences, as they appear on the inter-cell bus, are shown in Figure A-4. Tables A-5 and A-6 are the program that generates the global sequences. The make-up of the sequences is given in Tables A-8 and A-10 thru 7-12.

The example shown uses an exponential subroutine in order to calculate the atmospheric drag. As in example 1, the subroutines may be in various places throughout the system and may be used in various ways. The method proposed here was to have the exponential, squareroot, sin/cos in the CX cell. The exponential could also be sent from the controller cell, or the subroutine cell. Or the data could be sent to the subroutine cell and the subroutine cell can return the function. Figure A-4 assumes the exponential is located in CX and the data is sent over the neighbor line to CX.

It should be noted scaling and rounding have been omitted in all these examples. The equations in 1. 2. 4. 2. 1, (reference 2) calculating the eccentric anomaly, use only one variable and there are few opportunities for doing natural or applied parallelism. One possibility is to put sin in CX and cos in CZ, and calculate the sin and cos of the estimated eccentric anomaly simultaneously. In this example applied parallelism was not used.

A look at Tables A-5 and A-6 show many details, most of which are of interest only to professional programmers. A few items will be mentioned here because they are of interest in studying the distributed computer system.

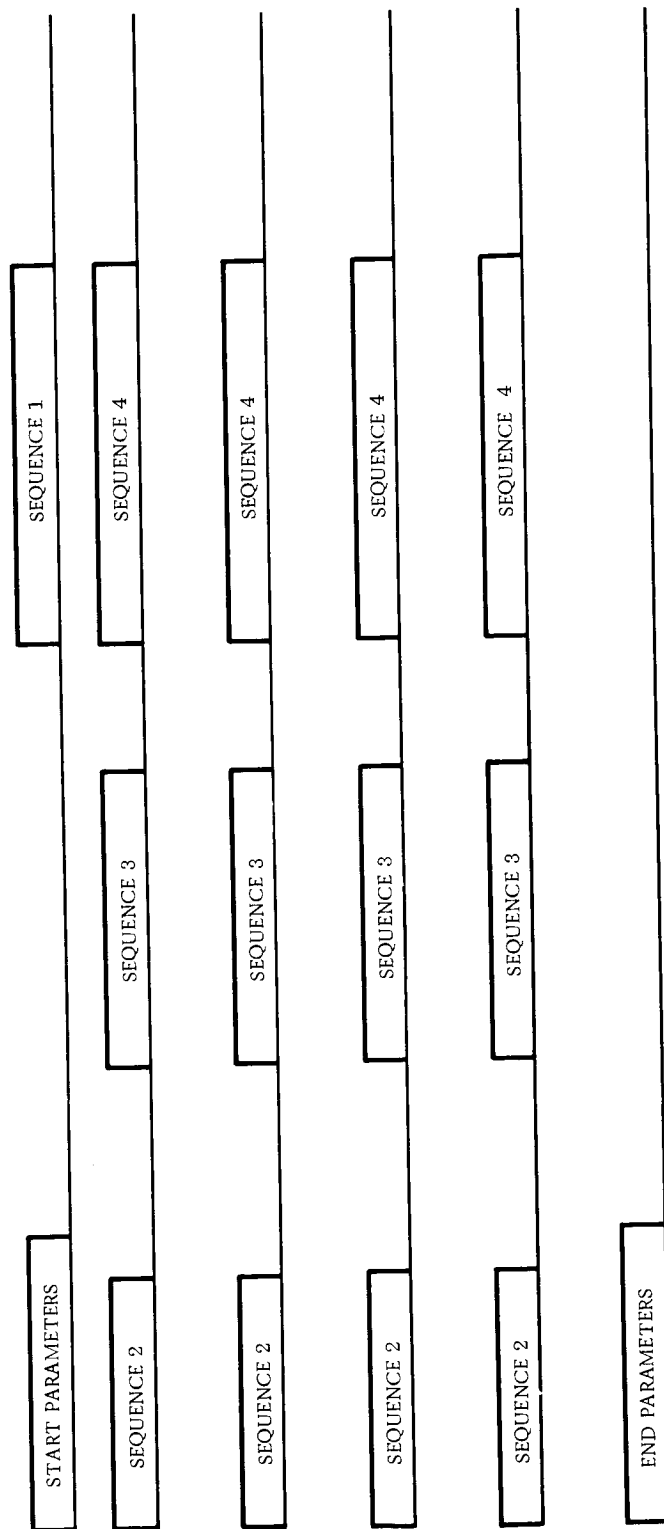
Table A-5 shows a program to solve the equations given in section 1. 2. 4. 1. of reference 2. These equations rectify the osculating orbit. Usually the orbit is rectified whenever the deviation vector becomes large. This example assumes that the rectification takes place, thus this is a worst case example.

The first two lines shown in Table A-5 transmit the components of RV and $\dot{R}V$ to the proper cells.

The first global sequence is given next. The BEGIN line tells the compiler that this next block is to be made into a global program. Cells CX, CY and CZ are all to receive the global instructions. The compiler will compile a program to be placed in the controller cell. This program will be called whenever this sequence is required. The next 4 statements will be made into a global program. This program, shown in Table A-7, will be executed by the controller cell. The output of this program is shown in Table A-8.

Looking first at Table A-7, the first three instructions are controller cell instructions and are executed by the controller cell. Instruction 1 will cause cell CX to be addressed, set to level L, and set to the dependent wait state (DW). Level L is any one of the 8 levels. Any level may be used, just so all three cells are set to the same level.

Instructions 2 and 3 will perform the same operation on cells CY and CZ. Thus three X7 control words will be sent out over the inter-cell bus.



NOTE:
TIMES ARE NOT TO SCALE

Figure A-4 Example 2, Inter-Cell Bus Usage

Table A-5. Program for 1.2.4.1 (Ref 2)

Line Number	Statement
1	TRANSMIT RVXT0, RDVXT0 FROM CY TO CX
2	TRANSMIT RVZT0, RDVZT0 FROM CY TO CZ
3	BEGIN GLOBAL SEQUENCE 1 TO CX, CY, CZ
4	RV2 = RVT0 * RVT0
5	RDV2 = RVDTO * RVDTO
6	DELTA = 0
7	DELTAD = 0
8	END GLOBAL SEQUENCE 1
9	TRANSMIT RV2X, RDV2X FROM CX TO CY
10	TRANSMIT RV2Z, RDV2Z FROM CZ TO CY
11	V02 = RDV2X + RDV2Y + RDV2Z
12	R0 = SQRT(RV2X + RV2Y + RV2Z)
13	A = UG * R0 / (2 * UG - R0 * V02)
14	ECOSE0 = 1 - (R0/A)
15	ESINE0 = V02/SQRT(UG * A)
16	N = SQRT(UG)/A ** 1.5

Table A-6. Program for 1.2.4.2 Through 1.2.4.5 (Ref 2)

Line Number	Statement
1	FP = 1 - (A/R0) * (1 - COS(DE))
2	GP = DT - (DE - SIN(DE))/N
3	TRANSMIT FP, GP FROM CY TO CX, CZ

Table A-6. (Cont)

Line Number	Statement
4	BEGIN GLOBAL SEQUENCE 2 TO CX, CY, XZ
5	$ROSCT = FP * RT0 + GP * RDT0$
6	$RM2 = ROSCT * ROSCT$
7	END GLOBAL SEQUENCE 2
8	TRANSMIT RM2 FROM CX TO CY
9	TRANSMIT RM2 FROM CZ TO CY
10	$RM = \text{SQRT} (RM2X + RM2Y + RM2Z)$
11	$FV = \text{SQRT} (UG * A) * \text{SIN} (DE) / (R0 * RM)$
12	$GV = 1 - (A/RM) * (1 - \text{COS} (DE))$
13	TRANSMIT FV, GV FROM CY TO CX, CZ
14	BEGIN GLOBAL SEQUENCE 3 TO CX, CY, CZ
15	$RDOSCT = FV * ROSCT0 + GV * RDOSCT0$
16	$RVT = ROSCT + DELTAT$
17	$RDVT = RDOSCT + DELTADT$
18	$Q = (DELTAT * (DELTAT - 2 * RVT)) / (RVT * RVT)$
19	END GLOBAL SEQUENCE 3
20	TRANSMIT Q FROM CX TO CY
21	TRANSMIT Q FROM CZ TO CY
22	$Q = QX + QY + QZ$
23	$FQ = (Q * (3 + Q + Q * Q)) / (1 + (1 + Q) ** 1.5)$
24	TRANSMIT RVT * RVT, RDVT * RDVT FROM CX TO CY
25	TRANSMIT RVT * RVT, RDVT * RDVT FROM CZ TO CY

Table A-6. (Cont)

Line Number	Statement
26	$RV2 = RVTX * RVTX + RVTY * RVTY + RVTZ * RVTZ$
27	$HE = \text{SQRT}(RV2)$
28	$LA = \text{SQRT}(RDVTX * RDVTX + RDVTY * RDVTY + RDVTZ * RDVTZ)$
29	$LB = \text{GAMMA} * LA * \text{EXP}(-BETA * HE)$
30	$LC = RM2 * \text{ROSCT}$
31	$L1 = J * RE2 / RV2$
32	$L2 = 5 * Z2 / RV2$
33	TRANSMIT L1, L2, LB, LC FROM CY TO CX, CZ
34	BEGIN GLOBAL SEQUENCE 4 TO CX, CY, CZ
35	DEFINE L3 = 1 IN CX, CY
36	DEFINE L3 = 3 IN CZ
37	$A2 = L1 * RV * (L3 - L2)$
38	$A1 = LB * \text{RDT}$
39	$AD = A1 + A2$
40	$KV(I) = \text{DELTADT}$
41	$KA(I) = AD - UG * (FQ * R + \text{DELTAT}) / LC$
42	IF I ≤ 3 DO
43	$\text{DTNHV} = \text{DTNV} + (H/2) * KV(I)$
44	$\text{DTNHA} = \text{DTNA} + (H/2) * KA(I)$
45	END DO

Table A-6. (Cont)

Line Number	Statement
46	IF I = 4 DO
47	$CV1 = (H/6) * (KV(1) + 2 * KV(2) + 2 * KV(3) + KV(4))$
48	$CA1 = (H/6) * (KA(1) + 2 * KA(2) + 2 * KA(3) + KA(4))$
49	RTN = ROSCT + CV1
50	RDTN = RDOSCT + CA1
51	END DO
52	END

Table A-7. Controller Cell Program to Generate Sequence One

Line Number	Instruction Operation	Operands
1	CC	CX, LEVEL L, DW
2	CC	CY, LEVEL L, DW
3	CC	CZ, LEVEL L, DW
4	CC	Transmit Mode, Level L
5	LOAD DP	U1, RV
6	MPY DP	U1, U1
7	STORE DP	U1, RV2
8	LOAD DP	U1, RDV
9	MPY DP	U1, U1
10	STORE DP	U1, RDV2
11	SUB DP	U1, U1

Table A-7. (Cont)

Line Number	Instruction Operation	Operands
12	STORE DP	U1, DELTAT0
13	STORE DP	U1, DELTADT0
14	CC	Execute Mode, Level L, DLC

Table A-8. First Sequence Received by CX, CY, CZ

Line Number	Operation	Operands
1	PREFIX	LEVEL L
2	LOAD DP	U1, RV
3	MPY DP	U1, U1
4	STORE DP	U1, RV2
5	LOAD DP	U1, RDV
6	MPY DP	U1, U1
7	STORE DP	U1, RDV2
8	SUB DP	U1, U1
9	STORE DP	U1, DELTAT0
10	STORE DP	U1, DELTADT0
11	GC	Go to Dependent Local Control State

Instruction 4 sets the controller to the transmit mode and a X0 control byte at Level L is sent out. Three dependent cells are at level L and the three cells will respond and switch to the global state. The controller cell will go into the transmit mode, not executing instructions until line 14 is fetched from memory. Lines 5 thru 13 are sent to the dependent cells. When the line 14 instruction is fetched, because it is a CC instruction, the instruction is executed only by the controller cell. The controller cell is set to the execute mode. An X0 control byte is sent over the global bus at level L, setting all three cells to the Dependent Local Control (DLC) state. The controller cell will now do some other task; the three cells will each execute their local instructions to solve the navigation problem.

Table A-8 shows the instructions received by each cell. Line 1 is the X0 control byte sent by the instruction on line 4 of Table A-7. The cells are all set to the global state and are ready to receive instructions from the inter-cell bus. Lines 2-4 form the square of RV. The DP means Double Precision, or a 32 bit data word is to be used. U1 and U2 are considered one 32 bit accumulator. The LOAD DP loads U1 and U2 with the double precision data word RV. The next instruction will multiply U1 and U2 by U1 and U2, giving a 64 bit product in accumulators U1, U2, U3, U4. In this example, only 32 bits are saved and stored in RV2. Normally, shift instructions would be used to scale the data. The scaling has been omitted in this example.

Instructions 5-7 square RV. The next instruction subtracts U1 - U2 from itself, setting it to zero. This double precision zero is stored by instructions 9 and 10. An alternate method would be to send the zeros from the controller cell as a D32 data word. In this case, Table A-7, line 11 - 13, would be changed as follows:

<u>Line</u>	<u>Instruction</u>	<u>Operands</u>
11	GC	D32
12	STORE DP	U1, DELTAT 0
13	Data	D0
14	GC	D32
15	STORE DP	U1, DELTADT 0
16	Data	D0

This sequence is clearly inefficient when a zero can be generated so easily in the cell by using a register - to - register instruction.

The last instruction received is one that sets the three cells to the dependent local control state. As seen in Table A-5, the cells next operation is to transmit the resultant squares to CY.

Cell CY will compute R0 and V02 from the components received from the neighbor cells. The machine coding for this neighbor-to-neighbor transfer is given in Table A-9.

Table A-9. Neighbor to Neighbor Communication

Line Number	Cell CX	Cell CY	Cell CZ
1	FLAG U1, CY	LOAD DP U3, U1	FLAG U1, CY
2	FLAG U2, CY	LOAD U1, CX	FLAG U2, CY
3	LOAD DP U1, RVT0X	LOAD U2, CX	LOAD DP U1, RVT0Z
4	FLAG U1, CY	ADD DP U3, U1	FLAG U1, CY
5	FLAG U2, CY	LOAD U1, CZ	FLAG U2, CY
6		LOAD U2, CZ	
7		ADD DP U3, U1	
8		STORE DP U3, V02	

Table A-10. Second Sequence Received by CX, CY, CZ

Line Number	Operation	Operands
1	PREFIX	LEVEL L
2	LOAD DP	U1, FP
3	MPY DP	U1, R
4	STORE DP	U1, ROSC
5	LOAD DP	U1, RD
6	MPY DP	U1, U1
7	ADD DP	U1, ROSC
8	STORE DP	U1, ROSC
9	MPY DP	U1, U1
10	GC	Go to Dependent Local Control State

Table A-11. Third Sequence Received by CX, CY, CZ

Line Number	Operation	Operands
1	PREFIX	LEVEL L
2	LOAD DP	U1, RDOSCT0
3	MPY DP	U1, GV
4	STORE DP	U1, RDOSCT
5	LOAD DP	U1, ROSCT0
6	MPY DP	U1, FV
7	ADD DP	U1, RDOSCT
8	STORE DP	U1, RDOSCT
9	ADD DP	U1, DELTAD
10	STORE DP	U1, RDVT
11	MPY DP	U1, U1
12	STORE DP	U1, TEMP1
13	LOAD DP	U1, ROSCT
14	ADD DP	U1, DELTAT
15	STORE DP	U1, RVT
16	MPY DP	U1, U1
17	STORE DP	U1, TEMP4
18	LOAD DP	U1, RVT
19	CMPL	U1
20	CMPL	U2
21	SHIFT	
22	ADD DP	U1, DELTAT
23	MPY DP	U1, DELTAT

Table A-11. (Cont)

Line Number	Operation	Operands
24	DIVIDE DP	U1, TEMP4
25	STORE DP	U1, Q
26	GC	Go to Dependent Local Control State

Table A-12. Fourth Sequence Received by CX, CY, CZ

Line Number	Operation	Operands
1	PREFIX	LEVEL L
2	LOAD DP	U1, L3
3	SUB DP	U1, L2
4	MPY DP	U1, L1
5	MPY DP	U1, RV
6	STORE DP	U1, A2
7	LOAD DP	U1, LB
8	MPY DP	U1, RDT
9	STORE DP	U1, A1
10	ADD DP	U1, A2
11	STORE DP	U1, AD
12	LOAD DP	U1, DELTADT
13	STORE DP	U1, KV(I)
14	LOAD DP	U3, FQ
15	MPY DP	U3, R
16	ADD DP	U3, DELTAT

Table A-12. (Cont)

Line Number	Operation	Operands
17	MPY DP	U3, UG
18	LOAD DP	U1, AD
19	SUB DP	U1, U3
20	DIVIDE DP	U1, LC
21	STORE DP	U1, KA(I)
22	LOAD DP	U1, H
23	SHIFT	
24	MPY DP	U1, KV(I)
25	ADD DP	U1, DTNV
26	STORE DP	U1, DTNHV
27	LOAD DP	U1, H
28	SHIFT	
29	MPY DP	U1, KA(I)
30	ADD DP	U1, DTNA
31	STORE DP	U1, DTNHA
32	LOAD DP	U1, H
33	DIVIDE DP	U1, SIX
34	STORE DP	U1, TEMP
35	LOAD DP	U1, KV + 1
36	SHIFT	
37	LOAD DP	U3, KV + 2
38	SHIFT	

Table A-12. (Cont)

Line Number	Operation	Operands
39	ADD DP	U1, U3
40	ADD DP	U1, KV + 3
41	ADD DP	U1, KV
42	MPY DP	U1, TEMP
43	STORE DP	U1, CV1
44	ADD DP	U1, ROSCT
45	STORE DP	U1, RTN
46	LOAD DP	U1, KA + 1
47	SHIFT	
48	LOAD DP	U3, KA + 2
49	SHIFT	
50	ADD DP	U1, U3
51	ADD DP	U1, KA + 3
52	ADD DP	U1, KA
53	MPY DP	U1, TEMP
54	STORE DP	U1, CA1
55	ADD DP	U1, RDOSCT
56	STORE DP	U1, RDTN
57	CC	Go to Dependent Local Control State

Notes for Table A-12

Instructions 22 through 31 are sent out only when I is less than 4.
Instructions 32 through 57 are sent out only when I is equal to 4.

