SAMSO TR-67-106

PHASE II OF AN ARCHITECTURAL STUDY FOR A SELF-REPAIRING COMPUTER

J. P. Roth, W. G. Bouricius, W. C. Carter, and P. R. Schneider

International Business Machines Corporation
Thomas J. Watson Research Center

NOVEMBER 1967

U. S. AIR FORCE
SPACE AND MISSILE SYSTEMS ORGANIZATION
AIR FORCE SYSTEMS COMMAND
LOS ANGELES, CALIFORNIA

447

SAMSO TR-67-106

PHASE II OF AN ARCHITECTURAL STUDY FOR A SELF-REPAIRING COMPUTER

J. P. Roth, W. G. Bouricius, W. C. Carter, and P. R. Schneider

International Business Machines Corporation
Thomas J. Watson Research Center

NOVEMBER 1967

U. S. AIR FORCE
SPACE AND MISSILE SYSTEMS ORGANIZATION
AIR FORCE SYSTEMS COMMAND
LOS ANGELES, CALIFORNIA

FOREWORD

This technical report summarizes Phase II efforts of an Architectural Study for a Self-Repairing Computer performed by the Research Division of the IBM Corporation under U. S. Air Force Contract Number AF 04 (695) - 1056 for the period June 1966 to August 1967. Phase I (SSD-TR-65-159) was performed by the Federal Systems Division of the IBM Corporation under U.S. Air Force Contract Number AF 04 (695) - 769 in 1965.

This work was begun under the supervision of the Space Systems Division with Major Emmet G. DeAvies as Air Force Project Officer and completed under the supervision of the Space and Missile Systems Organization with Capt. Ronald J. Starbuck as Air Force Project Officer and members of the Aerospace Corporation giving technical direction.

Information in this report is embargoed under the Department of State ITIARs. This report may be released to foreign governments or agencies of the U.S. Government subject to approval of SAMSO (SMTRE) or higher authority within the Air Force. Private individuals or firms require a Department of State export license.

U. S. Air Force and IBM Research personnel involved were:

a. Air Force Project Officer

Ronald J. Starbuck
Capt. USAF

b. IBM Technical Direction

W. G. Bouricius

c. IBM Study Contract Personnel

J. P. Roth (Principal Investigator)

W. G. Bouricius

W. C. Carter

P. R. Schneider

REVIEW AND APPROVAL

This technical report has been reviewed and is approved

[illegible]

RONALD J. STARBUCK

Capt. USAF
Space and Missile Systems Organization
Air Force Systems Command

ABSTRACT

The architecture and organization of a self-repairing computer which operates correctly even if some of its components malfunction is described. This computer, using currently available components, has a 0.997 probability of correct operation on a 10,000 hour mission with a 25% duty cycle. First the problem is stated, an overview of previous work is given, and the results of the investigation are summarized. Then several current computer component technologies are appraised and the assumption in reliability models of a Poisson failure distribution for these components is justified. Next, the development of several algorithms for the generation and evaluation of diagnostic test patterns for computer circuits is described. These algorithms are illustrated by examples and by detailed APL programs. Next, mathematical reliability evaluation models for a variety of functional computer units are devised, their APL programming implementation given, and their use in the design process illustrated. The effects of failure coverage, duty cycle, number of spares, failure tolerance, lower component failure rate with power off, and other parameters are tabulated and delineated. Last, the architecture and operation of an automatically repaired computer at different levels of organization is described, including failure tolerant storage, ROS, and arithmetic logical organs. The design of reconfiguration switches, status registers, and a fully checked decoder is also given. By employing these techniques, which require about 2.5 times as much hardware as an equivalent simplex computer, it is possible to gain the same benefits as would be produced by a 600 fold decrease in component failure rates.

BLANK PAGE

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

BLANK PAGE

1 Background and Summary

1.1 Statement of Problem Space satellite applications require computers with extremely high reliability and availability, so high in fact that they cannot be readily achieved with any of the current computer configurations using extant or near-future technologies. Further, these applications require that the computer operate with no human or mechanical repair facilities, i.e. be "self-repairing". Typical guide-line specifications used in this report call for a medium sized computer (CPU, ALU, core storage, and channels) with approximately the computing power of a 360/model 44 which is to perform with a reliability and availability of .997 for a 10,000 hour mission.

Present simplex computers of this capacity have mission times on the order of 5-10 hours when this reliability specification is used. Breaking the computer up into 15 units and using triple modular redundancy (TMR) on each unit can only increase the effective mission time to about 200 hours. Thus a more imaginative computer architecture is required to meet the problem of achieving this 1000 fold increase in mission time over that of the simplex machine. Such an architecture is developed in this report for a computer to be called ARC - Automatically Repaired Computer.

Early in the development it was apparent that there existed a definite lack of adequate tools required to implement a truly ultra-reliable computer architecture, e.g. knowledge of failure mechanisms, precise diagnostic algorithms, computer oriented reliability evaluation models, techniques for designing effective self-repair mechanisms, etc., were all unavailable. This necessitated the preliminary solution of these problems before the actual computer architecture could be specified with any degree of confidence in its efficacy.

1.2.1 The Early Years The first digital computers made extensive use of relays and tubes, devices which were noted for a general lack of reliability. Thus, during the early development of computers a large effort was expended in the area of computer checking and self-repair.

In relay computers intermittent faults far outnumbered permanent faults [ERA, 50]. For this reason, extensive dynamic checking (biquinary code and proper finish of instruction) was used. The success of Bell Relay Computers (BRC) is evidenced by the following quotation, "Starting with the Model III delivered to the Armed Forces in 1944, not one of our customers has reported their computers giving out a wrong answer as the result of a machine error" [Harvard, 49]. In addition, replacement procedures were available. The BRC Model V at Aberdeen Proving Grounds had two central processors. When one had to stop a problem because of an error, the other would read in the next problem from paper tape and begin its execution [ERA, 50]. The Model VI had an improvement – a "second try" feature for retrying an operation immediately if it failed to receive an OK signal at the end of the operation [Harvard, 49].

In the area of electronic computers, Raydac [Harvard, 49] had 1) a modulo 31 arithmetic check which was used for data transfers and all arithmetic operations, 2) checking for illegal op-codes, and 3) checks to see that the correct address was given the memory and that a valid word came from that address. R. M. Block [Harvard, 49] explicitly stated the necessity for checking computer controls as well as the ALU. The EDVAC (1949) seems to be the first computer to use two arithmetic units and a comparison of their results; as long as they agreed, processing continued. The IAC [Weik, 55] computer started the hardware design of illegal result tests, such as the divide check, and implemented a special instruction to take a check sum of part of memory. This latter instruction was designed to be used to check I/O by an examination of all data blocks as read in.