Cell CX set the neighbor flag for its accumulator U1 to indicate the contents of this accumulator are to be sent over the neighbor line to cell CY. Cell CX is expecting cell CY to request a word of data. When this request comes, accumulator U1 has been set for cell CY. Since the product which is to be transferred was just computed, there is no need to load the accumulator. At the same time, cell CY is moving its product to the other pair of accumulators, U3 and U4. Now (line 2) U1 is loaded with the first accumulator that has the flag for cell CY. Thus CX will transfer its accumulator U1 to CY, which will place the contents in its U1. When CY executes the line 2 instruction, the accumulator U2 contents will be transferred from CX to CY.

While CY is transferring U2, CX will attempt to execute its instruction on line 3. However, U2 continues to contain a flag, therefore this instruction execution will be delayed until CY executes its line 4 instruction.

The remaining instructions operate in a similar manner. Thus the CZ components are moved to CY and added. Because the data is double precision, 32 bits, the program is longer than if the data were single 16 bit precision. In the latter case, the addition can be done directly, the instructions to load the CY accumulators are not required.

Returning to Table A-5, the line 12 instruction takes the square root of the sum of the squares. In this example, the square root subroutine is assumed to be located in one of the neighbor cells. The data is transferred to the neighbor and the square root is computed and returned. The remaining statements are similar.

The equations in section 1. 2. 4. 2. 1 of reference 2 are not described here. The programming is all contained in cell CY and is conventional.

Table A-6 contains the program for the remaining sections. The data description, data equivalence and similar statements have been deleted. The statements are not written in any particular language, since the purpose of these examples is to show how a problem may be implemented and not to develop a compiler or assembler language. The programs to obtain the starting parameters and return the ending parameters to the controller cell have also been omitted. The other global programs have been written in detail and are given in Table A-10 thru A-12. These are similar to the sequence one program, Table A-8. One comment that should be made is $K(I)$, as given in the equations, is a 6 element vector. To make the program easier to follow, the vector was made into two three component vectors, $KV(I)$ and $KA(I)$. The resultant program is equivalent.

Storage estimates were made for the example 2 program. These estimates are given in Table A-13. Cell CX contains instructions, data storage and the subroutines. The instructions are all required to move data into and out of the cell over the neighbor lines and to transfer from the local control to wait state. There are 10 instructions included to set the cell up to receive the first sequence of global instructions. Some words are required to save intermediate values as they are calculated and to save data as it comes from cell CY. This data storage is temporary and may be used by other tasks when the navigation program is completed. Only cell CY is assumed to save permanent data that may be required between executions of the navigation program.

Table A-13. Example 2 Storage Estimates

<u>Cell CX</u>	
Instructions	60 words
Data Storage, temporary	58
Subroutines	<u>228</u>
Total CX	346 words
<u>Cell CZ</u>	
Instructions	60 words
Data Storage, temporary	<u>58</u>
Total CZ	118 words
<u>Cell CY</u>	
Navigation Program Executive, I/O, and error checking	102 words
Instructions	207
Program to calculate delta E	73
Program for block 4	17
Data Storage	<u>98</u>
Total CY	497 words
<u>Controller Cell</u>	
Instruction Storage	<u>91 words</u>
Total Controller Cell	91 words
Total Storage	1052 words

Cell CZ is like CX, only the subroutine package is not required. Cell CY contains the remaining programs, the executive and I/O interface with the controller cell. The instructions include the same 60 instructions as cells CX and CZ. The remaining are the instructions required to solve the equations in Table A-5 and A-6 that are not part of the global program sequences. The program to calculate delta E (1. 2. 4. 2. 1) is here, as well as block 4. The data storage are the remaining words.

The controller cell will require about 90 words to hold the global program.

Figure A-5 shows the general time requirements for example 2. Each line represents 4000 cycles. The blocks on the lines show the approximate inter-cell bus time required. A little over 52000 cycles of time are required to complete the navigation program, or 5.2 percent of the available time, assuming 10^6 cycles per second of real time. The inter-cell bus requirements go to 0.55 percent of the available time, again assuming 10^6 cycles per second.

Comparing example 1 with example 2, a cost of storage and increased inter-cell bus time requirements was paid to reduce the execution time. For a critical program such as navigation, the example 1 approach appears to have definite advantages. There is no speed problem and two cells should be better than three. The two examples show how the distributed processor computer system gives the programmer the flexibility of changing the programs to best meet the system objectives. For a problem that requires much data handling, the converse will be true; the mechanization with the global programming will require much less time and less storage.

START
PARAMETERS

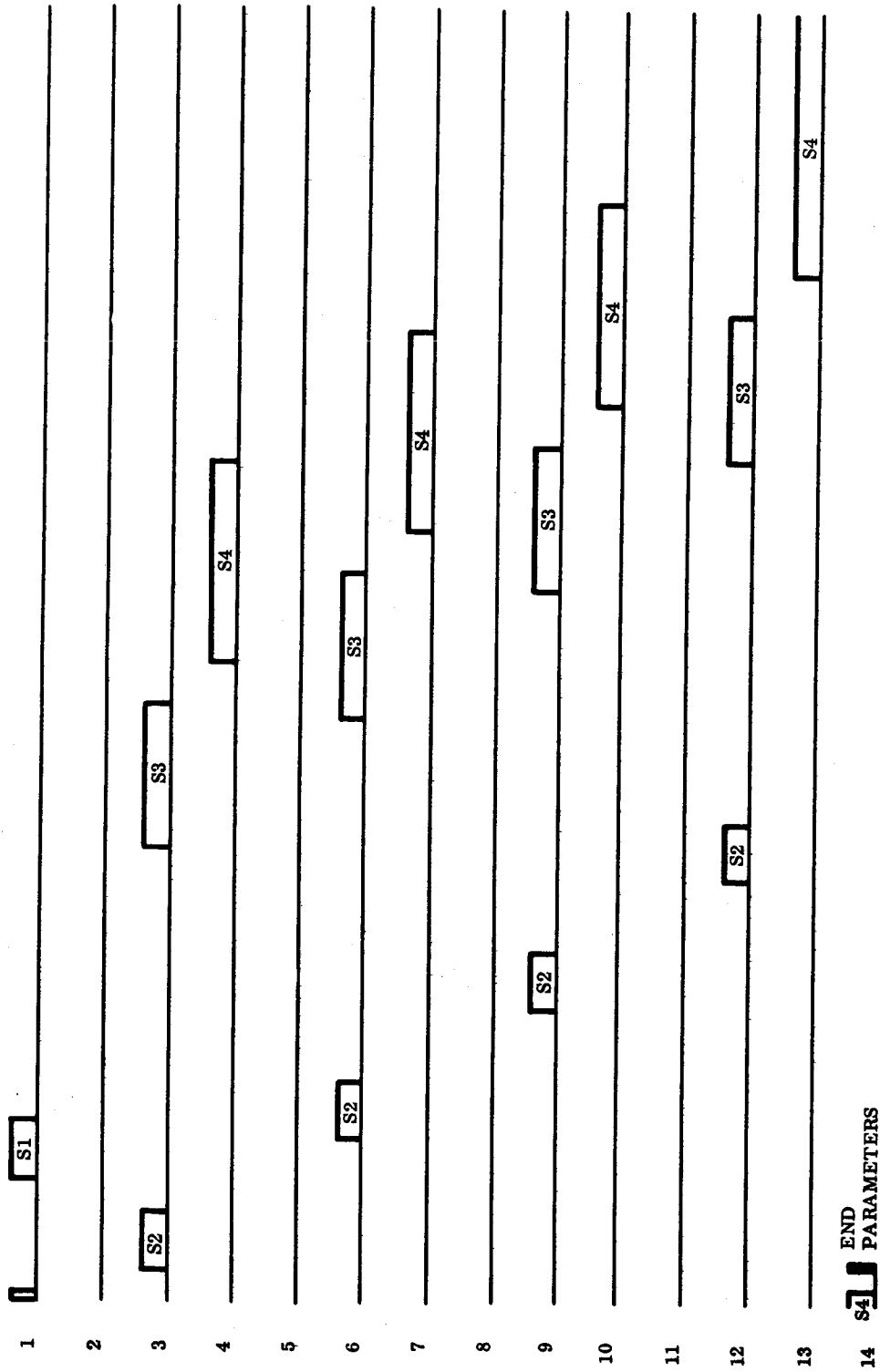


Figure A-5. Example 2, Timing

APPENDIX B. ERROR ANALYSIS OF MONTE CARLO PROGRAM

The Monte Carlo program used in the reliability simulation is described in Section 12. This appendix will present an analysis of the accuracy and confidence one can expect from Monte Carlo techniques.

The number of runs, N , necessary to achieve a given accuracy and confidence in the results can be found from the following equation:

$$N \geq \frac{p(1-p) K^2(\alpha)}{\epsilon^2}$$

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 37). In order to simplify this task without sacrificing accuracy the normal approximation to the binomial distribution is applied. This states that:

$$p \left[\frac{|N(f_n - p)|}{\sqrt{Np(1-p)}} \leq h \right] = 2 \Phi(h) - 1$$

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 $\frac{|N(f_n - p)|}{\sqrt{Np(1-p)}}$

$$\Phi(h) = \int_0^h \frac{e^{-1/2 y^2}}{\sqrt{2\pi}} dy$$

This can be rewritten as

$$p \left[\frac{|f_n - p|}{\gamma} \leq h \right] = 2 \Phi(h) - 1$$

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, $(\sqrt{\frac{p(1-p)}{N}})$, 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:

$$P \left[|f_n - p| \leq \epsilon \right] \approx 2 \Phi \left(\epsilon \sqrt{\frac{N}{p(1-p)}} \right) - 1$$

if $K(\alpha)$ 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

$$P \left[|f_n - p| \leq \epsilon \right] \geq \alpha \quad \text{if} \quad \epsilon \sqrt{\frac{N}{p(1-p)}} \geq K(\alpha)$$

which will be satisfied if

$$N \geq \frac{K^2(\alpha)}{\epsilon^2} p(1-p)$$

$K(\alpha)$ can be found from a table of the normal curve of error. If the value of $1/2 \alpha$ is found in the area column the corresponding value in the t column is $K(\alpha)$. 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.

$$\epsilon \leq K(\alpha) \gamma \quad \text{i.e., } \epsilon \leq K(\alpha) \sqrt{\frac{p(1-p)}{N}}$$

Figure B-1 contains the error as a function of p for various number of runs at a 95% and 75% confidence level.

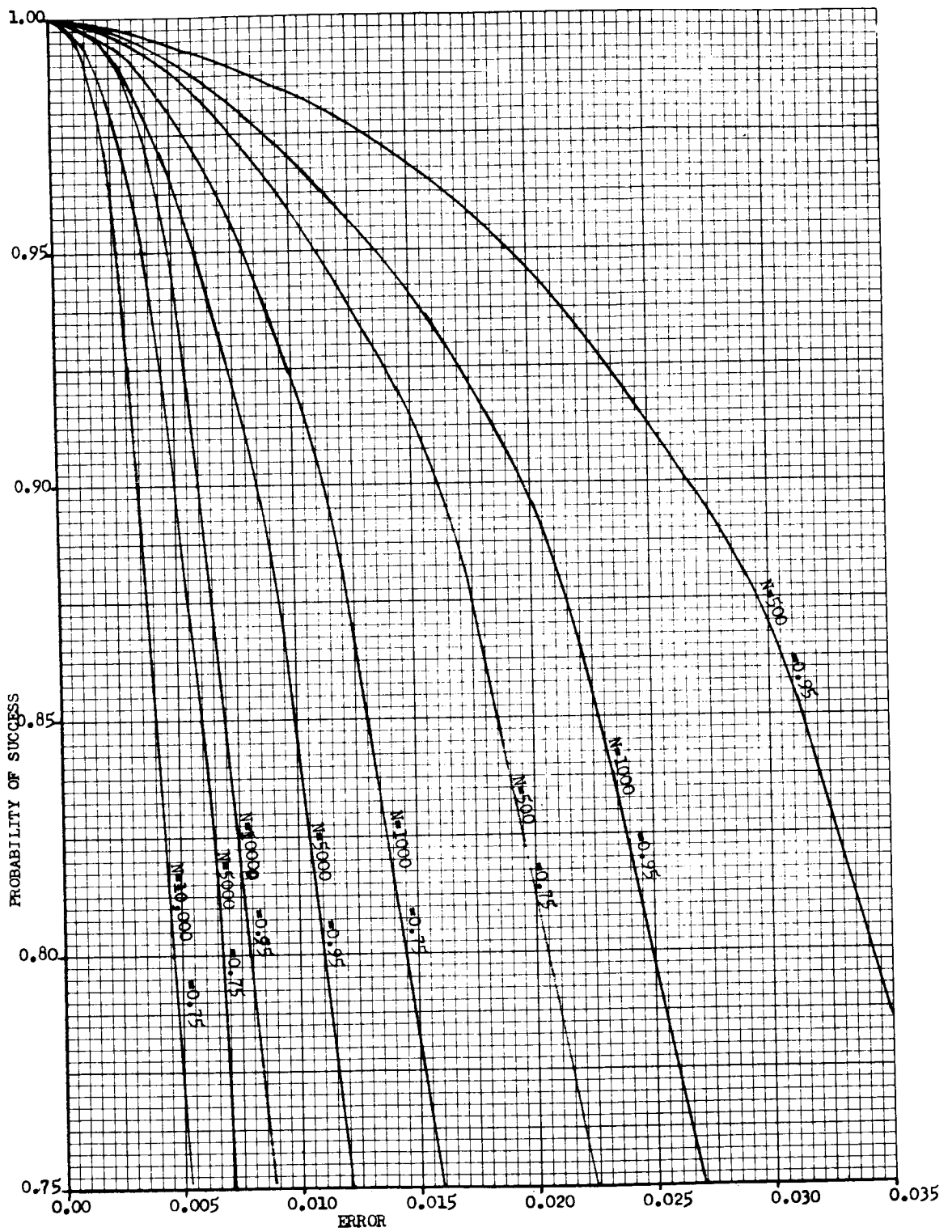


Figure B-1. Error vs P_s in Monte Carlo Results

APPENDIX C. MONTE CARLO SIMULATION RESULTS

This appendix contains the actual print outs from the Monte Carlo simulation discussed in Section 12 of the report. A description of the program along with an explanation of the print out may be found in that section.

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING (M=0, N=1, K=1, Y=0, PD=.95, MTBF= 75,000)

CASE 2 --- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99555		0.417	2.595	0.265	0.015	0.001	
CR-2	2963.8		1.00000	0.0	0.001	0.0	0.0	0.0	0.0
MO-3	3600.0	0.98753		0.118	0.674	0.056	0.003	0.001	
MO-4	3938.1	0.98293		0.047	0.322	0.041	0.002	0.002	
NC-5	4500.0	0.97752		0.071	0.464	0.046	0.001	0.0	
NC-6	6500.0	0.96581		0.266	1.618	0.162	0.010	0.002	
NC-7	8500.0	0.94611		0.302	1.581	0.151	0.009	0.002	
NC-8	10180.7	0.92628		0.230	1.254	0.147	0.009	0.004	
CR-9	10180.9		0.99900	0.0	0.0	0.0	0.0	0.0	0.0

SPACEBORNE MULTIPROCESSING (M=0, N=1, K=1, Y=0, PD=.95, MTBF=200,000)
 MONTE CARLO SIMULATION OF

CASE 3 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99912	1.00000	0.160	0.958	0.106	0.006	0.002	
CR-2	2963.8			0.0	0.0	0.001	0.001	0.0	0.0
MO-3	3600.0	0.99822		0.049	0.231	0.019	0.0	0.001	
MO-4	3938.1	0.99757		0.014	0.100	0.008	0.0	0.0	
NC-5	4500.0	0.99646	1.00000	0.020	0.165	0.018	0.0	0.0	
NC-6	6500.0	0.99413		0.113	0.623	0.069	0.002	0.002	
NC-7	8500.0	0.98982		0.110	0.677	0.053	0.004	0.001	
NC-8	10180.7	0.98495		0.093	0.542	0.061	0.007	0.001	
CR-9	10180.9			0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING (M=0, N=1, K=1, Y=0, PD=.95, MTBF=500,000)

CASE 4 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PRGB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99972	1.00000	0.056	0.386	0.047	0.005	0.0	0.0
CR-2	2963.8			0.0	0.0	0.0	0.0	0.0	
MO-3	3600.0	0.99957		0.015	0.096	0.012	0.001	0.0	
MO-4	3938.1	0.99921		0.008	0.049	0.008	0.001	0.0	
NC-5	4500.0	0.99930	1.00000	0.018	0.071	0.008	0.001	0.0	0.0
NC-6	6500.0	0.99860		0.042	0.287	0.031	0.001	0.0	
NC-7	8500.0	0.99776		0.031	0.254	0.014	0.002	0.0	
NC-8	10180.7	0.99698		0.045	0.238	0.016	0.0	0.001	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING (M=0, N=1, K=0, Y=0, PD=.95, MTBF= 75,000)

CASE 5 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99440	1.00000	0.405	2.610	0.138	0.012	0.0	0.0
CR-2	2963.8			0.0	0.0	0.0	0.0	0.0	
MO-3	3600.0	0.98538		0.104	0.616	0.031	0.001	0.0	
MO-4	3938.1	0.97546		0.058	0.314	0.012	0.001	0.0	
NC-5	4500.0	0.97396	1.00000	0.071	0.440	0.024	0.003	0.001	0.0
NC-6	6500.0	0.95879		0.239	1.511	0.092	0.006	0.003	
NC-7	8500.0	0.93135		0.248	1.587	0.092	0.010	0.0	
NC-8	10180.7	0.90549		0.202	1.158	0.092	0.005	0.002	
CR-9	10180.9		1.00000	0.0	0.0	0.059	0.005	0.002	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING (M=0, N=1, K=0, Y=0, PD=.95, MTRF=200,000)

CASE 6 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99901	1.00000	0.149	0.962	0.056	0.002	0.0	0.0
CR-2	2963.8			0.0	0.001	0.0	0.0	0.0	
MD-3	3600.0	0.99661		0.024	0.187	0.010	0.002	0.0	
MD-4	3938.1	0.99383		0.016	0.123	0.003	0.001	0.0	
NC-5	4500.0	0.99572		0.027	0.189	0.010	0.002	0.0	
NC-6	6500.0	0.99269		0.120	0.716	0.029	0.002	0.0	
NC-7	8500.0	0.98655	1.00000	0.120	0.615	0.038	0.001	0.001	0.0
NC-8	10180.7	0.98009		0.093	0.527	0.001	0.001	0.0	
CR-9	10180.9			0.0	0.0	0.036	0.001	0.0	
						0.0	0.0	0.0	

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING (M=0, N=1, K=0, Y=0, PD=.95, MTBF=500,000)

CASE 7 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99966	1.00000	0.069	0.417	0.015	0.002	0.0	0.0
CR-2	2963.8			0.0	0.0	0.0	0.0	0.0	
MO-3	3600.0	0.99880		0.017	0.080	0.004	0.0	0.001	
MO-4	3938.1	0.99810		0.007	0.040	0.0	0.0	0.0	
NC-5	4500.0	0.99863	1.00000	0.011	0.062	0.001	0.0	0.0	0.0
NC-6	6500.0	0.99797		0.042	0.282	0.022	0.002	0.0	
NC-7	8500.0	0.99711		0.038	0.227	0.015	0.002	0.0	
NC-8	10180.7	0.99635		0.035	0.219	0.016	0.0	0.0	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING (M=0, N=0, K=0, Y=1, PD=.95, MTBF=200,000)

CASE 9 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99257	1.00000	0.151	0.924	0.046	0.013	0.0	0.0
CR-2	2963.8			0.001	0.002	0.0	0.0	0.0	
MO-3	3600.0	0.98386		0.030	0.207	0.015	0.002	0.0	
MO-4	3938.1	0.97862		0.027	0.121	0.006	0.0	0.001	
NC-5	4500.0	0.97823	1.00000	0.029	0.169	0.009	0.001	0.0	0.0
NC-6	6500.0	0.97157		0.111	0.630	0.032	0.006	0.0	
NC-7	8500.0	0.96132		0.108	0.618	0.034	0.005	0.0	
NC-8	10180.7	0.95126		0.078	0.469	0.035	0.005	0.001	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING M=0,N=C,K=0,Y=0,PB=.005,MTBF= 75,000,0/0=1-2

CASE 10 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2562.8	C.58C25		0.454	2.391	0.154	0.012	0.0	
CR-2	2963.8		1.00000	0.0	0.0	0.0	0.0	0.0	0.0
MO-3	3600.0	C.96C14		0.109	0.549	0.026	0.002	0.0	
MO-4	3938.1	C.94816		0.057	0.333	0.023	0.0	0.0	
NC-5	4500.0	C.94C70		0.081	0.437	0.031	0.001	0.0	
NC-6	6500.0	0.92051		0.263	1.485	0.094	0.005	0.001	
NC-7	8500.0	C.88953		0.238	1.407	0.089	0.009	0.0	
NC-8	10180.7	C.86166		0.201	1.176	0.058	0.003	0.001	
CR-9	10180.9		C.99000	0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
 SPACERDRNF MULTIPROCESSING M=0,N=0,K=0,Y=0,PB=.005,MTBF=200,000,0/0=1-2

CASE 11 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.55250		0.163	0.929	0.034	0.006	0.0	
CR-2	2962.8		1.00000	0.0	0.001	0.0	0.0	0.0	0.0
MC-3	3600.0	0.58447		0.035	0.216	0.010	0.002	0.0	
MO-4	3938.1	0.97929		0.029	0.131	0.009	0.0	0.0	
NC-5	4500.0	0.57758		0.025	0.171	0.006	0.0	0.0	
NC-6	6500.0	0.97013		0.105	0.633	0.039	0.004	0.0	
NC-7	8500.0	0.95926		0.099	0.568	0.038	0.004	0.0	
NC-8	10180.7	0.94926		0.092	0.505	0.032	0.001	0.0	0.0
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	

MONTE CARLO SIMULATION OF
SPACERORNE MULTIPROCESSING M=0,N=0,K=0,Y=0,PB=.005,MTBF=500,000,D/O=1-2

CASE 12 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.59661		0.066	0.363	0.020	0.002	0.0	
CR-2	2963.8		1.00000	0.0	0.0	0.0	0.0	0.0	0.0
MO-3	3600.0	0.59360		0.015	0.098	0.006	0.001	0.0	
MO-4	1938.1	0.99077		0.011	0.051	0.005	0.001	0.0	
NC-5	500.0	0.55138		0.011	0.065	0.003	0.0	0.0	
NC-6	6500.0	0.58840		0.045	0.247	0.011	0.002	0.0	
NC-7	8500.0	0.58411		0.053	0.259	0.020	0.004	0.0	
NC-8	10180.7	0.98076		0.030	0.186	0.010	0.001	0.0	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	0.0

SPACEBORNE MULTIPROCESSING MONTE CARLO SIMULATION OF
M=0,N=0,K=0,Y=1,PB=.005,MTBF= 75,000,D/O=1-2

CASE 13 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.97578		0.429	2.396	0.115	0.025	0.0	
CR-2	2963.8		0.99900	0.0	0.001	0.0	0.0	0.0	0.001
MO-3	3600.0	0.95760		0.092	0.546	0.038	0.004	0.0	
MO-4	3938.1	0.94460		0.047	0.288	0.022	0.002	0.0	
NC-5	4500.0	0.94292		0.079	0.419	0.022	0.005	0.0	
NC-6	6500.0	0.92339		0.248	1.497	0.104	0.018	0.002	
NC-7	8500.0	0.89436		0.262	1.465	0.081	0.012	0.001	
NC-8	10180.7	0.86739	0.99700	0.225	1.208	0.071	0.016	0.001	
CR-9	10180.9			0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING M=0,N=0,K=0,Y=1,PR=.005,MTBF=500,000,0/0=1-2

CASE 14 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2562.8	C.99656		0.053	0.363	0.025	0.005	0.0	
CR-2	2963.8		1.00000	0.0	0.0	0.0	0.0	0.0	0.0
MO-3	3600.0	C.99405		0.013	0.069	0.005	0.001	0.0	
MO-4	3938.1	C.99227		0.007	0.033	0.001	0.0	0.0	
NC-5	4500.0	C.99212		0.011	0.076	0.003	0.0	0.0	
NC-6	6500.0	C.58557		0.046	0.241	0.013	0.002	0.0	
NC-7	8500.0	C.98522		0.037	0.232	0.020	0.002	0.001	
NC-8	10180.7	C.98131		0.029	0.198	0.014	0.004	0.0	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING M=0,N=0,K=1,Y=0,PB=.005,MTBF= 75,000,0/0=1-2

CASE 15 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROR SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2562.8	0.57542		0.455	2.485	0.263	0.011	0.002	
CR-2	2563.8		1.00000	0.0	0.002	0.0	0.0	0.0	0.0
MD-3	3600.0	C.96105		0.106	0.579	0.063	0.002	0.001	
MD-4	3938.1	C.95420		0.045	0.293	0.035	0.003	0.0	
NC-5	4500.0	C.94451		0.075	0.477	0.048	0.005	0.0	
NC-6	6500.0	0.92773		0.324	1.605	0.186	0.012	0.002	
NC-7	8500.0	0.90363		0.309	1.465	0.163	0.013	0.002	
NC-8	10180.7	C.88287		0.228	1.225	0.132	0.008	0.003	
CR-9	10180.9		0.98800	0.0	0.0	0.0	0.0	0.0	0.0

SPACERORNE MULTIPROCESSING MONTE CARLO SIMULATION OF
M=0,N=0,K=1,Y=0,PB=.005,MTBF=200,000,0/0=1-2

CASE 16 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	C.99226		0.136	0.903	0.104	0.007	0.002	
CR-2	2963.8		1.00000	0.0	0.0	0.0	0.0	0.0	0.0
MO-3	3600.0	0.98537		0.046	0.217	0.024	0.002	0.0	
MO-4	3938.1	C.58295		0.027	0.118	0.011	0.0	0.001	
NC-5	4500.0	0.97544		0.025	0.176	0.009	0.0	0.0	
NC-6	6500.0	0.97283		0.103	0.638	0.066	0.004	0.0	
NC-7	8500.0	0.96304		0.110	0.597	0.064	0.005	0.0	
NC-8	10180.7	C.95440		0.090	0.522	0.050	0.004	0.001	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING M=0,N=0,K=1,Y=0,PB=.005,MTBF=500,000,D/D=1-2

CASE 17 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS. FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	C.55732		0.043	0.323	0.038	0.001	0.0	
CR-2	2962.8		1.00000	0.0	0.0	0.0	0.0	0.0	0.0
MO-3	3600.0	C.99443		0.022	0.105	0.009	0.002	0.0	
MO-4	3938.1	C.99317		0.010	0.048	0.004	0.0	0.0	
NC-5	4500.0	0.99212		0.010	0.071	0.004	0.001	0.0	
NC-6	6500.0	C.98929		0.043	0.266	0.020	0.0	0.0	
NC-7	8500.0	C.58518		0.049	0.264	0.024	0.001	0.0	
NC-8	10180.7	C.58149		0.031	0.197	0.024	0.002	0.0	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.002	0.0	0.0

SPACEBORNE MULTIPROCESSING MONTE CARLO SIMULATION OF
M=0,N=0,K=1,Y=0,PB=.005,MTBF= 75,000,D/O=1-1

CASE 18 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2562.8	0.58005		0.489	2.805	0.284	0.017	0.003	
CR-2	2563.8		1.00000	0.0	0.0	0.0	0.0	0.0	0.0
MO-3	3600.0	0.95603		0.084	0.578	0.076	0.005	0.001	
MO-4	3938.1	0.94884		0.063	0.314	0.033	0.0	0.0	
NC-5	4500.0	0.94495		0.083	0.487	0.064	0.004	0.0	
NC-6	6500.0	0.92900		0.289	1.720	0.186	0.007	0.003	
NC-7	8500.0	0.90475		0.301	1.695	0.180	0.011	0.003	
NC-8	10180.7	0.88297		0.262	1.377	0.155	0.009	0.004	
CR-9	10180.9		0.96700	0.0	0.0	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING M=0,N=0,K=1,Y=0,PB=.005,MTBF=200,000,0/0=1-1

CASE 19 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	C.99229	1.00000	0.169	1.106	0.115	0.003	0.001	0.0
CR-2	2963.8			0.0	0.001	0.0	0.0	0.0	
MD-3	3600.0	C.58314		0.030	0.213	0.025	0.001	0.0	
MD-4	3938.1	C.58045		0.021	0.130	0.009	0.0	0.0	
NC-5	4500.0	0.57530	1.00000	0.038	0.189	0.022	0.001	0.0	0.0
NC-6	6500.0	C.97283		0.119	0.701	0.081	0.003	0.004	
NC-7	8500.0	C.56284		0.134	0.704	0.071	0.003	0.0	
NC-8	10180.7	C.95401		0.127	0.605	0.071	0.005	0.001	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	0.0

SPACEBORNE MULTIPROCESSING MONTE CARLO SIMULATION OF
M=0,N=0,K=1,Y=0,PB=.005,MTBF= 75,000,D/O=1-5

CASE 20 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2562.8	0.58085		0.357	2.168	0.268	0.014	0.005	
CR-2	2962.8		1.00000	0.0	0.001	0.0	0.0	0.0	0.0
MO-3	3600.0	0.96476		0.104	0.604	0.054	0.007	0.001	
MO-4	3938.1	0.95627		0.057	0.345	0.042	0.004	0.0	
NC-5	4500.0	0.94588		0.065	0.438	0.041	0.001	0.001	
NC-6	6500.0	0.52980		0.233	1.348	0.157	0.0	0.002	
NC-7	8500.0	0.50612		0.258	1.391	0.135	0.005	0.004	
NC-8	10180.7	0.88414		0.199	1.130	0.125	0.003	0.0	
CR-9	10180.9		0.59800	0.0	0.001	0.0	0.0	0.0	0.0

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING M=0,N=0,K=1,Y=0,PB=.005,MTBF=200,000,0/0=1-5

CASE 21 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	C.99204		0.138	0.899	0.072	0.005	0.0	
CR-2	2963.8		1.00000	0.0	0.0	0.0	0.0	0.0	0.0
MC-3	3600.0	C.98591		0.035	0.228	0.032	0.001	0.0	
MO-4	3938.1	0.58309		0.012	0.110	0.018	0.001	0.0	
NC-5	4500.0	C.97861		0.017	0.149	0.016	0.001	0.002	
NC-6	6500.0	C.97158		0.097	0.591	0.063	0.002	0.001	
NC-7	8500.0	C.96170		0.109	0.577	0.071	0.003	0.002	
NC-8	10180.7	0.95308		0.077	0.455	0.054	0.001	0.0	0.0
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	

MONTE CARLO SIMULATION OF
SPACEBORNE MULTIPROCESSING M=0,N=0,K=1,Y=0,PB=.01,MTBF=200,000,0/D=1-2

CASE 22 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.99255	1.00000	0.135	0.902	0.091	0.013	0.0	0.0
CR-2	2963.8			0.0	0.0	0.0	0.0	0.0	
MD-3	3600.0	0.98553		0.048	0.232	0.027	0.003	0.001	
MD-4	3938.1	0.58270	1.00000	0.018	0.114	0.018	0.001	0.0	0.0
NC-5	4500.0	0.57914		0.039	0.188	0.022	0.002	0.001	
NC-6	6500.0	0.97275		0.102	0.576	0.048	0.006	0.002	
NC-7	8500.0	0.56387	1.00000	0.103	0.589	0.066	0.006	0.001	0.0
NC-8	10180.7	0.95447		0.082	0.544	0.070	0.009	0.0	
CR-9	10180.9			0.0	0.0	0.0	0.0	0.0	

MONTE CARLO SIMULATION OF
SPACERBORNE MULTIPROCESSING M=0,N=0,K=1,Y=0,PB=.02,MTBF=200,000,U/D=1-2

CASE 23 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.59274		0.151	0.939	0.103	0.022	0.001	
CR-2	2963.8		1.00000	0.0	0.0	0.0	0.0	0.0	0.0
MO-3	3600.0	0.98531		0.039	0.210	0.024	0.003	0.002	
MO-4	3538.1	0.98252		0.025	0.137	0.012	0.003	0.0	
NC-5	4500.0	0.97521		0.031	0.169	0.021	0.003	0.0	
NC-6	6500.0	0.97241		0.128	0.633	0.076	0.011	0.003	
NC-7	8500.0	0.96296		0.110	0.614	0.077	0.012	0.004	
NC-8	10180.7	0.95353		0.089	0.549	0.053	0.005	0.002	0.0
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	

SPACEBORNE MULTIPROCESSING MONTE CARLO SIMULATION OF
M=0,N=2,K=0,Y=0,PB=.005,MTBF=200,000,D/O=1-2

CASE 24 -- SYSTEM STATISTICS

PHASE	TIME	CELL AVAIL	PROB SUCCESS	CELLS 1-2-3 FAIL	TOTAL CELL FAILURES	GROUP SWITCH FAILURES	IC BUS FAILURES	IG BUS FAILURES	UNDETECT
NC-1	2962.8	0.59950		0.158	1.068	0.045	0.008	0.0	
CR-2	2963.8		1.00000	0.001	0.001	0.0	0.0	0.0	0.0
MO-3	3600.0	C.95675		0.033	0.252	0.015	0.003	0.001	
MO-4	3938.1	C.99465		0.011	0.119	0.003	0.0	0.0	
NC-5	4500.0	C.55850		0.031	0.190	0.012	0.0	0.0	
NC-6	6500.0	0.99720		0.093	0.647	0.033	0.005	0.0	
NC-7	8500.0	C.99457		0.107	0.680	0.043	0.006	0.0	
NC-8	10180.7	C.59029		0.092	0.571	0.027	0.004	0.0	
CR-9	10180.9		1.00000	0.0	0.0	0.0	0.0	0.0	0.0

APPENDIX D. COMMUNICATIONS

D.1 INTRODUCTION

This section of the report discusses the communications within the distributed processor. Primarily this is the communications between the cells and between the groups. Presently the organization is being studied and designed on the basis of using conventional DC transmission of binary signals at the computer clock rate (approx. 2 MHz) and providing for the transmission of a half word (8 bits) in parallel (actually 10 lines may be required when control and parity are considered).

The approaches to communications may be considered as DC or RF. RF approaches modulate a high frequency carrier that is then demodulated to receive the transmitted information. Within the DC approaches, the primary variables are the number of lines provided (the number of bits in parallel) and the clock rate or bit rate of transmission. The RF approaches have many variables that yield many communications systems. Four systems were defined and considered in this section.

1. Simple Time Multiplex: Data is transmitted serially from a cell with only one cell transmitting at a time.
2. Simple Frequency Multiplex: Data is transmitted in parallel over one line from a cell with only one cell transmitting at a time.
3. Full Frequency - Simple Time Multiplex: Data is transmitted serially from a cell with all cells transmitting simultaneously over one line.
4. Full Frequency - Frequency Multiplex: Data is transmitted in parallel over one line from a cell with all cells transmitting simultaneously over this one line.

These RF approaches will be explained in further depth in this section. It should be pointed out that RF approaches to the communication system are particularly attractive since data may be multiplexed by modulating carriers at various frequencies. This offers the possibility of using only one line to each cell for communication purposes. This minimization of the number of connections to each cell is of prime importance since the number of connections have a great effect on reliability. The advantages and effects on the computer organization will be explained in this section.

In addition, it is desired to fabricate the RF circuitry on the same wafer as the cell is mechanized on. RF circuit technology will be examined and discussed in this section also.

D.2 COMMUNICATION REQUIREMENTS

The distributed processor organization uses approximately 20 cells per group and two group switches per group. The basic internal computer word length is 16 bits. As far as communications is concerned parity and a control bit may add two more bits to every "word" communicated from cell to cell. Therefore, the basic word length from the communications standpoint will be 18 bits. Each cell is to be an individual wafer so

that the main communication problem is that of communicating between cells (wafers) in a group (probably on one board).

The data rates have not been established firmly at this time. However, they are not expected to be severe and a nominal figure of 10 megabits per second will be assumed (total rate per group).

Reliability is the prime requirement; therefore, this requires that the communication system use as few leads to each cell as possible (primary importance) and require a minimum amount of circuitry for its mechanization (secondary importance).

D.3 RF CIRCUIT TECHNOLOGY

This section will present a brief discussion of RF circuits that may be fabricated on the same wafer as used for the digital devices to mechanize a cell. The feasibility of using an internal RF link to communicate from one section of the computer to another is within the scope of present technologies. All of the integrated circuit RF components necessary to construct an internal communications system of this type have been operated in the laboratory. The fabrication of these components is also compatible with the thin film techniques which are used in the construction of high speed digital integrated circuits. As in the lower frequency circuits the size of the integrated circuit is controlled by the passive circuit elements. At the higher RF frequencies circuit elements are developed from lengths of transmission line which perform the electrical function of capacitors and inductors. The physical circuit size will be related directly to the wavelength of the RF energy propagated on a microstrip transmission line. In order to illustrate the size reduction of these circuits possible with various substrate materials, Figure D-1 shows both the reduction of the linear dimensions and the reduction of area that results at a given frequency for various values of substrate dielectric constant. As can be seen from this chart, the materials presently in use and under consideration for use in integrated circuits have dielectric constants in the vicinity of 10. This would result in linear dimensions which are 38 percent and equivalent areas which are 15 percent of those when the dielectric constant of the transmission line is unity (air line). It should also be noted that if the system is operated at a higher frequency the linear reduction in circuit size will be directly proportional to the frequency increase. Both of these factors would determine the total area per wafer occupied by the RF communication link circuits.