The set of computers designed just a little later had the checking features shown in the following table [Weik, 55]:

Computer	Data Transfer Checking	Arithmetic Checking
UNIVAC	parity.	duplicate ALU.
	I/O block character count and automatic reread.	
Whirlwind I	parity.	some ALU checks.
IBM 650	character validity.	
	horizontal and vertical parity on tape.	
NORC	modulo 4 check	modulo 9 check

1.2.2 The Middle Years The advent of the transistor (possessing a higher inherent reliability than relays or tubes) and its mass influx into the vast commercial computer market of the middle-to-late '50's brought about a marked deemphasis in computer error checking and self-repair. At this time error detection and/or correction centered primarily on the I/O equipment, special purpose computers, and special technologies.

One of the first computers to be designed with truly high availability goals in mind was IBM's AN/FSQ7 (SAGE) which was used for on-line real-time data processing by the AIR FORCE [EZB, 57]. This system featured two complete tube-type computers, one operating on-line and the second acting as a backup. The memory of the backup computer was periodically updated so as to contain the same data as the on-line machine. Each computer had an elaborate failure detection scheme employing parity checks and conventional software diagnostic programs. On detection of an error, the switchover and recovery was executed by software. Later versions, the AN/FSQ 31 and AN/FSQ 32 [Weik, 61], used transistorized circuitry and more inclusive checking including the FIX concept of correcting single errors via software.

Remington Rand's Titan guidance computer, the Athena [RR, 58], obtained some measure of high availability by carefully controlled construction and selection of its components. These were then assembled in a white room. While a gain in reliability was achieved, the cost-effectiveness was not very high.

In the large commercial computer area three machines stand out. Remington Rand's LARC [Weik, 57] featured Hamming single error correction codes for data transfers and storage, and checked the arithmetic operations by comparing the output of duplicate ALU's. The STRETCH computer, IBM's 7030 [Buchholz, 62], used parity, duplication, and residue checks (casting out 3's) on all arithmetic operations. The memories and data transfers used a Hamming single error correction, double error detection code. When uncorrectable errors occurred, the machine was interrupted and an extensive diagnostic "log-out" of all pertinent data performed. The SABRE system [PP, 61], a natural outgrowth of the SAGE concepts, features a pair of IBM 7090's in on-line - standby roles supporting a vast network of airline terminals.

Concurrent with the design of these actual computers, effort was being made to extend the theoretical models upon which such machines could be based. In the area of masking redundancy, von Neumann [von Neumann, 56] developed and analyzed a highly reliable scheme employing triplication of all units with majority voting at selected interfaces; this is the so-called Triple Modular Redundancy or TMR. Replication of circuit components, introduced in [MS, 56] for relays, culminated in the quadded logic method of design [Tryon, 62]. Tryon's somewhat more sophisticated approach to masking redundancy suffered from a lack of a really smooth way to handle memory devices; it resorted to TMR. Models for standby or replacement redundancy techniques were also proposed [Creveling, 56 ; Fein, 57] and analyzed in great detail [Flehinger, 58]. Later a method for actually effecting the replacement of the failed unit by its standby spare was precisely formulated in [GMRW, 60 ; GMR, 62] for a cryogenic technology.

This report also evaluated the effect of switch failures on the overall system reliability, showing that as the number of switchable interfaces increases (correspondingly, the size of the replaceable unit decreases) the switch failures tend to dominate the reliability.

Several algorithms for the diagnostic testing of computer circuits also made their appearance in this era [Eldred, 59; Moore, 56]

<u>1.2.3 The Present</u> Recently a strong surge of support for self-repairing, highly available computers has been felt. The major users interested in this effort are the on-line or time-sharing, transaction-oriented, commercial computing facilities and the long term space missions. In the former, the concern is over maintenance of continuous customer servicing acknowledging that failures will occur in the ever more complex computers required to keep pace with the transaction demand. In the latter, the need is for self-repairing computers capable of maintaining some minimum level of computing power with a very high probability over an extremely long mission time.

In the commercial line the present approach taken to achieve high availability is that of multi-processor configurations employing graceful degradation as in Bell's Electronic Switching System [ESS, 64], IBM's 9020 [Wood, 65] and Burrough's B-8500 [Gluck, 65]. Such machines are typically organized with several memory units, CPU's and I/O channels accessed off a common data buss. When the checking and/or periodic diagnostic testing determines that a device has failed, that device, together with a maintenance subsystem, is reconfigured out of the main computer with a subsequent performance degradation. After the failure is located (probably by use of diagnostic tests) and fixed, the subsystem is brought back into the main computation system [DS, 66].

A very good example of how diagnostic tests can be used to automatically locate circuit failures is the Fault Location Technology (FLT) employed in SYSTEM 360. Special "SCAN" lines were added to the circuitry to permit data to be entered into and read out of most of the circuit's latches in order to facilitate the use of combinational circuit diagnosis algorithms (cf. section 3.1.2 for a review of such algorithms). Precomputed diagnostic tests, stored on a magnetic tape, are then used to rapidly detect and locate all circuit failures of a prespecified type. Use of the FLT concept greatly reduces the down-time (hence increases the availability) of a simplex computer or a maintence subsystem of a multiprocessor.

Recent approaches to the high availability problem in military computers have been varied. IBM's Saturn V program [SCF, ., 63] segmented the computer into seven functional modules which were then implemented in TMR. The memory was duplexed with parity checking used to isolate the failed unit. Extensions of this work [AES-EPO; 65] proposed and analyzed a full TMR computer for use on a 90 day mission. RCA's variable instruction computer (VIC) [Miller, 66] possesses an architecture which provides duplicate control stores and at least two alternate methods for performing any operation. When the checking indicates a failure in a particular portion of the computer it automatically "degrades" to the alternate algorithm for performing the tasks which formerly used the failed portion.

A recent, advanced ultra-reliable computer is the JPL Self-Testing and Repairing (JPL - STAR) computer [Avizienis, 66]. The JPL-STAR uses complete hardware implemented error detection, rather than periodic diagnostic testing, since transient failure protection was deemed critical. Arithmetic operations are checked modulo 15. The control memory uses a four bit check in the 32 bit data word to verify that the 14 bit address has correctly accessed the proper word. The main read-write memories were duplexed. No adequate facilities have been selected for removing the Central Control Unit (CCU) from the "hard core" — hard core is defined to be that portion of the computer in which a single failure causes system failure.