The frequency of operation for an RF computer communication system will be limited at the upper frequency by the RF power output of available devices. At the lower frequency of operation, the size of the passive components will increase and the area required for communication purposes will increase, the amount of area available for the RF circuits will therefore determine the lower frequency. A plot of frequency vs. power output of a group of Solid State RF power generators utilizing the bulk effects in semiconductors is shown in Figure D-2. These devices could be ideally suited for use in the RF computer communication system because of their small size and simplicity. A number of these devices have been fabricated in thin film configuration and are presently being studied to enhance their power generating capability. The data which has been collected in Figure D-2 represents the performance of many solid state bulk effect devices operating in different modes and for comparison also presents the most recent upper limit for transistor operation. This plot compares

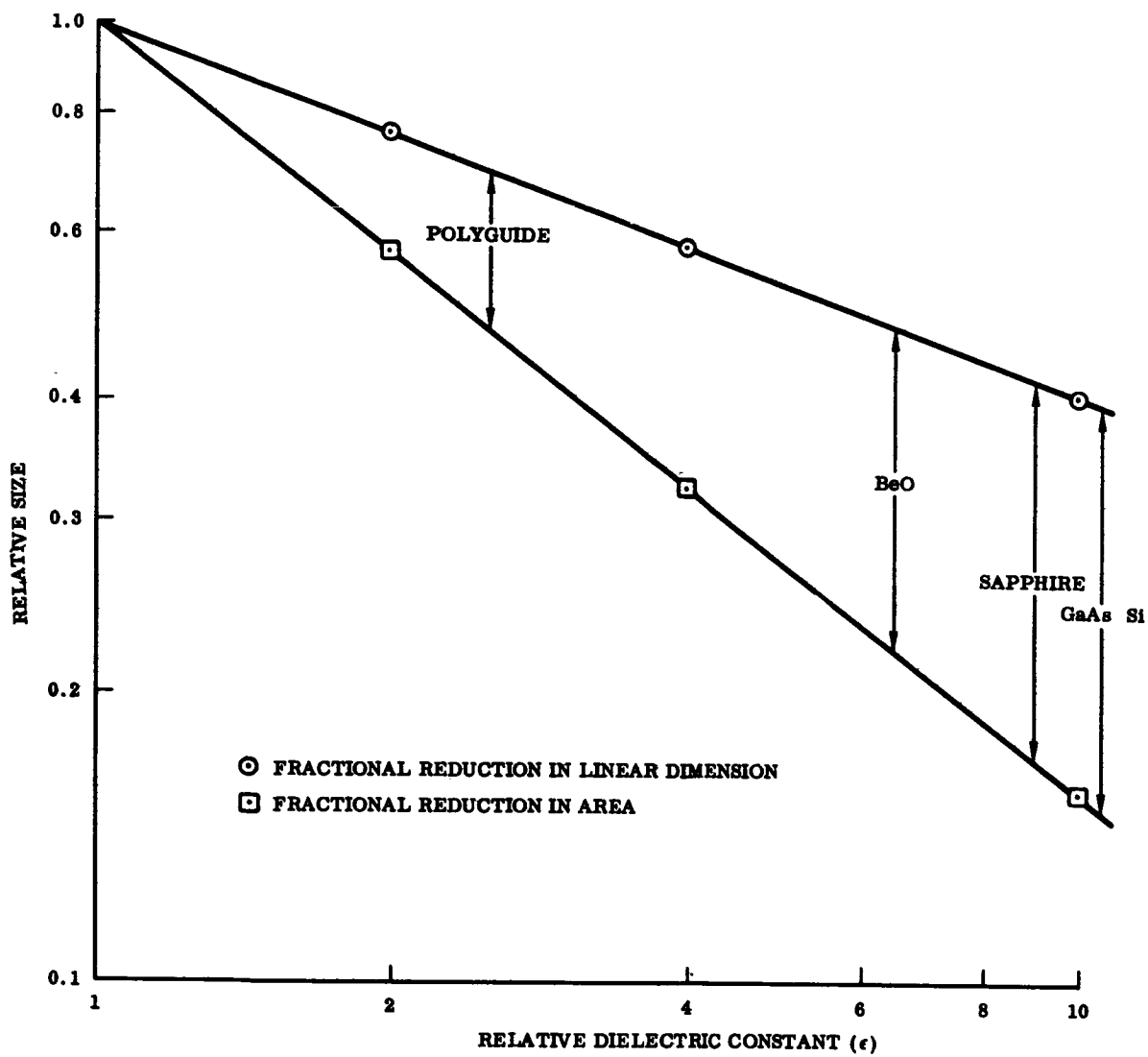


Figure D-1. Dielectric Effects on Size

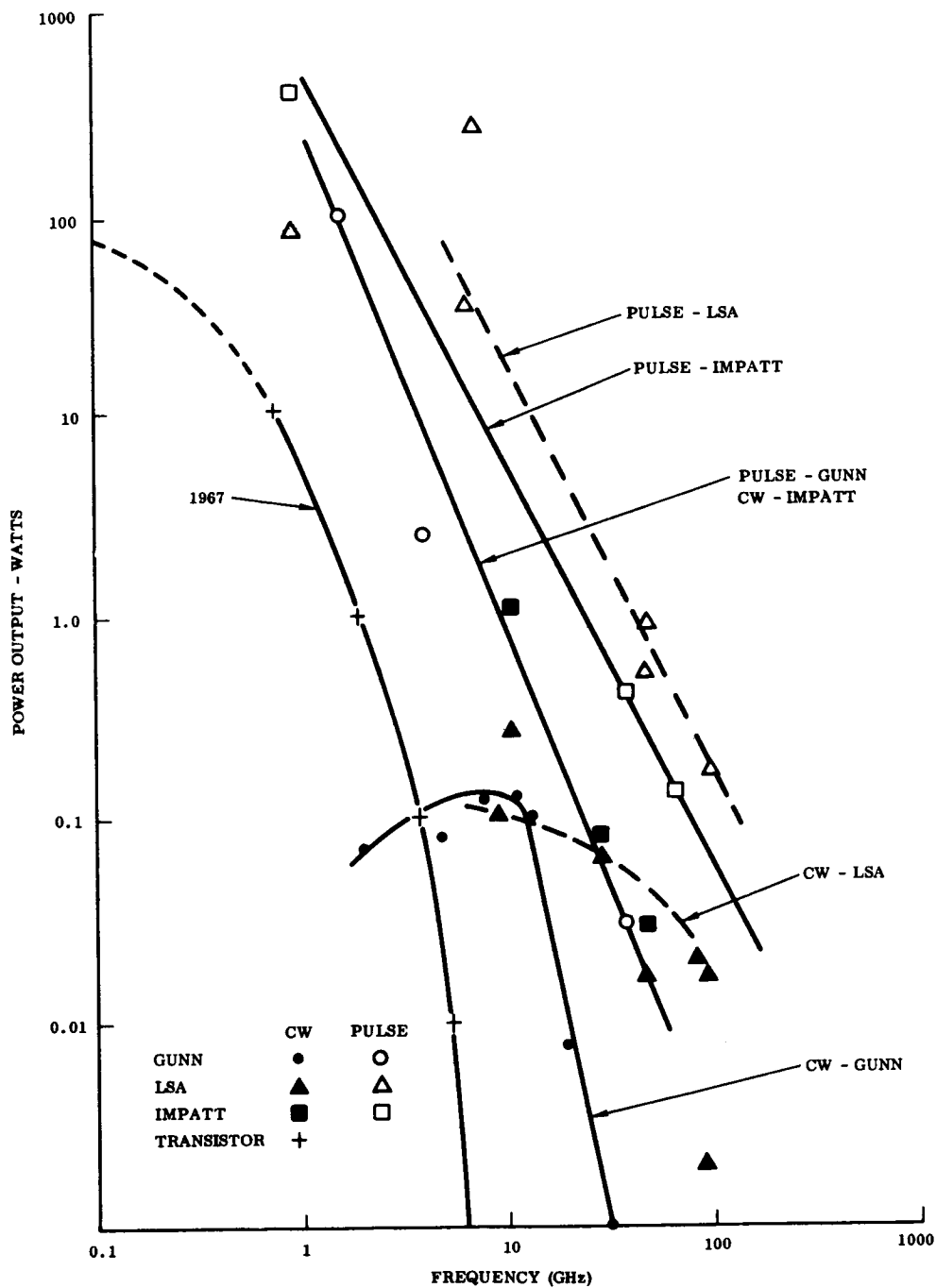


Figure D-2. Solid-State RF Power Oscillators

both the cw and pulse modes of performance that has been achieved for the Gunn, LSA (Limited Space Charge Accumulation), and IMPATT (Impact Avalanche Transit Time) devices. For further discussion on these types of devices the reader is referred to reference 38.

One example of the type of circuit compatibility that is available is the operation of the Gunn device whose frequency is dependent on sample thickness. To obtain RF power at a frequency of 10 GHz the sample thickness required is approximately 10 microns. These thicknesses, especially at the higher frequencies are the same order of magnitude as those which are controlled by thin film processing techniques. The data for the LSA mode of operation as of August 1967 shows that the operation of these devices is feasible up to the region of 90 GHz. The power output levels are greater than that of the Gunn mode oscillators. Since these LSA devices may be used in conjunction with microstrip resonators, the choice of the type of device at the upper RF frequencies would be an LSA mode oscillator for the transmitter source. Since these devices are just in their initial experimental stages, rapid advances in power output, frequency range, and efficiency are expected in the next few years.

The power requirement on the transmitting device will be modest. One of the reasons for this low transmitter power output requirement is due to the use of the highly sensitive detecting devices now available in integrated form. One of the most suitable candidates for this detection function is the Schottky barrier diode. These diodes are compatible with integrated construction techniques and have shown performance which is superior to standard diodes at frequencies up to 90 GHz. The use of this type of a detector would provide compatibility with the present solid state active devices and passive components at the upper RF frequencies. Another reason for a low transmitter power is the short distance (less than 12 inches) that the signals will need to be transmitted.

D. 4 RF COMMUNICATION SYSTEMS

D. 4. 1 Analysis

D. 4. 1. 1 Systems Considerations

This section will present an analysis of the performance of a general r-f communication system that utilizes frequency multiplexing of binary data channels. The results here will apply to specific systems presented in Section D. 4. 2. In the design of a frequency multiplexed communication system, it is common practice to divide a band of frequencies into a number of equally spaced data channels. Each channel has a bandpass filter whose general function is the prevention of interference or crosstalk from signals of other channels as well as the reduction of noise.

In most communication systems, bandwidth limitation is the primary consideration. However, in this intracomputer communication study the tuned circuit components are the limiting factors.

Because the total communication system must be capable of being mounted on an integrated circuit wafer, the overall size of the circuit components such as the quarter wave sections used for the single tuned circuits must be on the order of a few millimeters. This requires that frequencies in the range of 10 GHz or higher must be

utilized. However, the maximum realizable tuned circuit Q's that can be obtained for resonant elements at these frequencies is about 50. For frequencies on the order of 10 GHz to 20 GHz, a Q of 50 results in 3 db bandwidths between 200 to 400 MHz, which is considerably greater than the signal bandwidth (for purposes of this analysis, assumed to be 4 MHz). For this reason, the system design is controlled by circuit limitations rather than data transfer rates.

Shown in Figure D-3 is a plot of the signal to crosstalk ratio as a function of the frequency spacing of the channels and the number of tuned circuits.

The total crosstalk in one channel is the RMS sum of its response to all of the transmitted tones. Although the crosstalk is usually affected by the spectrum of the transmitted signal, the 3 db circuit bandwidth to data spectrum is sufficiently large in this instance so that it can be neglected.

The curves in Figure D-3 were plotted assuming a carrier frequency of 10 GHz. From this figure it can be seen that for a signal to crosstalk ratio of about 20 db at least two tuned circuits and a channel spacing of 400 MHz is required. The overall system bandwidth for 20 channels would then be approximately 20 x 400 MHz or 8 GHz. Also because resonant elements are used for the tuned circuits it is important to limit the overall system bandwidth to less than one half octave to avoid problems with harmonics. This then requires the 20 frequency multiplexed channels to operate between approximately 16 and 24 GHz which is within the current state of the technology.

Consideration of more than approximately 20 channels is given in one of the RF systems to be presented in the next section. This system assigns one channel to each bit in each and every cell. Such a system would require a bandwidth of approximately 70 GHz and an operating frequency range which would extend from 140 to 210 GHz. Such a system does not appear to be technically feasible. Therefore only systems that limit the number of channels to approximately 20 will be considered.

D.4.1.2 ON-OFF Amplitude Modulation Performance Analysis

On-off modulation was chosen as the intra-computer communication technique because of its simplicity in application, its amenability to microminiaturization at operating frequencies in the range of 16 GHz and because an optimum modulation technique is not essential in a system where the signal-to-noise ratio can be maintained at relatively high levels.

Shown in Figure D-4 is a simplified block diagram of an RF communication system. At the output of the envelope detector shown in the block diagram, it is possible to define an envelope function (ρ) where:

$$\rho^2 = (E_s + x)^2 + y^2$$

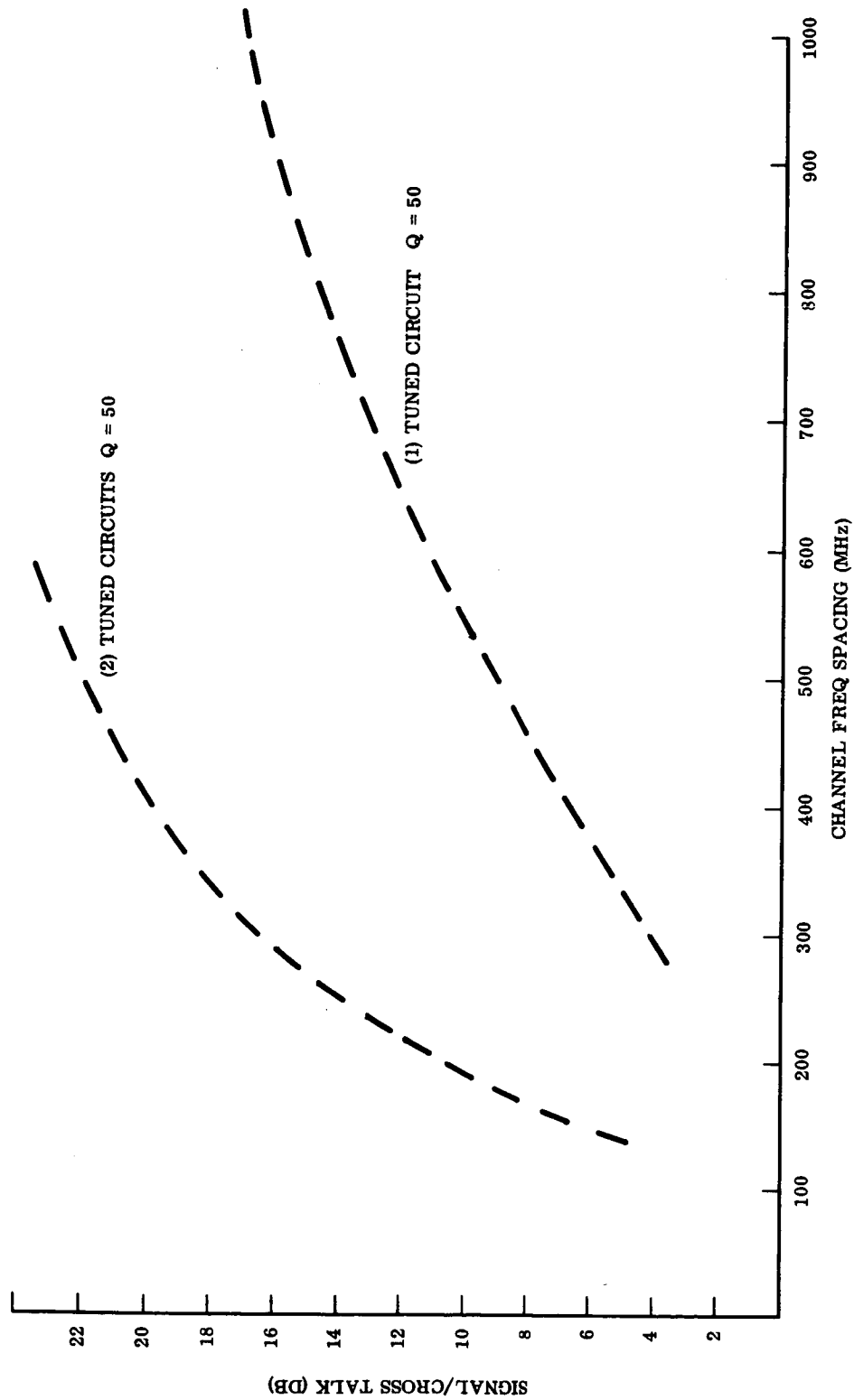


Figure D-3. Signal/Crosstalk Ratio Plot

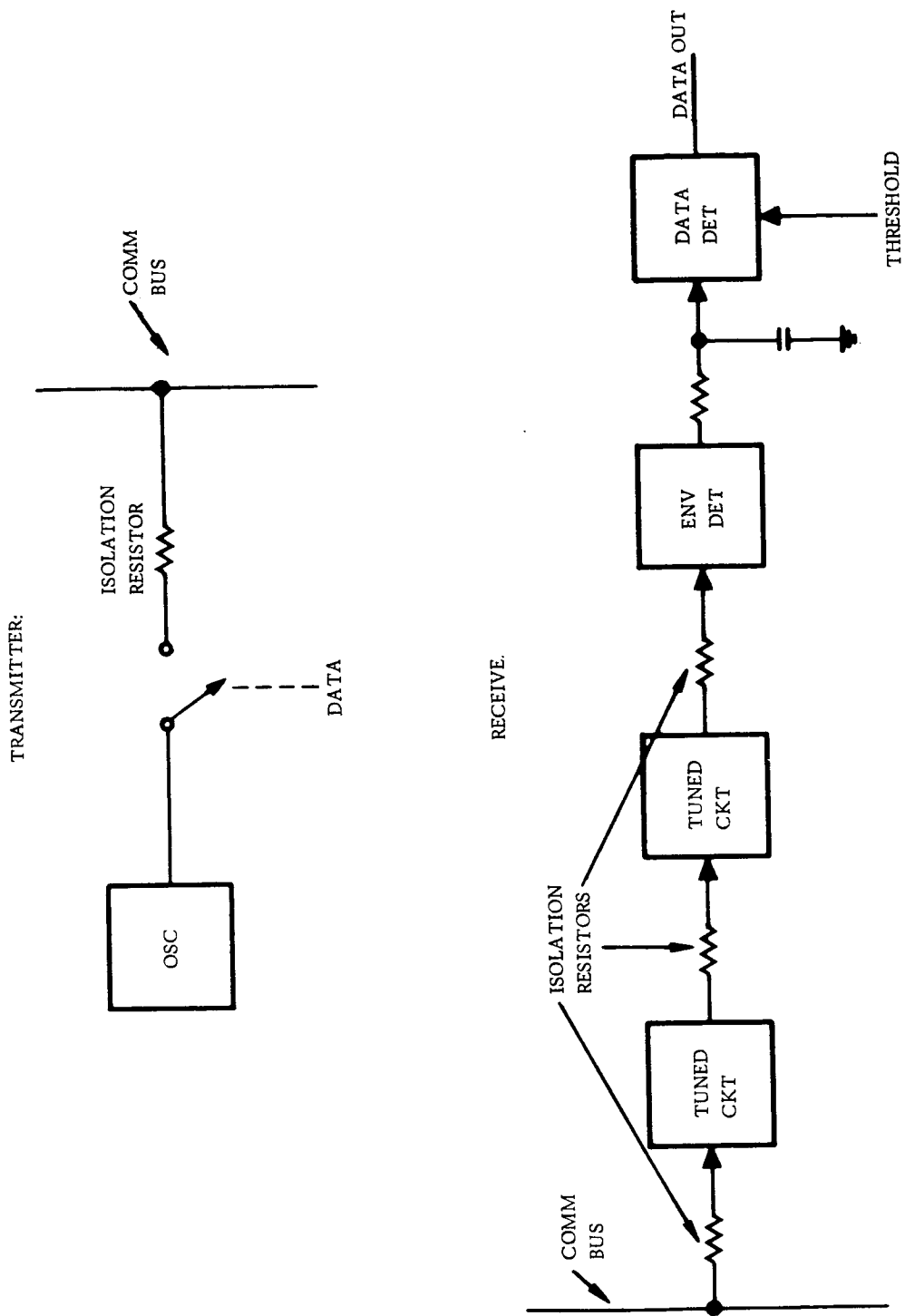


Figure D-4. Intra-Computer Communication System Simplified Block Diagram (Single Channel)

where E_s is the signal voltage when the transmitter is turned on by the data; x and y are assumed to be independent gaussian variables where:

$$p(x) = \frac{\exp(-x^2/2\sigma^2)}{\sqrt{2\pi} \sigma}$$

$$p(y) = \frac{\exp(-y^2/2\sigma^2)}{\sqrt{2\pi} \sigma}$$

are the density functions of these variables and where σ is defined by:

$$\sigma^2 = 2 \int_0^{f_c} \omega(f + f_c) df$$

where $\omega(f + f_c)$ is the spectral density of the noise and f_c is the equivalent lowpass cut-off frequency of the two cascaded single tuned circuits shown in Figure D-4.

The density function of the envelope when the transmitter is on can be shown to be:

$$p_{s,n}(\rho) = \frac{\rho}{\sigma^2} I_0\left(\frac{E_s \rho}{\sigma^2}\right) \exp\left(\frac{-\rho^2 + E_s^2}{2\sigma^2}\right)$$

where $I_0(E_s \rho / \sigma^2)$ is the zero-order Bessel function of imaginary argument.

When the transmitter is turned off, $E_s = 0$, and the probability density function of the envelope without signal is simply:

$$p_n(\rho) = \frac{\rho}{\sigma^2} \exp\left(\frac{-\rho^2}{2\sigma^2}\right)$$

The probability distribution function of the envelope without the transmitter signal present is:

$$P_n(\rho) = \int_0^\rho q_n(\rho) d\rho = 1 - \exp\left(\frac{-\rho^2}{2\sigma^2}\right)$$

7

which is the so-called Rayleigh distribution which applies to the envelope of narrow-band gaussian noise. The distribution function $P_{s,n}(\rho, E_s)$ representing the envelope of the sum of a sine wave and gaussian noise is given by:

$$P_{s,n}(\rho, E_s) = 1 - Q\left(\frac{E_s}{\sigma}, \frac{\rho}{\sigma}\right)$$

in terms of the Q function where:

$$Q(a, b) = \exp(-a^2/2) \int_b^\infty x \exp(-x^2/2) I_0(ax) dx$$

where:

$$I_0(ax) = \frac{1}{\pi} \int_{-1}^1 (1-t^2)^{-1/2} \exp(axt) dt$$

Numerical integration can be used to evaluate the integral to the accuracy desired. $Q_{s,n}(\rho, E_s)$ is the Rice distribution which degenerates to the Rayleigh distribution when E_s is zero and approximates a gaussian distribution when $E_s \gg \sigma^2$.

The probability of error of the threshold detector which follows the envelope detector can now be calculated. The threshold is set at kE_s so that a mark is detected if the detector output ρ is greater than kE_s and a space is detected if ρ is less than kE_s . If a space signal is sent, that is if the transmitter is not on, the signal is erroneously read as a mark if ρ exceeds kE_s . Hence the probability of this error is given by:

$$P(M/S) = 1 - P_n(kE_s) = \exp\left(-\frac{k^2 E_s^2}{2\sigma^2}\right)$$

If a mark is sent, the envelope is that of the noise plus the transmitted signal and an error is made if ρ is less than kE_s and the probability of this error is given by:

$$P(S/M) = P_{s,n}(kE_s, E_s)$$

When the signal-to-noise ratio is relatively large the following approximation holds:

$$P(S/M) \cong \frac{\exp\left[-\frac{(1-k)^2 \gamma}{2(1-k)}\right] \sqrt{\frac{k}{\pi \gamma}} \left[1 - \frac{3 + 6k - k^2}{16k(1-k^2)\gamma} + \dots\right]}$$

where $\gamma = E_s^2 / 2\sigma^2$.

In contrast to the case of coherent detection, the probabilities of error are not equal when the threshold is set at $E_s/2$. Shown in Figure D-5 is a graph illustrating the effect of signal-to-noise and threshold variation on the error rate. Although the optimum threshold does vary somewhat as shown in Figure D-5 it does not change sufficiently to cause more than about a 1 db degradation in the overall performance of the system. Shown in Figure D-6 is the performance curve of the system under consideration showing the error rate that would be obtained as a function of the signal-to-noise ratio at the input to the decision device. Also shown are the error rate curves that would be obtained for $k = 0.5$.

Although these performance results were obtained by assuming that the disturbing influence was white gaussian noise, it in fact will most likely be adjacent channel crosstalk, which will consist primarily of the two adjacent channel tones that will not be completely filtered by the channel filter. Therefore, it will not be gaussian and somewhat better performance than that indicated by Figure D-5 can be obtained.

From the analysis in the previous section (D.4.1.1), it was seen that a signal-to-crosstalk ratio of 20 db could be obtained, which based on Figure D-6 and the above comments should result in approximately errorless performance for the intracomputer communication link when considering the effects of crosstalk alone. Other noise sources such as logic noise, etc., have not been considered here, the intent of this section primarily being an investigation of the system operation.

D.4.2 Description of RF Communications Systems

D.4.2.1 Introduction

There are two principal methods of combining several (binary) channels on a single physical bus. One is frequency multiplexing and the other is time multiplexing.

Frequency multiplexing utilizes several carrier frequencies which are independently modulated by parallel data sources. At the receiving station these frequencies are separated by filters or other means and the modulation detected in order to recover the data. For the systems under consideration, only simple on-off modulation will be considered. Then the receiving station simply has to detect the presence or absence of each of the tones.

In time multiplex the transmission time is divided up into time slots and each of the multiplexed channels has a time slot assigned to it. The time slots can correspond to serial bits transmitted from a cell or to the sequence of transmitting times assigned to cells or a combination of these two.

Control and supervision of the cells in the communications subsystem is a difficult question involving the processor organization to some extent. Certain trade offs are involved which cannot be evaluated here. For example, most of the supervision could be concentrated in the group's control cell or most of the supervision could be distributed among the communicating cells. There is a software vs hardware trade off implicit in this dichotomy which requires detailed descriptions of each of the technically feasible systems before the trade offs can be fully specified. This problem is not considered further here except to illustrate systems operation.

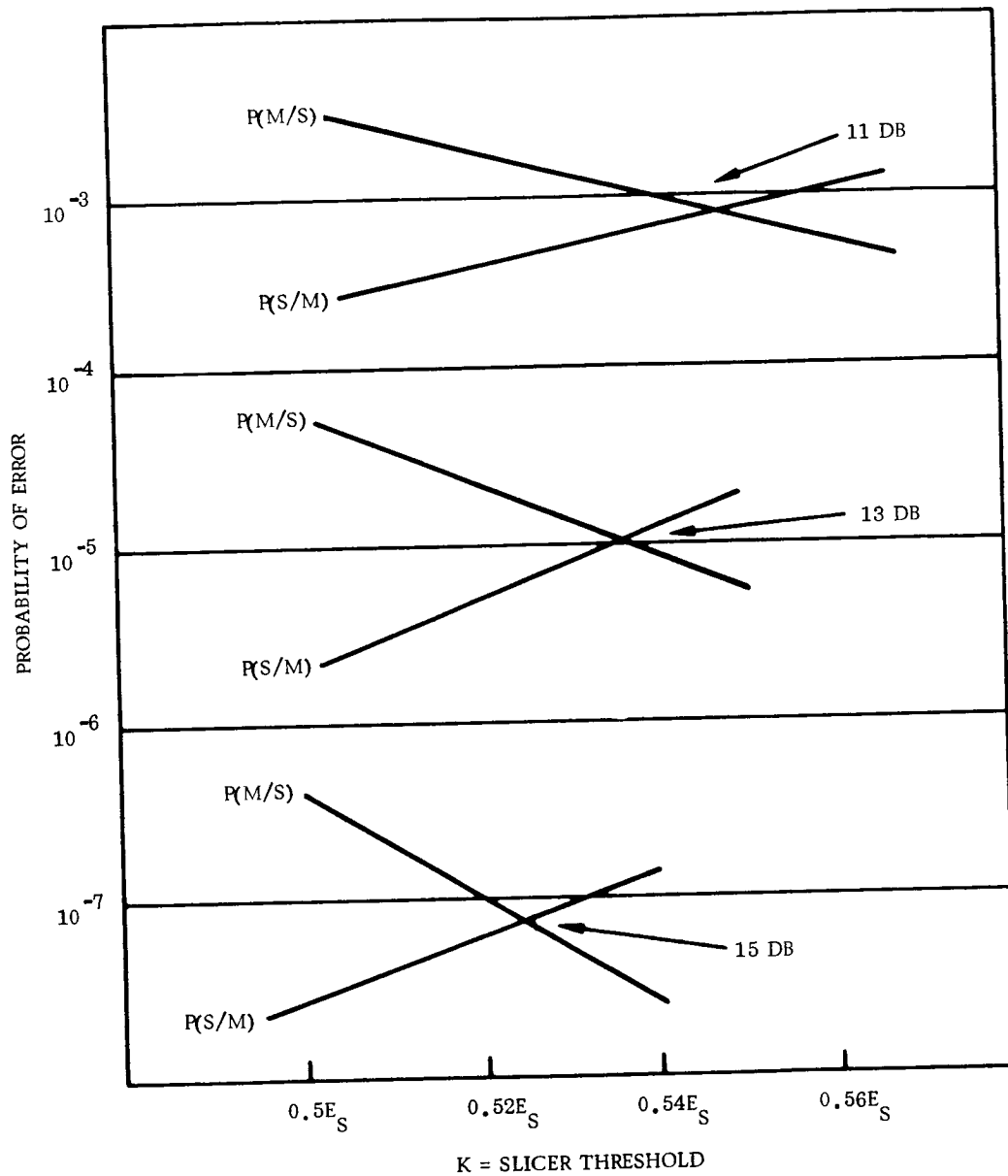


Figure D-5. Probability of Error as a Function of Data, Threshold Level and Signal-to-Noise Ratio

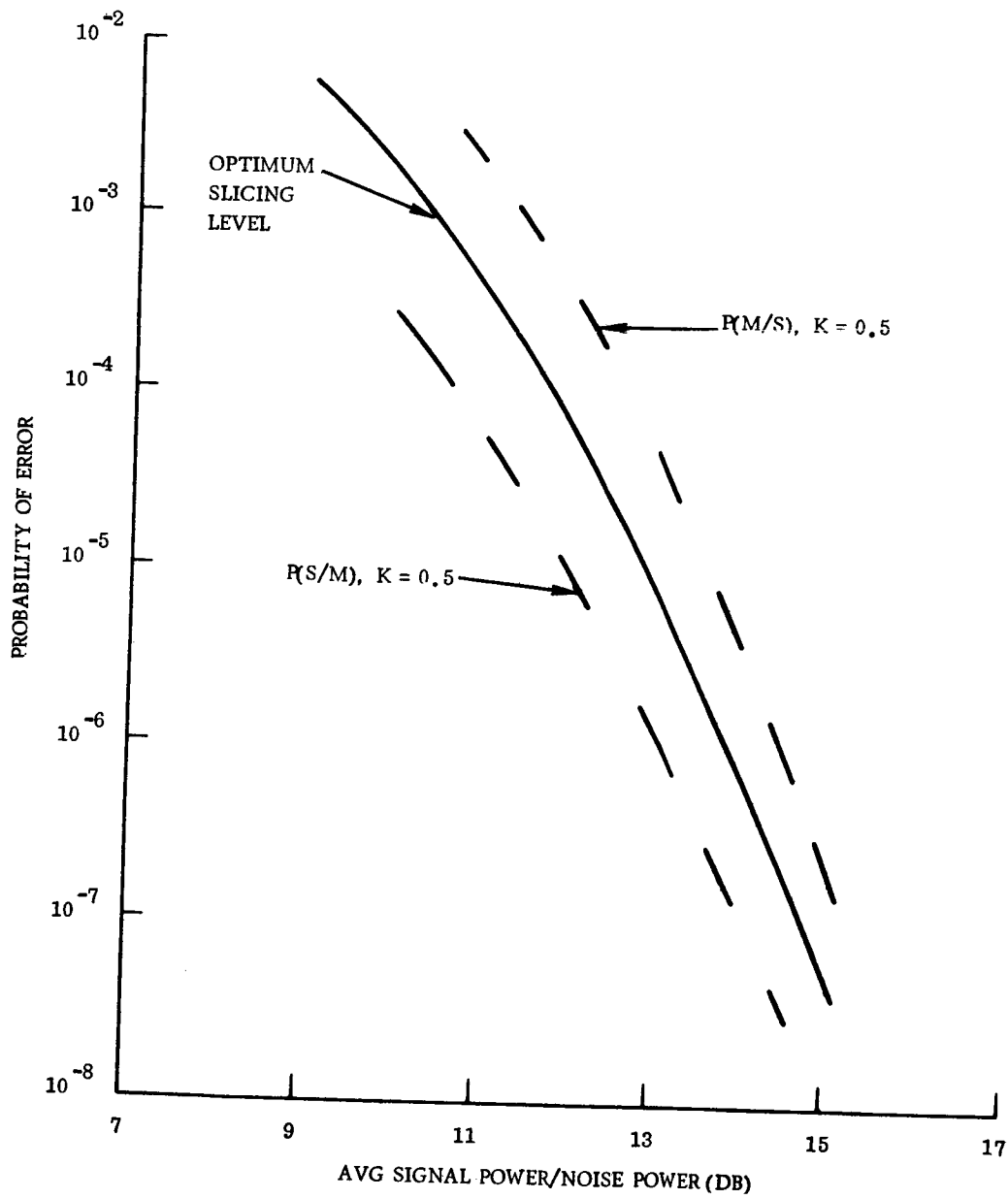


Figure D-6. Theoretical Performance of ON-OFF Amplitude Modulation with Envelope Detection in White Gaussian Noise

In this section block diagrams of four possible RF communications systems are discussed. These are: 1. Simple time multiplex; 2. Simple frequency multiplex; 3. Full frequency - simple time multiplex; and, 4. Full frequency - frequency multiplex.

D. 4. 2. 2 RF System 1 - Simple Time Multiplex

In RF System 1, the words are transmitted serially with only one cell transmitting at a time. (The words are time multiplex; the cells time share the bus.)

Figure D. 7 shows a functional block diagram of a simple time multiplex system. In this system an oscillator is fed through a modulator (an on-off gate) which is controlled by the output of a parallel to serial converter fed by the data register. The parallel to serial converter converts the data word into a serial bit stream which turns the modulator on and off controlling the output of the oscillator. This sequence of carrier pulses is transmitted on the bus to the receiving cells.

Each receiving cell is activated (for reception) by enabling a gate feeding a filter. The filter may not be essential for this system unless switching transients, etc., are severe. The filter feeds a detector which produces an output whenever the oscillator's tone is present on the bus. The output of the detector is fed to a serial to parallel converter which reconstructs the parallel computer word.

This system is almost as simple as DC serial word transmission; it is feasible if the average and peak data transmission demands of the system are not too high for the technology.

It may be possible to speed up the transmission of the serial data words through the use of a fast rise time device in the modulator. In radar work it has been possible to modulate gigahertz carriers using 100 megahertz modulating signals. It thus appears feasible to modulate the output of an oscillator operating in the 10-50 gigahertz range at better than $18 \times 2 = 36$ megahertz rates. If this is possible, then a computer with a basic clock rate of 2 megahertz could transmit 18 bit words serially in each clock period. This implies that simple time multiplexing should impose no delay restrictions on the cells. That is, when the transmitting cell is allotted its time slot for transmission, the transmitting cell could come up with data words for transmission as fast as it can and the time multiplex system would transmit them to the receiving cells.

D. 4. 2. 3 RF System 2 - Simple Frequency Multiplex

Figure D-8 is a functional block diagram of a scheme for frequency multiplexing the 18 bits in a word so that they can be transmitted in parallel from one cell to another. In this scheme the outputs of 18 oscillators operating at 18 different frequencies are controlled by gates operated by the data bits in the 18-bit register. (The gates are the "modulators.") It should be noted that a combination of time multiplexing could be considered here also. For example, 9 oscillators could be used to transmit half a word at a time etc. Thus, the output of oscillator 1 is fed through to the bus whenever d_1 is a 1, and is not fed through to the bus when d_1 is a 0. Similarly, the output of oscillator 2 is fed through when d_2 is a 1 and is not fed through when d_2 is a 0.