Another recent effort in the field of self-repairing, self-diagnosing computers was initially reported in Phase I of this contract [ARSY, 65] and later published in [FRST, 65 ; AFS, 67]. An approach to the formalization of the theory of self-diagnosing digital systems was made; theorems specifically stating necessary and sufficient conditions for self-diagnosibility were given. The theorems require that the digital system be partitioned into connected sets of diagnostic subsystems each capable of diagnosing its neighbors. The results of the theory were then used to propose the architecture of a computer, designated the DX-1, which has minimal hard-core circuitry. One interesting procedure which was discussed was the possibility of forming the diagnostic subsystems by dividing the machine up into three subsystems right through the middle of the data paths, i.e. a 36 bit data word machine used for actual computation would be segmented into three distinct 12 bit data word machines for diagnosis. When one failed, it could be diagnosed by the other two and a degraded mode of operation with 24 bit words initiated.

Omitted from the Phase I report was a sufficiently detailed examination of the design for the interface circuitry interconnecting the diagnostic subsystems and its effect on the required independence of the diagnostic subsystems. Also omitted was the work necessary to precisely evaluate the reliability of self-repairing computer architectures for use in studying the effects of parameter trade-offs on this reliability.

The work described in the remainder of this report is a direct offshoot of the Phase I efforts. It was undertaken with the intent of extending the existing work on computer diagnosis, models for computer reliability evaluation, and self-diagnosing, self-repairing computer architectures and organizations. As stated in the original proposal [RBCF, 66],

"The ultimate objective of this proposal is the design and construction of a truly ultra-available computer".

1. 3 Summary of Current Report

1. 3. 1 Background The first section of each chapter contains a comprehensive discussion of the need for the work covered therein, a brief review of previous efforts in the area, and a summary of the chapter. Hence this section, 1. 3, will be limited to a general overview.

1. 3. 2 Technology The mechanisms of circuit failure in present technologies were investigated in order to 1) establish the class of failures for which diagnostic tests should be established and 2) obtain a probability distribution for the failures together with the appropriate failure rates. The normal stuck-at-1 , stuck-at-0 and, possibly, short circuit failures are still valid diagnosis models. The failure mechanism is such that after a short burn-in period the failures can be characterized by a Poisson distribution, i.e. $R = e^{-\lambda T}$ with λ the constant failure rate.

Large Scale Integration (LSI) was examined with the conclusion that it has definite power-weight advantages but has an uncertain failure rate and delivery for mass manufacturing. Further, the large circuit size and inherent pin limitation of LSI implies that cyclic circuits must be diagnosed without resorting to extra inputs for cutting feedback loops, i.e. implies true cyclic circuit diagnosis algorithms be developed.

1. 3. 3 Diagnosis Algorithms The D-calculus, due to Roth, was used to develop several diagnosis algorithms necessary to achieve the extremely efficient failure coverage required in any ultra-reliable computer. The algorithms are:

1) D-Algorithm (DALG-II) guarantees finding a test.

if one exists, for any failure, of a specified class, in an acyclic or combinational circuit.

2) TESTDETECT ascertains all acyclic circuit failures, of a specified class, which a given test will detect.

3) CD-Algorithm computes tests for failures in a certain well defined class of cyclic or sequential circuits.

Both DALG-II and TESTDETECT were programmed on the APL time-sharing system.

Allied with the development of these three algorithms was the formulation of 1) a multiple-failure variation of the D-algorithm for use in finding a single test to detect one of several failures, 2) a Test-Response variation of TESTDETECT which not only tells if a failure is detected by a given test but also specifies the precise output response when the failure occurs and 3) a synchronous circuit model suitable for facilitating cyclic circuit diagnosis.

1.3.4 Reliability and Evaluation In order to determine the relationships among the parameters of the Automatically Repaired Computer — ARC, mathematical reliability evaluation models had to be devised which contained specific and valid representations of these parameters. Typical quantities considered are listed below:

λ	-	failure rate during power-on
μ	-	failure rate during power-off
m	-	number of spares initially available
q	-	number of modules required to be operating at all times (power-on)
f	-	number of failures each unit can withstand before malfunctioning.
T	-	mission time
c	-	failure coverage, or probability, that given a failure occurs, it is detected by the checking

and/or diagnostic tests and that successful recovery occurs.

The parameters which effect the form of the reliability equations are m, q, c, and f. In order to display the status of these parameters each reliability equation is written as ${}^{f}_{c}R^{q}_{m}$. A new recursive formulation of an integral equation to express reliability of a system having spare modules with turned off power was devised. Closed form solutions were obtained for all cases where $f = 0$. When $f \neq 0$ closed solutions could only be obtained for certain small values of f and q; in the general cases involving $f > 1$ approximations and bounds were obtained.

Several interesting results of some of the parameter studies are tabulated below.

1) It is proved that it is always worthwhile for reliability (as well as power consumption) to turn unused spares off.
2) It is shown that if the error coverage decreases from 1 to .9 then the length of survival time is decreased by a factor of three.
3) The reliability ${}_{c}R_{2}$ and the reliability of TMR'd computers were compared. It was proved that for $\lambda T \leq .1$ and coverage equal 1, using two spares gave greater reliability than TMR as long as the extra switching required for two spares was less than 248.3% of the total original equipment. If $c \geq .9$, then for $\lambda T \geq .02$ ${}_{c}R_{2}$ is greater and for $\lambda T < .02$ the maximum difference of reliability values is 8×10^{-5}.

A program, REL, using these models for ${}^{f}_{c}R^{q}_{m}$, with explicit entrance of parameters, has been written in APL and run on on-line consoles by the system designers. REL was used extensively in evolving the architecture ARC as discussed in Chapter 5 .

<u>1. 3. 5 Architecture/Organization</u> The design of ARC started with dividing a standard computer into functional units to facilitate hardware controlled retry within the unit. Standby redundancy and segmentation were then assumed for each unit and the program REL used to determine the optimum ARC I configuration. Then a new version, ARC II was obtained by redesigning memories and arithmetic units which could tolerate one or more errors. The following table shows the various design configurations considered.

Configuration	Mission Time with .997 Probability	Equipment	Mission Time / Equipment
Simplex	6. 59	1	6. 59
TMR	271	3	90
ARCI	9, 068	3. 5	2, 590
ARCII	10, 879	2. 5	4, 350

Detailed implementations of the following self-repair mechanisms are effected:

1) <u>Reconfiguration Switches</u> Reconfiguration switches to effect standby redundancy self-repair techniques were designed with the property that a failure in the switch caused one of the modules being used to be unavailable without affecting the others. The status registers canbe designed to be error correcting using masking redundancy.

2) <u>Error Tolerant Memory Unit</u> Using the same type of reconfiguration switching it is easy to design a reconfigurable bit plane memory unit. A new method of dynamic storage address blocking achieved error toleration in the addressing circuitry.

3) Ultra Reliable ROS Configurations   The same techniques whicn made a memory module more reliable are used in a simplified form to make a ROS error tolerant.

4) Error Tolerant Byte ALU (BALU)   The BALU is designed so that its bit planes are relatively independent. By adding a spare bit plane and changing the carry and shifting circuitry, a reconfiguration switch can be used to switch bit planes and make the BALU tolerate a single component malfunction.

5) TMR/Sparing   Using ideas derived from the reconfiguration switching, it is possible to combine the features of TMR and standby redundancy to design error tolerant circuits called TMR/Sparing which have higher reliability than a straight extension of TMR for $n > 3$.

Methods of designing and operating ultra-reliable computers are also studied. Operating ultra-reliable computers is discussed. The necessity for a "sentinal" to operate during turn-off periods is shown, and the similarity between computer "bring up" procedures at turn-on and after an error signal treated.

2. Investigation of Technologies

2.1 Scope. The investigation was limited to current and incipient computer technologies applicable to the CPU and memories. The components and devices of concern included transistors, diodes, capacitors, resistors, connectors (solder and weld joints) and ferrite cores.

2.2 Failure Phenomena. One concern in the investigation has been the mechanism of failure. It was hoped that a knowledge of the physical and electro-chemical phenomena involved in failure could be used to predict the nature of the distribution of failures over time. This proved useless because of the larger effect caused by manufacturing defects (see section 2.3). For system reliability calculations the values of the failure rates and the failure rate distribution (in time) must be assumed known. Another important effect is the observation [Nerber, 65] that solid state devices, but not connectors or cores, exhibit significantly lower failure rates under power-off conditions; lower by a factor of 2 to 10.

2.3 Measurements. Actual measurements of computer failure rate as a function of time follow the solid line curve depicted in Fig. 2.1, and not a straight horizontal line that the discussion in the above section would lead one to expect. The reason seems to

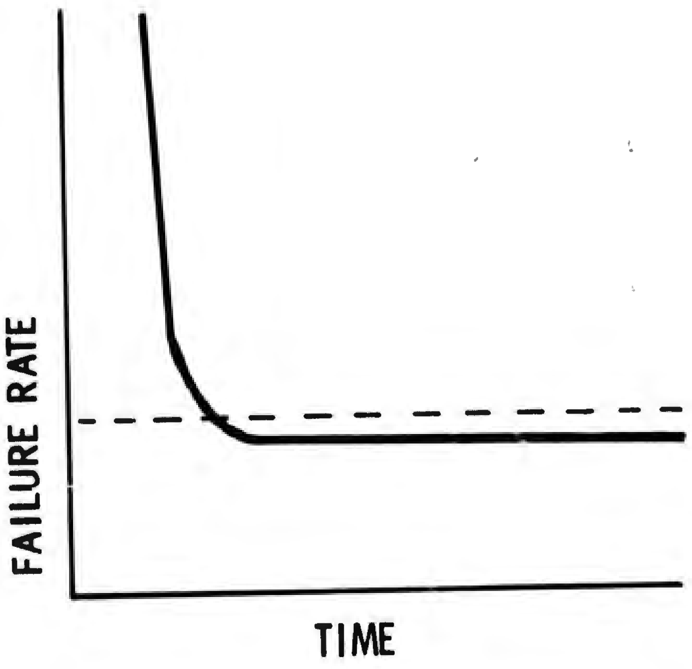


Fig. 2.1 Experimental failure rate.

be that initially, at least, the major cause of failures is manufacturing imperfections [Porter, 64 Raymond, 65]. That part of the curve above the dashed line is called the "infant mortality" rate or the "burn-in" rate. One reason for operating a space computer for a relatively long length of time prior to flight is to detect and replace defective components during this initial test period.

A statistical analysis [Lewis, 64] has been made of the measured failure rate for that portion of the curve to the right of the burn-in. A good fit to the distribution of failure incidents is a primary Poisson sequence (failure rate independent of time) followed by a secondary sequence. One plausible explanation for this lack of strict adherence to a pure Poisson distribution is the fact that the computers measured were constantly being repaired, and that there is no guarantee that the real trouble was found and fixed. In fact, there is good reason to believe that in many cases the observed failure just went away to come back again at a later date. Temperature, humidity, noisy power sources, and other environmental factors could cause this type of behavior.

__2.4 Failure Distribution Assumption.__ For the reason given in the above two sections, it is believed that after burn-in has occurred one is justified in assuming a constant value for the failure rate. This may be a slightly pessimistic assumption, since a very slight negative slope does seem to exist, but this is probably as accurate as the data warrants. This assumption leads to the well known Poisson distribution in which R, the probability of success for a mission of duration T is given by the equation $R = e^{-\lambda T}$ where λ is the sum of the individual failure rates of those components forming the system. As will be seen in Chapter 4, the Poisson distribution assumption makes reliability analysis and calculations feasible that would not be possible with almost any other failure distribution.

2.5 Failure Rates. From the field data, laboratory tests, and manufacturing experience, the predicted failure rates of Table 2.1 were obtained for a particular extant computer technology known as 4 Pi [BCJ, 67] and employed in MOL and other space vehicles.

SOLID FAILURES / 10^6 HOURS

	1968	1970	1970
LOGIC PAGE (= 475 Circuits)	4.7		2.6
ROS (2K 100 bit words)	20		14
MAIN STORE(ferritecore) (8 K 36 bit words)	47 (Solid State) +12 (Arrays)		37.5 11. 5
DRUM (16K 36 bit words)	15		9
POWER SUPPLY	15		12
SECONDARY STORE (A slower Main Store)	10 (Arrays only)		8

Table 2.1

2.6 Types and Consequences of Failures. The ratio of transient to solid type failures is large [AES-EPO 65]. New data from an installation (FAA-Atlantic City), where careful statistics are kept, give a ratio of 5 intermittents to 1 solid-type failure for the electronic portion of the computer during the first five months of

1967. This ratio, as discussed later in section 5.5, warps the machine organization to a great extent because intermittent failures need to be handled differently from solid failures.

This investigation did confirm previous opinions regarding the effects of failures on the circuit logic, and have generated more confidence in the efficacy and coverage of the diagnostic tests generated, discussed in Chapter 3.

2.7 Other Technologies Investigated.

Various technologies for use in computer memories were investigated. Film memories are still so new that no dependable failure rates are available. There are technical reasons for believing that their ultimate reliability will approximate those of ferrite core memories since a) the number of connections is about the same, b) the amount of circuitry involved is about the same, and c) the failure rate of the magnetic storage device itself is negligibly small in both technologies. The remaining uncertainty is the power consumption, and the trade-offs between access speed and power consumption. The first use of film memories in space computers might well be in writable control stores, useful for reconfiguration and/or circuit diagnosis.

Monolithic transistor technology was also examined for application to space computer memories. They seem to offer high speeds, low weights, small size and small unit size (4K bits) at a feasible cost.