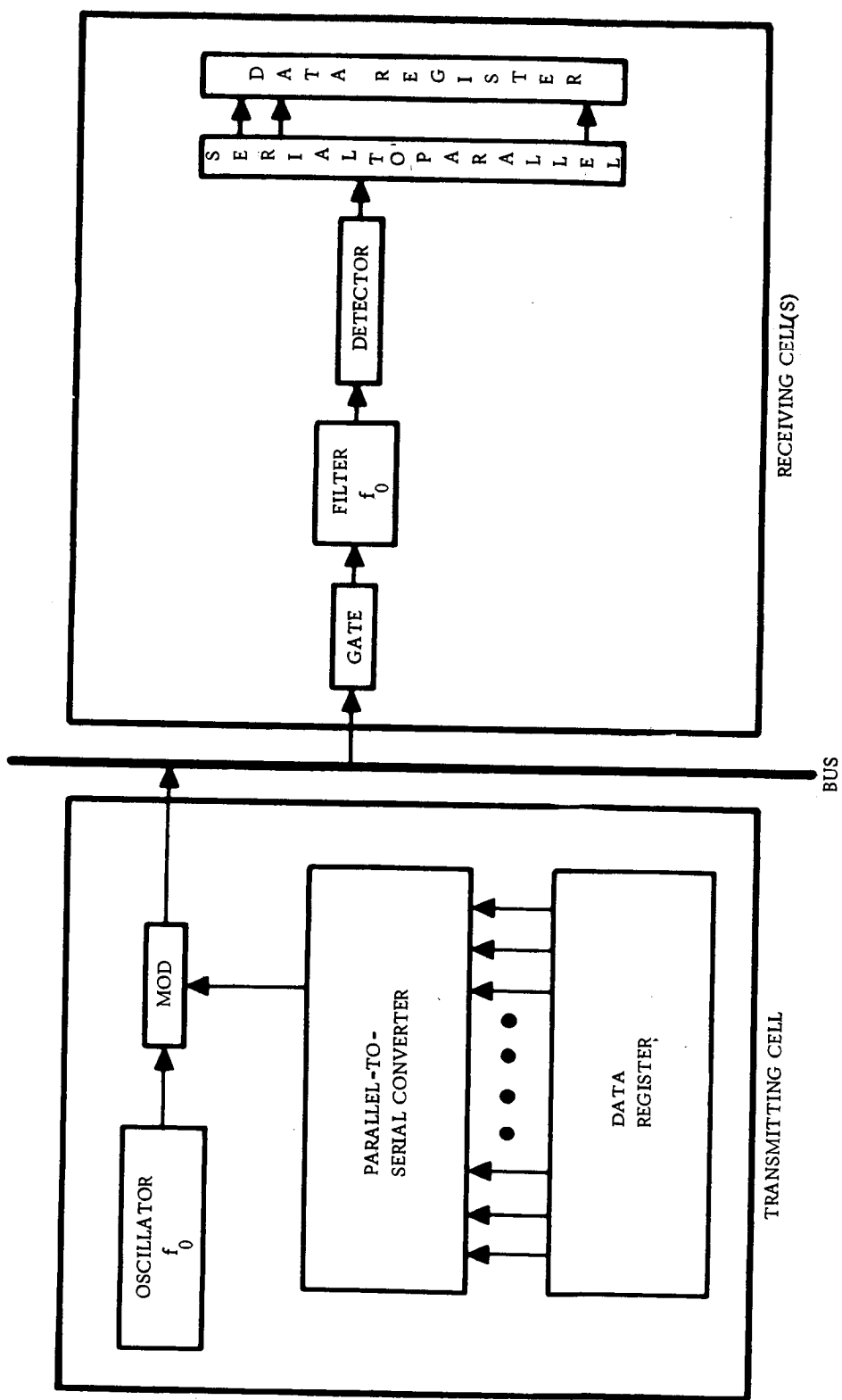


Figure D-7. Simple Time Multiplex

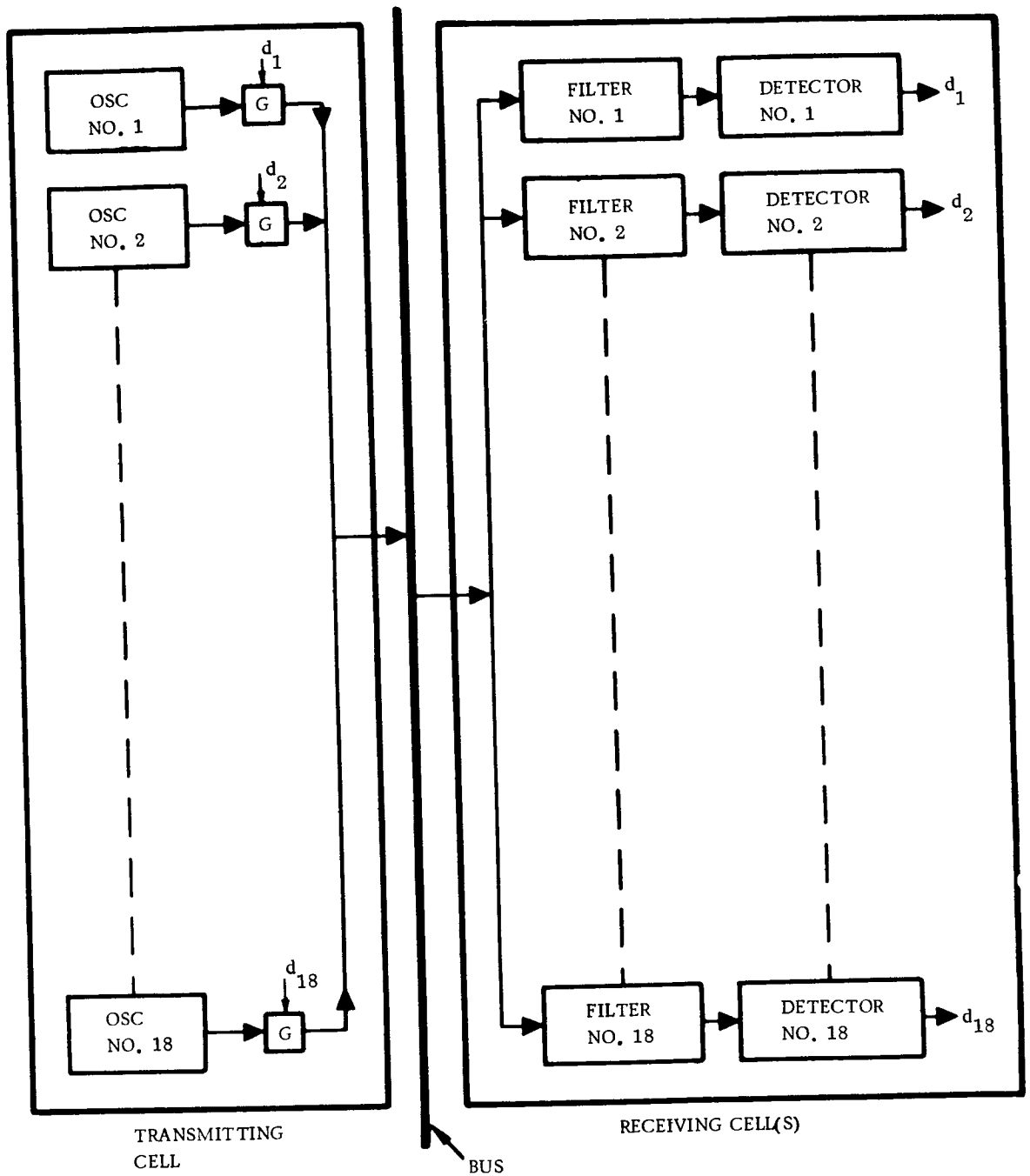


Figure D-8. Simple Frequency Multiplex

and similarly for the outputs of the other oscillators. Thus, a transmitted word appears on the bus as a sub-set of the 18 oscillator frequencies. In this kind of a system a transmitting cell can transmit to one or more receiving cells. Each of these receiving cells, in fact, each of the cells in the system, would have 18 filters corresponding to the 18 frequencies of the oscillators. The outputs of these filters would be fed to threshold detectors to determine the presence or absence of energy at those frequencies. Thus, if d_1 were zero, frequency 1 would not be present on the bus and the output of detector 1 would be a zero. Similarly if d_2 were a 1 the output of oscillator 2, f_2 , would be present on the bus and the output of detector 2 would be a 1. Similarly for the rest of the detectors.

This system is basically a switching system in which one transmitting cell at a time is switched on to the bus and the receiving cells are simultaneously switched on to the bus. The current transmitting cell would transmit its information to the receiving cell. The control cell could monitor the transmission and when it detected an end of transmission it could then activate the next transmitting cell, deactivate all the previously receiving cells and activate a new set of receiving cells. The new transmitting cell would then transmit its information, etc. This process would be repeated to take care of the transmission requirements of the system.

There are various drawbacks to this scheme aside from technological problems. First of all, those cells desiring to transmit must wait in line to get on the bus. This can result in lost time in the cells as well as potential queuing problems. Second, the control of this system requires that a previous set of receiving cells be deactivated and a new set of receiving cells be activated for each transmission period. Unless a separate supervisory bus is provided, this uses up transmission time and again raises the overhead of the system.

Technological problems may rule this system out; i. e., there are definite problems of interference between several oscillators operating simultaneously on a single chip, and there are also problems in connection with loading the bus with a varying number of filters.

The loading problem can be solved in a straightforward manner as illustrated in the detailed example of this system given below. Interference between oscillators operating simultaneously on a single ship is a more difficult problem. Some device, either passive or active, is required to isolate the oscillators from one another. A simple filter may be adequate, but if not, an isolator would very likely have to be developed. At present there is not an adequate isolator; however there may be one by the time the system is ready for development.

A detailed block diagram of the Simple Frequency Multiplex System will be given below. This system is presented in detail since it is expected to be the most demanding as far as the RF circuit mechanization is concerned. The discussion below may be extended to the other three systems in this section. The block diagram in Figure D-9 shows in detail the concept of a twenty channel simple frequency multiplexed communications system in the transmit-receive and terminated mode. A cell in the terminated mode is simply selected not to receive any of the transmitted information. The transmitter section consists of small LSA devices which are tuned at equally spaced frequencies $f_1 - f_{20}$. Each of the outputs to the main line is controlled by

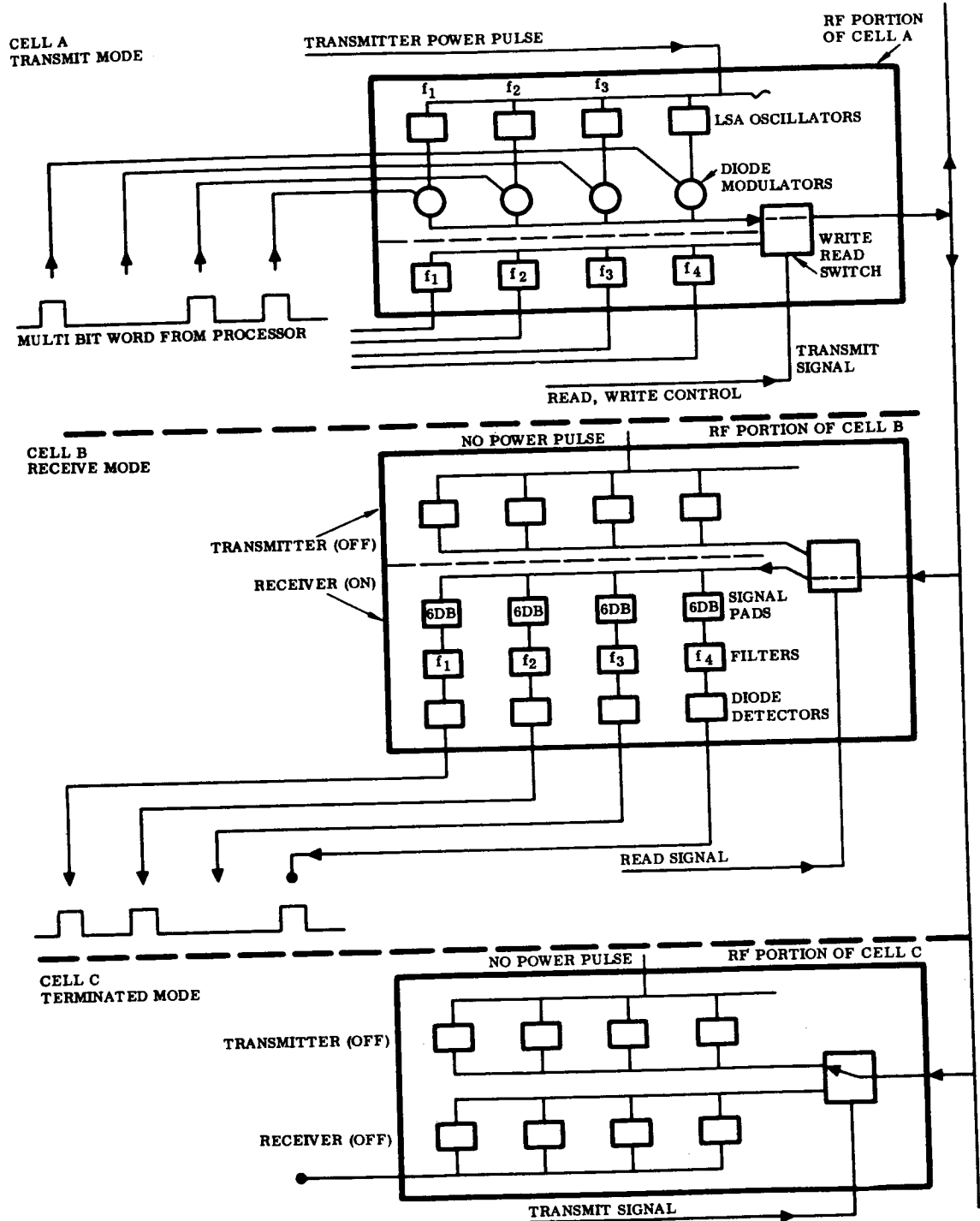


Figure D-9. Detailed Block Diagram of Simple Frequency Multiplex

switching diodes whose state is determined by the processor logic. The main drive for all of the transmitter oscillators is provided by a separate line which provides the necessary power at the proper intervals to generate the RF energy in the solid state oscillators.

One of the primary considerations of system operation is transmitter oscillator power. The following assumptions were made in calculating the transmitter power. The maximum transmission line length between transmitter and receiver will be less than 6 inches. The power will be fanned out to no more than twenty stations. The receiver sensitivity will be that of a typical video detector, - 50 dbm. A filter loss of 2 db will be experienced. The required minimum signal to noise ratio at the detector output will be 10 db. A receiver input attenuator pad of 6 db will be used as a simple method of reducing any mismatch reflections. When all of these factors are taken into account the required transmitter power for each frequency channel is 0.05 mw. The total output power of all 20 transmitters in a cell will be 1 mw. For a twenty cell group the total DC input power requirement will be only 660 mw if a device efficiency of 3 percent is assumed.

The components which will take up the largest area of the chip are the filters which are required to separate the frequency channels at the receiver. The second largest components may be the power dividing networks. The assumption is made that a two section filter is required in each channel and that the power division may be included by using component design which doubles the functions per area. The required separation between transmission lines to provide isolation is included in this calculation. The area per filter section was determined from an experimental filter constructed at 9 GHz on an alumina substrate ($\epsilon = 9$). At this frequency the total area required for the twenty filters in the receiver is 1.6 square inches. If the frequency were doubled, operation centered about 20 GHz, the area would be reduced to 0.4 square inches. This is probably a minimum area since a detailed design will require other functions. The area may be kept close to this by special multi-purpose circuits. For example the tuned circuits required for the LSA transmit mode operation, may be the same tuned circuits as used for the receiver section. These would then be shared by a diode switching scheme. The Schottky barrier diodes in the circuit should tolerate the transmitter RF power well since the minimum burn out level is in the 10 mw region. These diodes have very high frequency properties but unlike most microwave diodes can tolerate a high power level. Not only can these diodes be used as detectors but they can be made to function as a low power high speed switch. This leads to the possibility of using a more sophisticated switching scheme in which a combination series parallel transmission arrangement might be utilized.

D.4.2.4 RF System 3 - Full Frequency - Simple Time Multiplex

The simple time multiplexing scheme described in section D.4.2.2 can be combined with frequency multiplexing into a serial-word time multiplex, with cells frequency multiplexed as shown in Figure D-10. One possible way of obtaining time-frequency multiplex would be to have each of the 20 cells assigned a particular individual frequency. Its oscillator would then be tuned to that frequency. Then, to set up a broadcast network, the receiving cells would be actuated by gating on the appropriate filter. A particular configuration is shown as an example in Figure D-10 in which there are three transmitting cells which are assumed to be cells 1, 2, and 3,

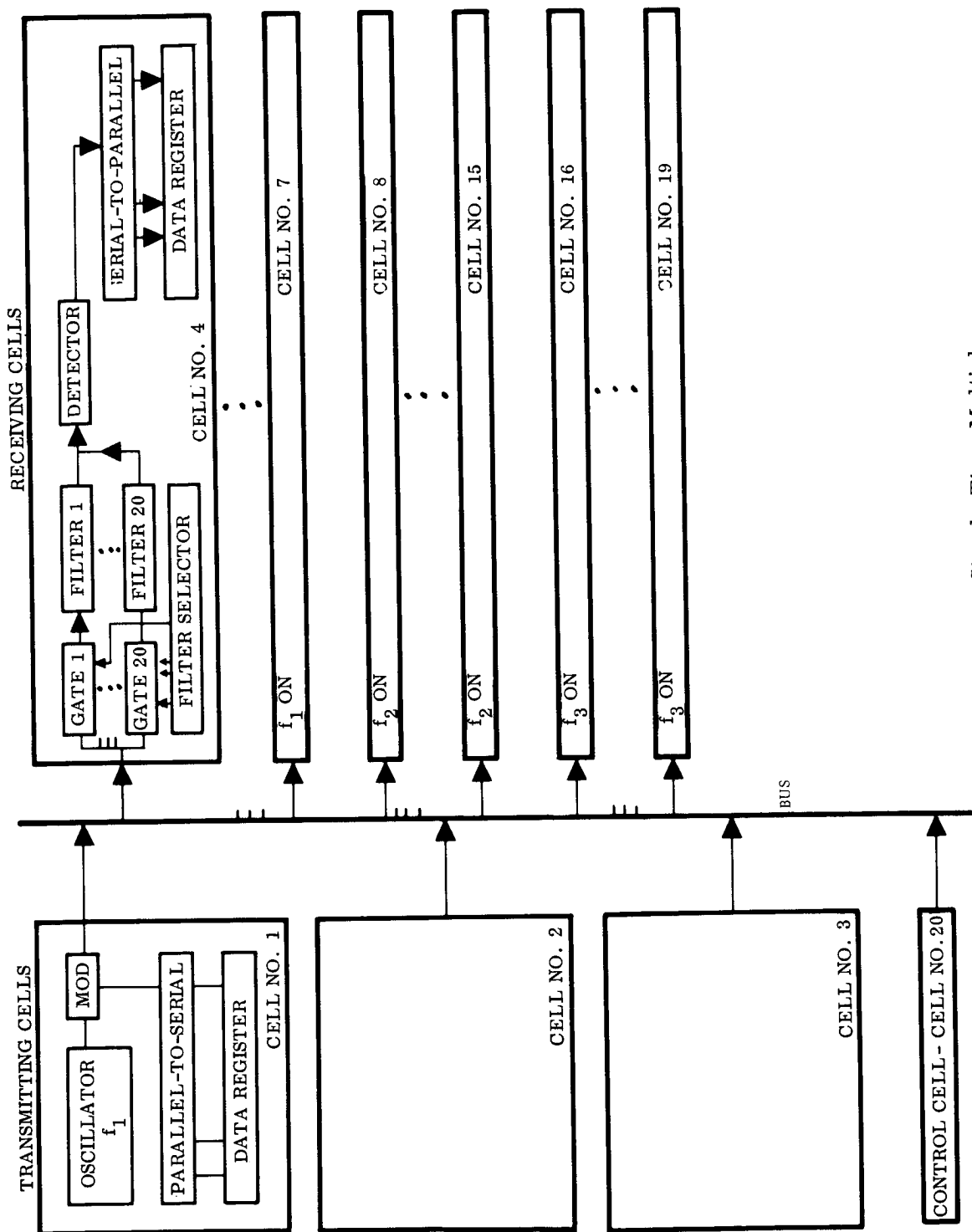


Figure D-10. Full Frequency-Simple Time Multiplex

transmitting on frequencies f_1 , f_2 , and f_3 . Each of these cells is received by several cells, shown at the right of the figure. Cell 1 is transmitting to cells 4 through 7. Similarly cells 8 through 15 are receiving data from cell 2 on frequency f_2 and their filters for f_2 will be turned on. Finally, cells 16 through 19 would be receiving from cell 3 on frequency f_3 and their filter for f_3 would be turned on. Cell 20 is the control cell. In this particular configuration three cells are then transmitting to the other cells in the group. It is clear that many other arrangements are possible with the maximum number of transmitting cells being ten, in which case there would be ten receiving cells. At the expense of additional RF hardware, one could have the cells simultaneously transmit and receive. There could then be a maximum of 20 transmitting and 20 receiving simultaneously. The frequency multiplex scheme which allows simultaneous transmission from 10 transmitting cells to 10 receiving cells is called "full frequency - simple time multiplex."

The tuning of the oscillators mentioned above can be achieved through the use of the Gunn effect. Devices based on this effect oscillate at a frequency which is dependent on an input voltage. If it is not possible to tune the oscillators to the precision required, then as many as 20 fixed frequency oscillators and 20 filters would be required on each chip.

It is simpler in terms of supervision to have each of the 20 cells have its individual frequencies as specified above. In this way, when a broadcast network is to be set up the only thing that must be done is to gate on the appropriate filter in each of the receiving cells.

To emphasize the total capability of full frequency multiplex, the example of Figure D-10 had all 20 cells in operation simultaneously. This is not necessary. Some of the cells could be in the process of computing and could be neither transmitting or receiving at a particular time.

D.4.2.5 RF System 4 - Full Frequency - Frequency Multiplex

The simple frequency multiplex scheme described in section D.4.2.3 can be expanded into a full frequency - frequency multiplex scheme. This can be done by assigning each of the 20 cells its own set of 18 carrier frequencies. These would be the frequencies of the oscillators on the chips. To set up a net, 18 filters corresponding to the oscillator frequencies of the transmitting cell would be connected from each of the receiving cells in the net to the common bus. This would require either tunable filters -- which do not appear feasible -- or a total of 360 filters on each chip in addition to its set of 18 oscillators. Clearly 360 filters plus 18 oscillators is too complex compared to the other schemes. It is possible to cut down the number of filters by using tunable oscillators. Even so, full frequency multiplexing between cells with the 18 bits of a word transmitted in parallel would require at least 10×18 or 180 frequency multiplex channels. This is a large number of channels in a bandwidth that is practicable for the projected state of the technology.

Expansion of frequency multiplexing in this fashion compounds the problems mentioned in section D.4.2.3. In addition the problem of how many channels can be multiplexed in a given bandwidth becomes critical and as pointed out in section D.4.1.1 may not be technically feasible.

D.4.2.6 Discussion of Systems

Various other combinations of time and frequency multiplexing can be considered for use in the distributed processor. However, the systems described can be viewed as bounding systems in that the first two are the simplest and the last two are the most complex that should be considered. The basic choice appears to be between simple time multiplex, simple frequency multiplex, and full frequency-simple time multiplex.

The system which has the most flexibility and highest capability of these three is the full frequency-simple time multiplex. This will be made clearer in section D.5 where the effects on the computer architecture are discussed. The basic difference between the first two systems and the last one is in the area of supervisory control. The first two time multiplex the cells and are quite similar in supervision to the present system which uses DC transmission. The last system can simplify certain aspects of supervision while making others quite complicated. Overall there may be an increase in supervisory complexity in going to the full frequency - simple time multiplex system. With regards to technology, the second system, namely the simple frequency multiplex system, would require the most complex mechanization. It also appears that simple frequency multiplex would put the biggest burden on the technology, i.e., interference between oscillators, variable loading due to filters, etc.

D.5 COMPUTER ORGANIZATIONAL CONSIDERATIONS

The previous discussion in this section has presented various RF communication systems. The last two systems discussed in section D.4.2, namely the Full Frequency - Simple Time Multiplex and the Full Frequency - Frequency Multiplex have the most significant affect on the computer organization or computer architecture. A discussion of the impact of a Full Frequency system will be given below. For purposes of this discussion the Full Frequency - Simple Time Multiplex System will be considered, however it is equally applicable to the Full Frequency - Frequency Multiplex System.

The Full Frequency system may be thought of as a full intercommunication system. There is a communication path between each and every group, and between each and every cell in a group.

The effects on the inter-group communications does not result in a great impact on the computer system design. Whether the groups can share one or several inter-group busses or whether the groups have a connection to every other group does not change the functional usage or the basic capabilities of the groups, other than an increased response time whenever communication is desired between groups. However, the result is significant where the inter-cell communication is concerned. The resultant computer system has greatly increased capability and can perform functions that are performed only with difficulty with the present DC time shared inter-cell bus system.

Presently the distributed computer system will be limited in capability primarily by its inter-cell communication system. In section 6 of this report, it was seen that the neighbor to neighbor communication system was justified, among several reasons, because of the difficulty in time sharing the inter-cell bus. The first step in improving

the system is to increase the capacity of the inter-cell bus. This means either increasing the transmission (clock) rate, adding more wires, or frequency multiplexing the bits. The clock rate reaches a limit based on the hardware capabilities, the extra connections required for more wires decreases the reliability. The RF schemes therefore appear particularly attractive in increasing the communication system capability. The construction of a cell using RF communications is shown in Figure D-11.

The first big advantage, over and above the increased communication capability available, is the reduced number of connections. There are no neighbor lines, no I/O lines etc. In fact, there are three basic lines required, DC power, ground, and a connection to the transmission line. A fourth connection might be needed for a cold (power off) start, but would have no other use. Each cell transmits on a particular frequency; the time-frequency spectrum of the communication system is shown in Figure D-12.

Now with regards to the effects of having a full intercommunication capability. No controller cell action is required to set up communications. In the present system, with only one cell bus shared by all, the controller cell was required to resolve conflicts and schedule the bus. In the new system, any cell can talk to any other cell. The only time a controller is required is when one cell wants to interrupt another cell that is already busy communicating with a third cell. I/O can go directly to the cell or cells in which it will be used, the controller cell is not needed. Several cells may receive the same data by having several cell's receiving on the same frequency.

Many dependent programs may take place simultaneously. The present computer system, because the dependent cell programs must go out on the cell bus, has only one controller cell that sends out dependent programs. The new system may have many cells send out instructions to many other cells, as shown in Figure D-13.

Cells C5, C8 and C19 are all tuned to the frequency used by cell 1, namely f1. Cell 20, transmitting on a different frequency, f20, can send a dependent cell program to C2 and C15 simultaneously and not conflict with cell C1. Thus the controller cell no longer has the problems concerned with the dependent cells. This may also reduce the need for a large number of groups. Groups would be required for increasing reliability and for a shortage of frequency allocation in one group.

Cells communicate when they are ready, and not when they are polled by the controller cell. This provides a great increase in capability in reconfiguration, and decreases the response time to failures and high priority interrupts. A cell may receive its inputs, do calculations, and send the outputs to another cell without requiring the controller cell. A faster response to an emergency condition is achieved. Reconfiguration is considerably simpler. There are no geometry restraints due to neighbor lines.

It is seen this system has greatly increased capabilities, and the software problems concerned with scheduling resources are simpler. The controller cell, which was concerned with trivial but time consuming problems of scheduling the intercell bus, has been transformed into a true executive. A true executive does not take part in the day-to-day operations, but initiates a program, gets it started, monitors the program's operation, and concludes the program when its priority

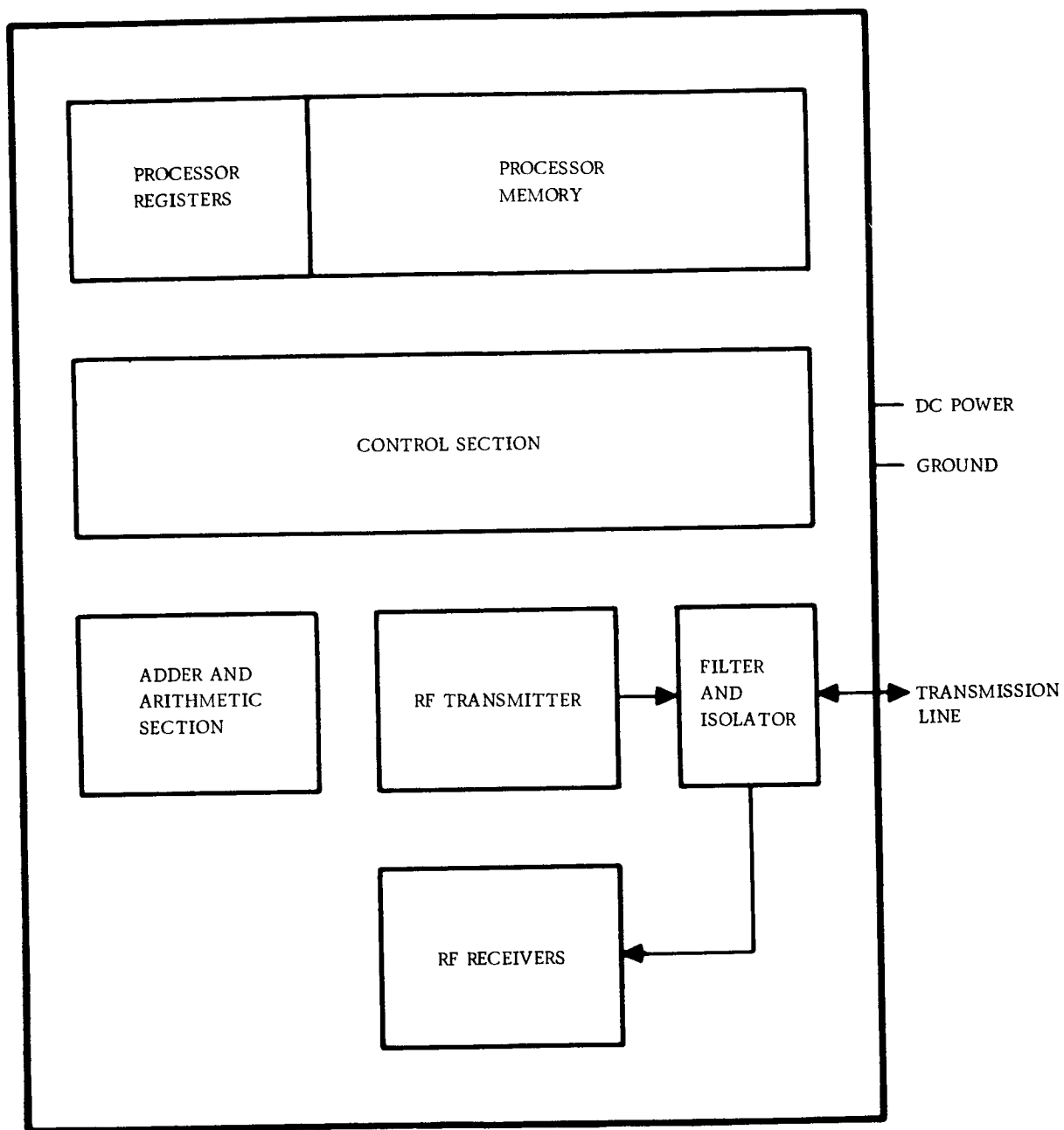


Figure D-11. Construction of New Cell

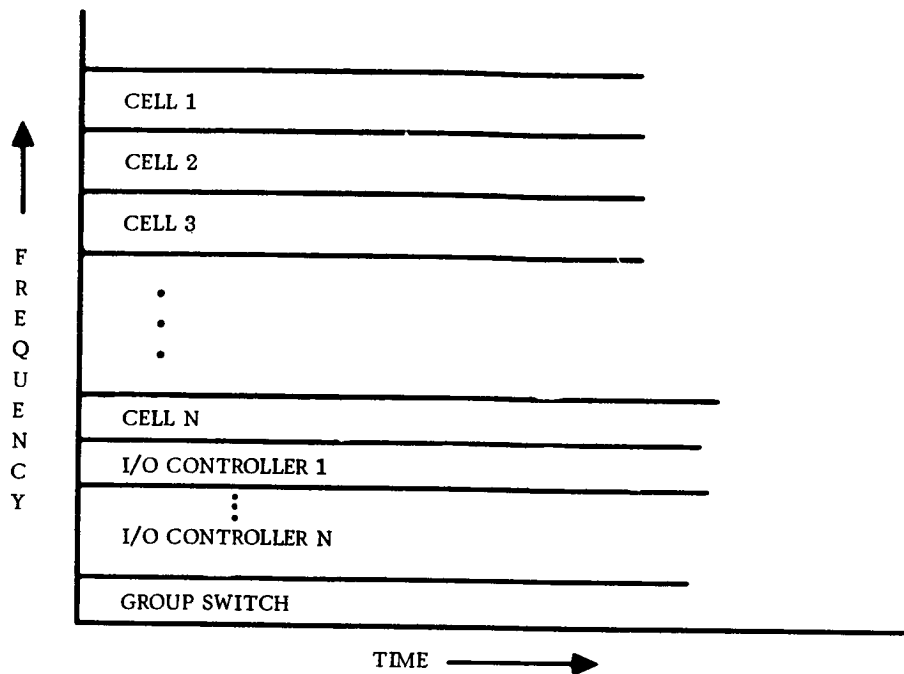


Figure D-12. Time Frequency Spectrum

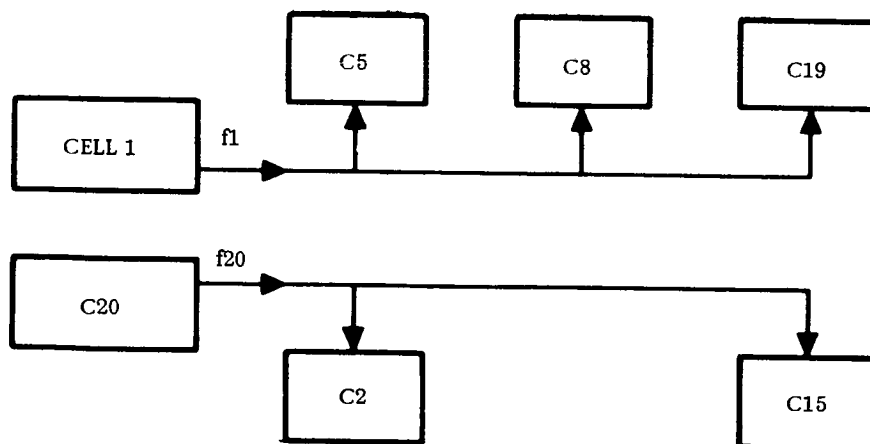


Figure D-13. Multiple Dependent Program Sources

becomes low, or its reason for existing has expired. These are the true executive functions. The input/output of normal data, instruction execution, normal output are all done without the controller cell having any essential part of the operation. The controller cell will only monitor the normal operations, thus, the controller cell may fail and the other cells may continue their normal operations with no changes. Changing the controller cell to an executive cell will increase reliability, the whole group will not necessarily fail if the controller cell fails.

The controller cell now becomes an executive cell with the following functions:

1. Monitor normal operations
2. Allocate system resources whenever
 - a. a conflict arises; e.g., two cells wish the same I/O controller,
 - b. a change in priority or mission occurs,
 - c. a failure occurs.
3. Monitor and communicate with the group that is the system executive.

This reduction in controller cell functions also reduces the storage imbalance between the controller cell and working cells. In the present system, the controller cell was expected to require considerably more storage than the other cells.

It should be noted that the I/O is not explained in great detail here. The amount of I/O, the frequency allocation, etc., will depend upon the capacity required. For example, the bulk storage unit may have several frequencies, as many as there are groups, so it may communicate to all the groups simultaneously.

Each I/O controller can have one or more frequencies allocated. Low data rate devices can be multiplexed into one controller. High data rate devices can use a unique frequency. The I/O controllers, since they do not normally communicate with each other but only communicate with cells, may be controlled by a cell to ensure there are not two I/O controllers on the same frequency. Note that this time sharing of a frequency allocation cannot be used with the cells, except in a degenerate case. The cells are all independent and may communicate with any other cell. Thus to force a time sharing of frequencies forces a special cell to allocate time and frequencies, and the system degenerates into the present system. Time sharing may be used only if some cells communicate with only one cell, the receiver cell may allocate frequencies and time. Now reconfiguration is more difficult.

The most capable cell is the cell that can transmit and receive simultaneously. This cell makes the best usage of the frequencies. A cell that may receive but not transmit simultaneously will always have one frequency not used; e.g., the transmit frequency. Thus half the frequencies may not be used.

A disadvantage of this system is the problem of providing a high priority interrupt. If the executive cell wishes to interrupt a cell that is busy communicating with another cell, special hardware on the receiver circuits may be required to let the cell know another cell wants to begin communicating with it.

D.6 CONCLUSION

This section has outlined various types of RF communications systems that could be applied to enhance the communication system of the distributed processor. It has been shown that these could have dramatic affects on the computer architecture. In addition, considerable research has been conducted in solid state RF circuits and in

the time frame of the distributed processor technology (1975) these devices should be capable of being incorporated into the cell wafer. Definitive comparisons of these systems both between themselves and with the present DC system cannot be made at this time. To do this the technology needs to be investigated further and the effects on computer operation, namely supervision, need to be studied in depth.