Their disadvantages are two, a) they are volatile, i.e., whenever power is interrupted their contents are lost, and b) they have a much higher failure rate than an equivalent ferrite core memory.

Large Scale Integration technology was looked at only peripherally. It promises to offer lower power consumption and greater component reliability. Its diagnostic complexity, however, will be greater.

2.7.1 Some Basic Technological Difficulties Connected with LSI. There are basic technological difficulties connected with LSI from the standpoint of system reliability. Because of the inherent complexity of LSI modules, the circuits will necessarily be cyclic, i.e. feedbackloops will be present. LSI therefore seems to imply the necessity of development of algorithms for cyclic diagnosis and (or) appropriate checking circuitry. As discussed in section 3.6.1 there are basic difficulties in the practical application of existing classical cyclic diagnosis algorithms based on state-diagram or state-table models. The SCAN attack on cyclic circuit diagnosis [CMPR, 64; HS, 65]. involving adding extra input and output "pins" to render the circuit acyclic for diagnostic purposes, is something not appealing for pin-limited technologies.

As implied by Chapter 4 Reliability/Evaluation, failure coverage (the fraction of failures detected by tests or hardware) must be very high to achieve the ultra-reliability goal with present or near future Technology. This was a prime motivation for the development of the cyclic diagnosis algorithm in section 3.6.4.

2.7.2 The test-point insertion problem. Can a "tradeoff" be made between test-point insertion and the efficiency of algorithm? How to define "natural" break points? Only specific examples have been worked out and the test-point insertion problem remains to be solved for practical realization of LSI expectations.

One proposal: Test points be inserted at "natural" break points, in the sense of architecture, e.g. from a flow-chart description of the design. A criterion for "naturality" might be: "minimize the number of test points that must be added in order to have the ability to test each 'possible' failure." Here "possible" is placed within quotation marks because we would, in any given procedure, be computing a set of tests to detect some prescribed class [category] of failures, such as all single solid failures, or all single shorts or opens.

3. Algorithms for the Detection and Diagnosis of Failures

3.1 Summary of Diagnosis

3.1.1 Need for Diagnostic Testing Reliability calculations (Chapter 4) show that it is extremely difficult to achieve ultra-high reliability/availability using masking redundancy methods such as TMR [von Neumann, 56], Quadding [Tryon, 62], error-correcting codes [Peterson, 61], etc. Even when extensive hardware checking is used in a computer it is still mandatory that diagnostic tests be used to diagnose failures in the checking circuitry (cf. section 5.4.7). Without such tests an undetected first-failure in the checking equipment could prevent identification of a subsequent failure which might occur in the "checked" logic.

It is also shown in Chapter 4 that the greatest reliability/availability potential lies in using stand-by or sparing redundancy: such redundancy requires large amounts of checking circuitry and/or diagnostic tests to locate the failed device and correctly switch in a replacement for it. Again, even if checking circuitry is used to supplant diagnostic testing of the main computer hardware, it is still necessary to use such tests to examine the checkers and reconfiguration switches (cf. section 5.4.2) for failures. The reliability models in section 4.3 and the calculations in section 5.3 clearly exhibit that "failure coverage" – i.e. the fraction of failures of a given category covered or detected by hardware checking and an ensemble of tests-has a considerable impact on the Reliability/Mission-Time picture. Without sufficient failure coverage the apparent reliability advantages of stand-by redundancy cease to exist.

During the experimental development of ultra-reliable circuits implementing various functions and their associated checking circuits it is extremely valuable to have a design tool which permits fast evaluation of the checking and diagnostic failure coverage: TESTDETECT (cf. section 3.5) is just such a tool. In section 5.4.7 an interesting example is presented which shows how a checking circuit design can be developed and evaluated using this algorithm. The use of diagnosis algorithms to assist in the actual circuit design, rather than using them after-the-fact, is crucial to designing a truly ultra-available computer.

With the advent of LSI it no longer becomes feasible to introduce large numbers of extra inputs and outputs in each sequential or cyclic circuit in order to render it combinational for diagnosis purposes; cf. the SCAN concept [CMPR, 64; HS, 65]. However, LSI is amenable to the introduction of extra hardware to facilitate diagnosis. This means that a true sequential or cyclic circuit diagnosis algorithm is required, but that extra circuitry can be introduced into the circuit or the circuit designed in some non-minimum hardware implementation so as to simplify the computational problems involved in any cyclic diagnosis algorithm (cf. section 3.6.2 on the synchronous circuit model).

3.1.2 Review of Previous Developments Reference [Roth, 66] provides a comprehensive review of combinational or acyclic circuit diagnosis as exemplified in [CMPR, 64; GNR, 64; HS, 65; MA, 63]. Approaches to diagnosis based on the use of Boolean equations are presented in [Armstrong, 66; Poage, 63; Eldred, 59]. Typical of the current approaches to diagnosis of sequential or cyclic circuits are those of [Hennie, 64; Kime, 66; FM, 64] all of which use the classical models of [Moore, 56; Huffman, 54; Mealy, 55].

3.1.3 Summary of Newly Developed Algorithms Fundamental to all the diagnostic algorithms developed in this study is the new D-calculus developed in [Roth, 66] (cf. section 3.3). This D-calculus provides a clear concise means for performing failure analysis on both acyclic and cyclic logic circuits in a manner suitable for both manual and computer computation. Based on this calculus, the following three major diagnosis algorithms were developed:

1) D-Algorithm (DALG-II) is an algorithm which computes a test, if one exists, for any failure in an acyclic or combinational circuit.

2) TESTDETECT ascertains all failures in an acyclic circuit of a given variety detected by a given test consisting of specified signal values on the circuits inputs; it is a kind of converse of the D-Algorithm.

3) CD-Algorithm computes tests for failures in a certain well defined class of cyclic or sequential circuits (cf. section 3.6.2).

Both DALG-II and TESTDETECT were programmed on the APL time-sharing system [Iverson, 63; FIS, 64; FI, 66] for use with the usual stuck-at-1, stuck-at-0 failures.

Allied with the development of these three algorithms was the formulation of 1) a multiple-failure variation of the D-algorithm (cf. section 3.4.3) for use in finding a single test to detect one of several failures, 2) a Test-Response variation of TESTDETECT (cf. section 3.5.4) which not only tells if a failure is detected by a given test but also specifies the precise response when the failure occurs and 3) a synchronous circuit model (cf. section 3.6.2) suitable for facilitating cyclic circuit diagnosis.

The existence of all these algorithms is considered fundamental to the foundation upon which the automatically repaired computer of Chapter 5 is built.

3.2 Terminology

3.2.1 Definition of Basic Terms The following terms will arise frequently in the discussions which follow.

primary input (PI) -	a line which is not fed by any other line in the given circuit.
primary output (PO)-	a line whose signal is accessible to the exterior of the given circuit.
logic block (LB) -	a combinational logic device, i.e. a device whose present output is always the same well-defined Boolean function of its present inputs when it is functioning correctly. This function is specified by a given "singular cover" as defined in section 3.3.
successor -	logic block i is a successor of logic block j if i has an input from j.
predecessor -	logic block (or primary input) j is a predecessor of logic block i if j is an input of i.
circuit (CKT)-	an interconnection of LB's which is usually described by an enumeration of the predecessors and logic (or singular cover) for each LB together with a list of the primary outputs.
failure -	any transformation of the hardware which changes the logic function realized by any portion of this hardware.

test (for the failure) - an assignment of values 1, 0 or x to the primary inputs such that the value of at least one primary output signal will differ depending on whether or not the failure is present.

Several additional terms used exclusively in the development of the CD - Algorithm will be defined as needed in section 3.6.

3.2.2 Conventions

Although the various D-Algorithms and TESTDETECT can easily be extended to handle other failures and logic, the programmed versions described in this chapter are limited to single failures of the stuck-at-1, stuck-at-0 variety. Extension to more involved failures is merely a matter of defining more complicated primitive D-cubes of failure(cf. section 3.3).

A circuit is said to have memory or storage capacity if it is not combinational, i.e. its present output, even ignoring failures, is not always the same function of its present inputs. A CKT with memory or storage capacity (sequential circuit) must contain feedback loops, i.e. must be cyclic. A CKT will likewise be termed combinational if its present output is always the same well-defined function of its present input, except possibly for the occurrence of failures. It is not necessarily true that a combinational CKT is acyclic although in this report such an assumption will usually be made.

All circuits considered here are composed of AND, OR, NAND, NOR, and XOR (exclusive-OR) logic blocks. When describing such circuits it is possible to assign the same number or name to both the logic block and its output line since the blocks are all single output. This convention will be used in all acyclic circuit algorithms so that the terms "block i" and "line i" can be used interchangeably. This convention is modified slightly in the section treating cyclic circuits wherein multiple output latches are permitted (cf. section 3.6.2).

For acyclic circuits, it will also be assumed that the n lines (or logic blocks) in the circuit are assigned unique number labels from the integers 1,2,..., n subject to the constraints that each line shall have a higher number than its predecessors, and the primary inputs shall be assigned successive numbers starting with 1. With this convention we define the terms backward to mean in the direction of decreasing line numbers and forward to mean in the direction of increasing line numbers. A special labelling algorithm is defined in section 3.6.2 to extend this numbering procedure to cyclic circuits.

3.2.3 Diagnosability Theorem

Theorem 3.1 Given a combinational Boolean Transfer Function, TF, there exists a CKT implementation of TF which cannot be diagnosed except by exhaustive application of all 2^n tests, where n = number of primary inputs.

Proof: If the circuit is implemented in two level disjunctive normal form, all possible inputs for which TF = 1 must be applied in order to test all the AND gates for stuck-at-0 failures. If the conjunctive normal form is used, all possible inputs for which TF = 0 must be applied in order to test the OR gates for stuck-at-0 failures. Since the implementation is not known, it could be either of the above. Thus all possible inputs must be applied to insure complete testing.

Q.E.D.

The impact of this theorem is that the actual circuit structure implementing a combinational logic function must be known if one hopes to perform failure detection by other than exhaustive testing. Knowledge of the TF alone is necessary but not sufficient.

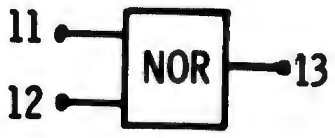
For cyclic or sequential machines it is also necessary to know the actual circuit implementation if one hopes to obtain efficient diagnosis. Knowledge of the input-output functions and state transitions apparent at the circuit terminals is not sufficient since the actual circuit could have equivalent "machine states", something which could cause circuit states to be missed during diagnosis.

3.3 Summary of D Calculus

A "calculus of D-cubes" was developed in [Roth, 66] which allowed one to describe analytically the behavior of, and to compute diagnostic tests for, failing automata. A brief resume of this calculus will be presented here.

The functional transformation for each logic block in the circuit is given in terms of its singular cover (Roth, 60), a kind of truth table description for the block's function with extra columns present in the table to account for circuit variables other than those associated directly with the given block. As usual, x's or blanks are used to denote that the position may be either 0 or 1. Each row in the singular cover is termed a singular cube. Such cubes are called prime when no larger singular cube of the function can exist which contains the given cube (prime singular cubes correspond to prime implicants.) Figure 3.1a shows a NOR block, with inputs 11 and 12 and output 13, which may be thought of as being imbedded in some larger circuit. Its singular cover, made up of three prime singular cubes, is given in Fig. 3.1b.

The set of so-called primitive D-cubes, pdc, for this NOR block are given in Fig. 3.1c. These D-cubes have the following interpretation: the symbol D may assume only the two values 0 and 1, and the assignment must be held fixed for all the D's in a given D-cube. The symbol $\bar{D}$ denotes the complement of the value assigned to the D. Thus the D-cube $D0\bar{D}$ represents the two cubes 001 and 100. In effect, this D-cube specifies that when the NOR is functioning properly and a 0 is placed on line 12, the output must be the complement of the signal on line 11. The two dual sets of D-cubes are provided since the assignment of the D or $\bar{D}$ to the inputs is, at this stage, arbitrary.



(a)

11	12	13
0	0	1
	1	0
1		0

(b)

11	12	13
D	0	$\bar{D}$
0	D	$\bar{D}$
D	D	$\bar{D}$
D	$\bar{D}$	0

11	12	13
$\bar{D}$	0	D
0	$\bar{D}$	D
$\bar{D}$	$\bar{D}$	D
$\bar{D}$	D	0

(c)

Fig. 3.1 Singular cover and primitive D-cubes for a two input NOR gate.

An algorithm for constructing the pdc's is as follows. Pick any subset σ of coordinate positions which correspond to 0 or 1 values in a given singular cube α. Complement these values and the output value of α to form a new cube α^*. Intersect [Roth, 59; Roth 60] α^* with each singular cube β in the cover. When α^* and β have a non-empty intersection $\gamma = \alpha^* \sqcap \beta$, a pdc can be formed from γ by replacing the σ coordinates in γ which are 1 by a D ($\bar{D}$) and those which are 0 by a $\bar{D}$(D). The same replacement is also performed on the output coordinate of γ. The alternate values in parenthesis are used to generate the dual pdc's. As an example, pick the first singular cube in Fig. 3.1b and select coordinate 11 as the one to be complemented. Then γ = 100 is non-empty. Replacing the 1 at coordinate 11 by a D and the 0 at the output coordinate by a $\bar{D}$ produces the first pdc in Fig. 3.1c. Generation of pdc's like the bottom ones in Fig. 3.1c is accomplished by leaving the output coordinate untouched in all the computations.