APPENDIX E. ERROR CONTROL IN THE COMMUNICATIONS SYSTEM

This appendix analyzes the error control problems associated with transmitting data from one cell to another within the multiprocessor. This analysis is particularly relevant to the RF Communication Systems. For high reliability of critical data, data that cannot be discarded, error correction is required. Error detection with retransmission of messages within which an error has been detected is the solution that will be considered here since it is assumed the system will have ample time for retransmissions. It will be assumed here that the S/N ratios at the receiving cell's input will be high enough that errors from this source can be neglected. The remaining sources of significance will be self-noise from the multiprocessor and crosstalk between the binary channels of the communications system. These can be reduced to very low levels through proper design.

Although the errors that do occur during transmission may be systematic, i. e., have patterns that occur more often than others, there is no way of determining this except through experiment with an actual system. Therefore in the derivations that follow it is assumed that errors occur independently and at random. It is also assumed that the probability of receiving a "1" when a "0" is transmitted is the same as for the opposite type of error. These assumptions are those which define the binary symmetric channel which is characterized by p , the probability of a bit being received in error. The communications system then is composed of a number of binary symmetric channels.

Messages which are transmitted from cell to cell are made up of 18 bit words. The number of words in a message could be fixed or it could vary from one transmission to the next. Therefore the formulas derived here will be for messages of N words.

The first step in deriving formulas expressing the performance of a retransmission error control system is to find P_w , the probability of an undetected error in an 18 bit word. Each 18 bit word contains one parity check bit checking all the bits in the word. P_w then is the probability of getting an even number of errors in the received word. Thus, letting $q = 1-p$:

$$\begin{aligned}
 P_w &= \sum_{\substack{i \text{ even} \\ i \neq 0}} \binom{18}{i} p^i q^{18-i} = \sum_{i \text{ even}} \binom{18}{i} p^i q^{18-i} - q^{18} \\
 &= 1/2 \sum_{i=0}^{18} (1 + (-1)^i) \binom{18}{i} p^i q^{18-i} - q^{18} \\
 &= 1/2 \left[(q + p)^{18} + (q - p)^{18} \right] - q^{18} \\
 &= 1/2 \left[1 + (1 - 2p)^{18} \right] - (1 - p)^{18}
 \end{aligned}$$

$$\begin{aligned} &\cong 1/2 \left[1 + (1 - \binom{18}{1} 2p + \binom{18}{2} 4p^2 \right] - \left[1 - \binom{18}{1} p + \binom{18}{2} p^2 \right] \\ &= 153p^2 \end{aligned}$$

This approximation holds when p is small with respect to $1/18$.

The next step is to derive P_N the probability of an undetected error in a message consisting of N words. This is:

$$P_N = 1 - (1 - P_w)^N \cong NP_w \quad \text{for } P_w \ll 1/N$$

This parameter is the probability of a message containing errors being accepted at a receiving cell as being correct.

When the transmitted data is not critical, it can be marked or discarded whenever an error is detected. When retransmission is used, there is the probability that an undetected error will occur in the retransmitted message. This can increase the expected probability of error, but the increase is still insignificant for the communications system. This follows from the following:

Let $PR_N(n)$ be the probability of an undetected error on the n^{th} transmission of a given N word message. That is, errors have been detected in $n - 1$ transmissions but not in the n^{th} . An error is detected in an 18 bit word only when an odd number of errors occur. Designate the probability of this event by Q_{DW} . Then:

$$\begin{aligned} Q_{DW} &= \sum_{i \text{ odd}} \binom{18}{i} p^i q^{18-i} = 1/2 \sum_{i=0}^{18} \binom{18}{i} (1 - (-1)^i) p^i q^{18-i} \\ &= 1/2 \left[(q + p)^{18} - (q - p)^{18} \right] = 1/2 \left[1 - (1 - 2p)^{18} \right] \\ &\cong 18p \end{aligned}$$

The probability, Q_{DN} , of detecting an error in an N word message is the probability of detecting errors in any one of the N words. Approximately:

$$Q_{DN} = 18Np$$

Then the probability, $PR_N(n)$, of detecting errors in $n - 1$ transmissions and not detecting errors on the n^{th} transmission is:

$$\begin{aligned} P_{RN}(n) &= (18Np)^{n-1} NP_w \\ &= (18Np)^{n-1} N153p^2 \end{aligned}$$

The expectation of $P_{RN}(n)$ over all values of n is:

$$P_{RN} = \sum_{n=1}^{\infty} (18Np)^{n-1} N153p^2 = N153p^2 \sum_{n=0}^{\infty} (18Np)^n$$

$$= \frac{N153p^2}{1 - 18Np} \cong N153p^2 \text{ for } 18Np \ll 1$$

The expectation of the transmission time lost due to retransmissions is an important parameter. This is:

$$\sum_{n=1}^{\infty} P_{RN}(n) T_n$$

where T_n is the time required for a n transmission. Substituting for $P_{RN}(n)$ from the above, and setting $T_n = nT$: where T is the time required for a single transmission:

$$E(T_n) = \sum_{n=1}^{\infty} P_{RN}(n) T_n$$

$$= \sum_{n=1}^{\infty} (18Np)^{n-1} N153p^2 n T$$

$$= N153p^2 T \sum_{n=0}^{\infty} n (18Np)^n$$

$$= N153p^2 T \frac{1}{(1 - 18Np)^2} \cong N153p^2 T \quad \text{for } 18Np \ll 1$$

It can be concluded from the above formulas that there will seldom be a retransmission, and almost never more than one retransmission. The probability of undetected errors is extremely small since p is small relative to the other pertinent system parameters.

APPENDIX F. ALTERNATIVE DISTRIBUTED ORGANIZATIONS

There are a number of alternative distributed logic systems to the chosen DAMP System described here. Three such systems, the Holland machine, Solomon machine, and Illiac were discussed earlier. These structures were shown to have interesting features but they were not practical for a spaceborne computer designed for high reliability on general purpose computations. Alternative forms of the DAMP System could use the same basic cell structure but different quantities of memory or different communication links. The addition of more memory to each cell in the DAMP System has been mentioned earlier. This would be necessary if the cells were found to be memory limited. The addition of memory could be carried out by adding to each single wafer cell additional wafers containing 512 to 1024 words of storage and no processing. This would of course increase the number of wafer types to two along with increasing the number of interconnections in the system. A system composed of the new larger cells (two or more wafers) would now contain a significantly smaller number of total cells.

Another consideration for a distributed processing system is to take the suggested DAMP System and decentralize it such that each set of sensors (e.g. the navigation sensors or the telecommunication sensors) have their own group of cells located physically with the sensors. In fact the group of cells could be packaged on the same board as one of the sensors in each set. The group of cells would handle all processing and control associated with the set of sensors. However there would still be a need for a central communication group to handle the transmission of information between sensors and to provide some executive functions. For example, results of scientific experiments would have to be sent to the telecommunications group. The resulting decentralized DAMP system is shown in Figure F-1. It should be realized that the distances from the sensors and groups of cells to the communication units may be quite long; as a result the communication rates can not be high. However, this later restriction should not be too severe if the sensors are grouped so that only a small amount of intercommunication between groups is necessary.

This decentralized system could be considered to try to achieve greater reliability from the system hardware. Locating the groups of cells physically close to the sensors cuts down the probability of failure of the communication lines between the processors and the sensors. It also decreases the power dissipation of the system and somewhat simplifies the real time control task of the processor. The latter improvement is realized due to the lack of long delays in receiving information from the sensors. However, the advantages of this organization may be outweighed by its disadvantages in relation to the chosen DAMP system. The need for central communication groups requires long communication lines from the sensors to a central point and as a result essentially cancels the reliability gained by decentralizing the computer. This can be understood by realizing that the central DAMP system used conditioners located near the sensors to minimize the communication lines to the central computer. However, not only may the decentralized structure not require conditioners, but also some of the groups could operate autonomously of the central executive once loaded from the bulk storage unit. Another possible disadvantage is the fact that each group in the decentralized system would require at least some amount of its own executive and common data storage areas. As a result, some increase in total equipment would be necessary. In addition, two central executive groups should be used to guarantee an available executive group. Of even more

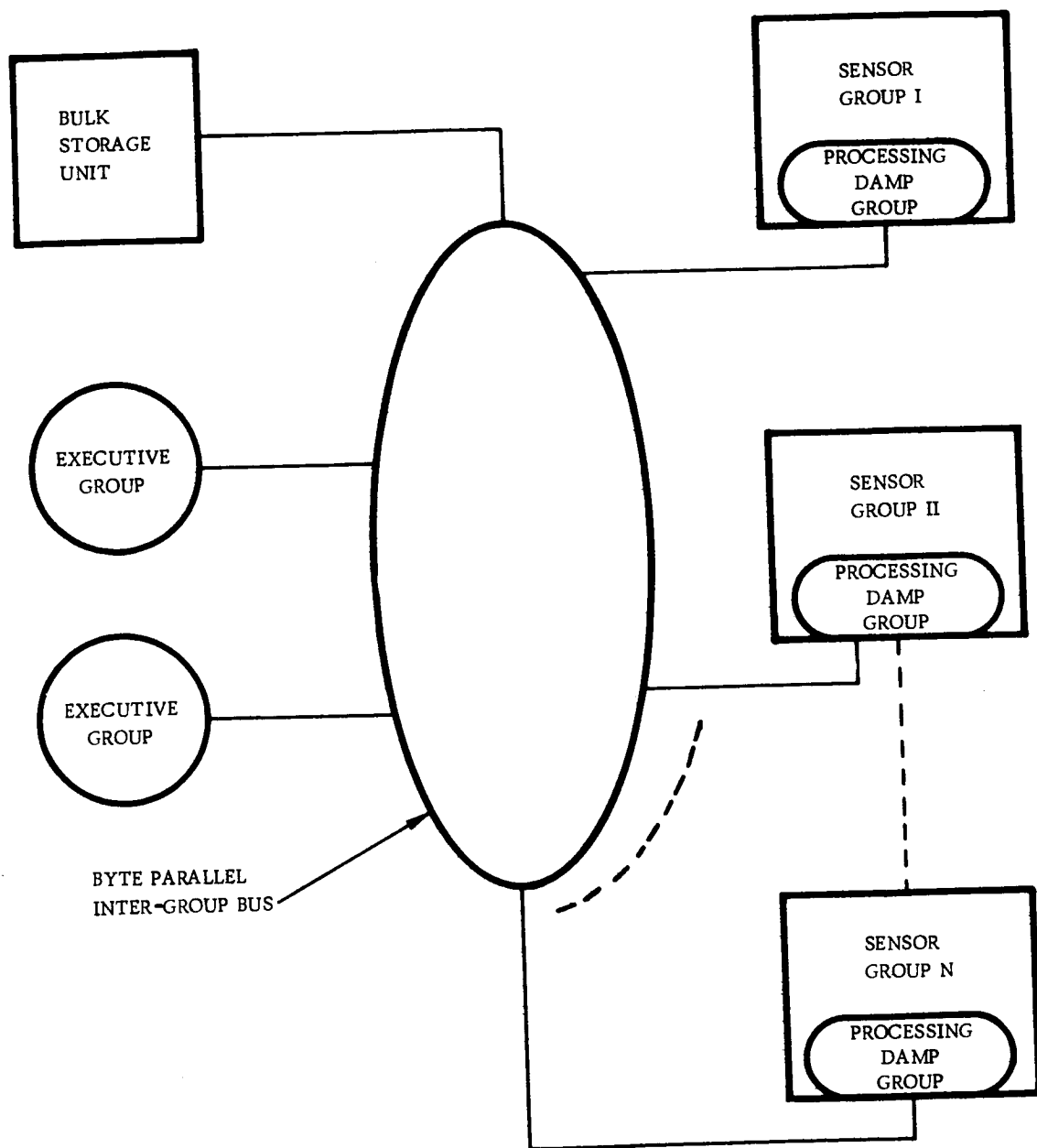


Figure F-1. Decentralized DAMP System

importance than these points is the fact that there would no longer be a relatively large and powerful central computing facility to carry out the background programs. The distributing of these programs may be somewhat difficult and inefficient due to the long communication delays between parts of a large program that could be located in two separate groups. It is also clear that the system is not as flexible as the chosen DAMP system and as a result its ability to gracefully degrade is somewhat limited. (The groups now have explicit functions rather than being available to carry out any combination of tasks). However, this latter point should not be as severe of a limitation as the earlier points since the reliability of a group of cells with a few spares connected on the bus should be considerably greater than the reliability of a complicated sensor.

The preceding discussion shows the difficulty in choosing a decentralized or centralized DAMP System as the best structure for the given requirements. A considerable amount of future design and development of the sensors will be necessary before a well founded decision can be made. In the meantime, for this study the centralized system was investigated because it was felt that this structure will be more natural for development of software and investigation of programming and system features. In addition, both have much in common so that the vast majority of the developmental work of the centralized structure will be equally applicable to the decentralized structure.

REFERENCES

1. Study of Spaceborne Multiprocessing Final Report, North American Aviation/Autonetics Division, April 1967, C6-1476.10/33.
2. Study of Spaceborne Multiprocessing, North American Aviation/Autonetics Division, First Quarterly Report - Phase 1, June 1966, C6-1476.1/33.
3. Manned Mars Surface Operations, AVCO/RAD, RAD-TR-65-26, Sept. 30, 1965.
4. Mission Requirements for Manned Mars and Venus Exploration, Vol. III, General Dynamics/Forth Worth, FZM-4366-3, May 30, 1965.
5. Renzetti, et. al., Projected NASA/JPL Deep Space Network Capabilities in the 1970's, AIAA Stepping Stones to Mars Meeting Baltimore, Maryland, March 1966.
6. Manned Interplanetary Spacecraft Systems, North American Aviation/Space Division, SID 65-1278.
7. Study of Subsystems Required for a Mars Mission Module. North American Aviation/Space Division, SID 64-1-1 and SID 64-1-4. January 1964.
8. Personal communication with North American Aviation/Space Division.
9. Warner, R. M., Jr., Comparing MOS and Bipolar Integrated Circuits, IEEE Spectrum, June 1967, pp 50-58.
10. Petritz, R. L., Technological Foundations and Future Directions of Large Scale Integrated Electronics, Proceedings of the Fall Joint Computer Conference, 1966.
11. Gange, R. A., Taking Cryoelectric Memories Out of Cold Storage, Electronics, April 17, 1967.
12. Stewart, R. D., PSIN Arrays - The Complete Optoelectronic Function, Proceedings of Conference on Integrated Circuits, 2 - 4 May 1967.
13. Manasevit, H. M. and Simpson, W. I., Late News Item, Proc. Am. Phys. Soc., Edmonton, Alberta, Canada, 1963.
14. Allison, J. F., Meiman, F. P., and Burns, J. R., Silicon on Sapphire Complementary MOS Memory Systems, Proc. ISSCC, Phila., Penn., February 15 - 17, 1967.
15. Allen, Charles, A., Design of Digital Memories That Tolerate All Classes of Defects, Stanford Electronics Laboratory, May 1966. SU-SEL-66-031.
16. Handbook for System Application of Redundancy, Naval Applied Sciences Lab, 30 August 1966.

17. Burns, J. R., Switching Response of Complementary Symmetry MOS Transistor Logic Circuits, RCA Review 25, 1964.
18. Martin, D. F., The Automatic Assignment and Sequencing of Computations on Parallel Processor Systems, University of California, Los Angeles, California, AD628220, January, 1966.
19. HU, T. C., Parallel Sequencing and Assembly Line Problems, Operations Research, 9:841-848, Nov.-Dec., 1961.
20. Held, M., and Karp, R., A Dynamic Programming Approach to Sequencing Problems, J.S.I.A.M., 10:196-210, March 1962.
21. Schwartz, E. S., An Automatic Sequencing Procedure with Applications to Parallel Programming, J.A.C.M., 8:513-537, Oct. 1961.
22. Garner, H. L. et. al., A Study of Iterative Circuit Computers, University of Michigan, report #AL-TDR-64-24, April, 1964.
23. Hollander, Gerhard L., Architecture for Large Computer Systems, Spring Joint Computer Conference Proceedings, April 1967, pages 463-466.
24. Minnick, R.C. et. al., Cellular Arrays for Logic and Storage, Stanford Research Institute, April 1966, AD643178.
25. Minnick, R.C., Survey of Microcellular Research, Stanford Research Institute, July 1966, AD637010.
26. Holland, John, A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously, Proceedings of the Eastern Joint Computer Conference, 1959, pages 108-113.
27. Slotnick, D. L., et. al., The Solomon Computer, Proceedings of the Fall Joint Computer Conference, 1962, pages 97-107.
28. Slotnick, D. L., Unconventional Systems, Spring Joint Computer Conference, April 1967.
29. Results of Multi-Accumulator Study for Next Generation Computer; Internal Report; Autonetics, Anaheim, California.
30. Computer Memory Banking Study; Internal Report; Autonetics, Anaheim, California.
31. Volden, Jack E., "The Cordic Computing Technique". Proceedings of the Western Joint Computer Conference, March 1959.
32. Hartig, David, "Microelectronic Digital Stabilization Computer", Bureau of Naval Weapons Symposium for Rotating and Static Components, April 22, 1964.
33. Study of Spaceborne Multiprocessing, third quarterly report - Phase 1, C6-1476.8/33; Autonetics, Anaheim, California.

34. Survey of Highly Parallel Information Processing Technology and Systems, Phase I of an Implications Study, by John C. Murtha and Robert L. Beadles, Westinghouse Defense and Space Center, Baltimore, Md. November 1964 AD480150.
35. Detection of Essential Ordering Implicit In Compiler Language Programs, by Harvey W. Bingham, et. al., Report ECOM-02463-2. Burroughs Corp., Paoli, Pennsylvania. AD650845 February 1967.
36. A General Approach to Parallel Operation in a Multiprocessor Environment, by Samuel D. Epstein, Auerbach Corp. RADC-TR-67-83 AD812274 March 1967.
37. Parzen, E., Modern Probability Theory and Its Applications, Chapter 5, Wiley & Sons, 1960.
38. Electronics, pages 105 - 136, November 13, 1967, Volume 40 Number 23

BIBLIOGRAPHY

1. Heller, J., Sequencing Aspects of Multiprogramming, J.A.C.M., 8:426-439, July, 1961.
2. Klein, M., On Assembly Line Balancing, Operations Research, 11:274-281, March-April, 1963.
3. Kilbridge, Webster, A Review of Analytical Systems of Line Balancing, Operations Research, 10:626-638, Sept.-Oct., 1962.