A <u>primitive D-cube of failure</u> pdcf is used to specify how to exhibit a given failure at the site where it occurs. The pdcf construction procedure will be limited to the simple stuck-at-1 or stuck-at-0 failures considered in this paper; [Roth, 66] gives a more general construction. When a line or block is to be tested for a stuck-at-1 (o) failure, it is necessary to attempt placement of a 0 (1) on the line. Thus the first step in the construction is to select for the given block associated with this line a prime singular cube with a 0 (1) at its output coordinate position. Then the output 0 (1) is replaced by a $\bar{D}$ (D). The use of the D and $\bar{D}$ here is by convention: a D meaning 1 in the good circuit and 0 in the failing circuit. For the NOR block of Fig. 3.1 the stuck-at-1 pdcf's are x1$\bar{D}$ and 1x$\bar{D}$ while the stuck-at-0 pdcf is 00D. For primary input lines, the pdcf's are merely a D or $\bar{D}$ at their respective coordinate position.

A new <u>D-intersection</u> can be defined via an extension of the usual singular cube intersection [Roth 59; Roth 60]. Let α and β be two D-cubes. Then their D-intersection $\alpha \sqcap \beta$ is defined using the coordinate D-intersection in Table 3.1 and the following rules,

$$\alpha \sqcap \beta : \begin{cases} 1. & \phi \text{ (empty) if any coordinate intersection is } \phi. \\ 2. & \psi \text{ (undefined) if any coordinate intersection is } \psi. \\ 3. & \text{The cube formed from the respective coordinate intersections if neither 1 or 2 holds.} \end{cases}$$

Thus 01xDx $\sqcap$ 0x1D$\bar{D}$ = 011D$\bar{D}$ and 01xDx $\sqcap$ 0x1$\bar{D}$D = ψ

∩	0	1	x	D	D
0	0	∅	0	Ψ	Ψ
1	∅	1	1	Ψ	Ψ
x	0	1	x	D	D
D	Ψ	Ψ	D	D	Ψ
D	Ψ	Ψ	D	Ψ	D

∅ = empty

Ψ = undefined

Table 3.1 Coordinate D-intersection.

The algorithm to be described in this paper will not actually store the singular cover or the set of primitive D-cubes for each block in the circuit; it will dynamically compute them on an as-needed basis. This will be possible since, for the function types currently being used (OR, AND, NOR, NAND, XOR), the computation is fairly simple. For example, the singular cover for an n input OR gate contains the information that the output is 0 if and only if all inputs are 0, and is 1 otherwise. The primitive D-cubes are constructed by picking any subset of the n inputs, assigning these inputs the value D (or $\overline{D}$), all other inputs 0, and the output D (or $\overline{D}$). Similar procedures can be stated for the other block types.

Now let F be a logic failure in a circuit and let T be a test (a particular assignment of 0's, 1's, and x's to the primary inputs). Then each T and F can be used to define a D-cube c(T,F), termed the D-cube defined by T anf F, in the following manner: if, for any line or coordinate i, the good and the failing circuit have the same value (either 0,1, or x), this value is placed in c(T,F) at coordinate i; if the good circuit has a 1(0) and the failing circuit a 0(1) on line i, then c(T,F) receives a D ($\overline{D}$) in coordinate i. This choice of how the D and $\overline{D}$ are assigned is, as with the pdcf, by convention. The importance of the D-cube c(T,F) is that it allows one to describe completely the operation of the good and failing circuit under the input T: set D = 1 (thus $\overline{D}$ = 0) and one gets the good circuit's response, set D = 0 ($\overline{D}$ = 1) and one gets the failing circuit's response. If T is indeed a test for the existence of F then c(T,F) must have a D or $\overline{D}$ on at least one primary output so that these two responses would differ. Reference [Roth, 66] develops a set of lemmas which show that c(T,F) must contain a connected chain of D's and/or $\overline{D}$'s linking the site of the failure to a primary output, and that c(T,F) could be constructed by a D-intersection of primitive D-cubes and singular cubes of the blocks in the circuit. Here <u>connected D-chain</u> is construed to mean a block (other than the site of failure) has a D or $\overline{D}$ on its output if and only if at least one of its inputs has value D or $\overline{D}$.

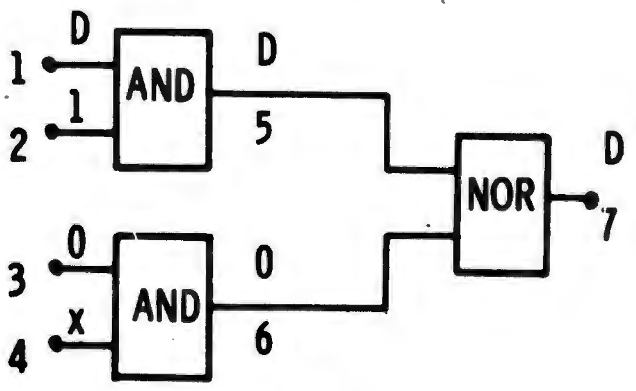
As an example, let F be the failure "line 1 stuck-at-0" in the circuit of Fig. 3. 2a and let test T consist of the values 110x on the primary inputs 1, 2, 3, and 4 respectively. Then c(T,F)=D10xD0$\overline{D}$. The connected D-chain is composed of the D on failing line 1, the D on intermediate line 5 and the $\overline{D}$ on the primary output line 7. It is also possible to show that under the rules of D-intersection, c(T,F) can be constructed as

$$c(T,F) = \text{Dxxxxxx} \sqcap \text{D1xxDxx} \sqcap \text{xxxxD0}\overline{\text{D}} \sqcap \text{xx0xx0x}$$

where the first term is the pdcf for the failing primary input line 1 and the last three terms are pdc's and singular cubes obtained from the pdc lists and singular covers shown in Fig. 3.2b (note that for brevity the dual set of pdc's are not shown). In the algorithm of reference [Roth, 66] the intersection of pdcf and the pdc's is termed the <u>D-drive</u> of the D-chain to a primary output. The intersection with the singular cubes, termed <u>consistency</u> follows the D-drive and is used to develop a consistent set of primary input values which will account for all lines set to 0 or 1 during the D-drive. In this example there was an arbitrary choice as to which singular cube to use: an equivalent assignment would have been to set line 4 to 0, in which case T' would be 11x0 and c(T',F) = D1x0D0$\overline{D}$. Finally, one might pick 1100 as a test T". The resulting c(T",F) = D100D0$\overline{D}$ is contained in (actually D-contained cf. [Roth, 66]) both c(T,F) and c(T',F).

3.4 D-Algorithm Version II (DALG-II)

A variation of the D-Algorithm Version II(DALG-II) suitable for manual implementation will be presented first via an example. This is followed by a description of the APL rendition of DALG-II. It will be seen that although DALG-II contains substantial improvements and changes over the version presented in reference [Roth, 66] (DALG-I), the basic philosophy is the same: first



(a)

1	2	3	4	5	6	7
1	1			1		
	0			0		
0				0		
		1	1		1	
			0		0	
		0			0	
				0	0	1
					1	0
				1		0

1	2	3	4	5	6	7
1	D			D		
D	1			D		
D	D			D		
D	$\bar{D}$			0		
		1	D		D	
		D	1		D	
		D	D		D	
		D	$\bar{D}$		0	
				0	D	$\bar{D}$
				D	0	$\bar{D}$
				D	D	D
				D	$\bar{D}$	0

(b)

Fig 3.2 Circuit illustrating properties of c(T, F).

select a primitive D-cube of failure, then try successively to D-intersect with D-cubes and singular cubes of the logic blocks in order to generate c(T.f).

3.4.1 Manual Example of DALG-II Three of the salient features of DALG-II will now be discussed while, for the benefit of those familiar with DALG-I, contrasting them with the methods used in [Roth, 66].

The DALG-II algorithm defines the activity vector A to consist of a listing of all those blocks which, at the given stage of the computation, have D's(or $\overline{D}$'s) on their inputs, an x on their output, and whose pdc intersection into the test cube has not been found to lead to an inconsistent line assignment. Thus the active blocks are on the D-frontier, or leading edges of the connected D-chain that the algorithm is driving to the primary outputs: decisions must be made concerning what to do regarding driving the D's through these blocks.

The second feature is that as soon as a decision to drive a D through a block is made (in other words to D-intersect with a primitive D-cube for that block) the full implications of the resulting line assignments from x to 0, 1, D, or $\overline{D}$ are carried throughout the circuit. Here "implications means all other line assignments which are a forced consequence of the totality of assignments already made up to this point. For example, if an AND block with an output equal to x receives a 0 input, its x output must be set to 0, or if an AND block with all inputs x receives a 1 output, all its inputs must be set to 1, or whenever an AND block has all its inputs set to 1 or D, the output must be set to a D, etc. The use of the implication concept at each step should make DALG-II several orders of magnitude more efficient than DALG-I since inconsistent decisions will be discovered much sooner in the processing. This is the major innovation of DALG-II over DALG-I.

The third change involves the order in which active blocks in A are processed in the drive to the primary outputs. Discussion of this will be reserved for later: let it suffice to say that DALG-II picks the lowest numbered block, skipping over those it has already found to lead to an inconsistency.

The manual variation of the algorithm to be used here will proceed roughly as follows: A primitive D-cube of failure (pdcf) will be selected and all singular cubes or D-cubes whose choice is implied by pdcf placed in a set I^o(pdcf). Then an initial test cube tc^o = pdcf $\sqcap$ $[\partial I^0(\text{pdcf})]$ is formed[1] and the activity vector A^o defined. At the kth step in the algorithm a primitive D-cube pdc^k is developed for some block in the activity vector A^{k-1}. Then the set $I^k(tc^{k-1}, pdc^k)$ of all those cubes which are new implications

1. We define ∂C to be the D-intersection of all the cubes in the set $C = \{c_1, c_2 \ldots\ldots, \}$ i.e. $\partial C = c_1 \sqcap c_2 \sqcap \ldots\ldots\ldots$.

of what has been done to date, is generated. Finally, the test cube $tc^k = tc^{k-1} \sqcap pdc^k \sqcap [\partial I^k(tc^{k-1}, pdc^k)]$ is formed and A^k defined. The iterations cease when a primary output has been reached with the D-chain of tc^k. At this point, blocks which were assigned output values 0 or 1 by the D-drive without having appropriate inputs to account for this output, have their signals "driven" back towards the primary inputs. This is done by generating intersections with the appropriate singular cubes in an attempt to find a consistent assignment of the primary inputs which will provide the desired signals. Because the implication process was used during the D-drive, this consistency operation will always involve a choice of one of several possible singular cubes for each block it encounters, i.e. if there is no choice, the implication process would have already used the singular cube in forming some tc^k.

When either the D-chain "dies" (A becomes empty) or an inconsistency is discovered, the process must back up to the last arbitrary choice of pdc^k or singular cube and make an alternate selection.

With these points in mind let us begin the construction of a test for the failure "line 6 stuck-at-0" for the circuit in Fig. 3.3 To test a line for being stuck-at-0 it is necessary to force a 1 on that line in the good circuit, so pdcf must be x00xxDxxxxxx. Since there are no implications for these three line assignments, $I^o = \emptyset$, tc^o = pdcf, and $A^o = \{9, 10\}$. In picking the block in A^o for which pdc^1 will be generated, the first arbitrary choice arises, DALG-II would pick the lowest numbered entry in A, line 9, and form pdc^1= 0xxxxDxx$\bar{D}$xxx. The output of block 5 is forced to be a 1 since both inputs are now 0, and this assignment in turn forces the output of 8 to 0. Thus $I^1(tc^o, pdc^1) = \{$0x0x1xxxxxxxx, xxxx1xx0xxxxx$\}$. The new test cube tc^1 is constructed as indicated below.

$$\begin{array}{llll}
 & & \text{x00xxDxxxxxx} & tc^o \\
 & & \text{0xxxxDxx}\bar{\text{D}}\text{xxx} & pdc^1 \\
 & & \text{0x0x1xxxxxxxx} & \multirow{2}{*}{$\left.\right\} I^1$} \\
 & \sqcap & \text{xxxx1xx0xxxxx} & \\
\hline
tc^1 & & \text{000x1Dx0}\bar{\text{D}}\text{xxx} &
\end{array}$$

The activity vector A^1 becomes $\{10, 12\}$.

Again, picking the lowest numbered active block, in this case 10, pdc^2 = xxx0xDxxx$\bar{D}$xx. The 0's at cooordinate 2 of tc^1 and coordinate 4 of pdc^2 imply a 1 on line 7 which in turn implies a 0 on line 11. At this point, line 9 and 10 are $\bar{D}$ and line 8 and 11 are 0, thereby forcing a D on the PO line 12. Thus I^2 (tc^1, pdc^2)$\{$x0x0xx1xxxxxx; xxxxxxx1xxx0x; xxxxxxxx0$\bar{D}\bar{D}$0D$\}$ and tc^2 is,

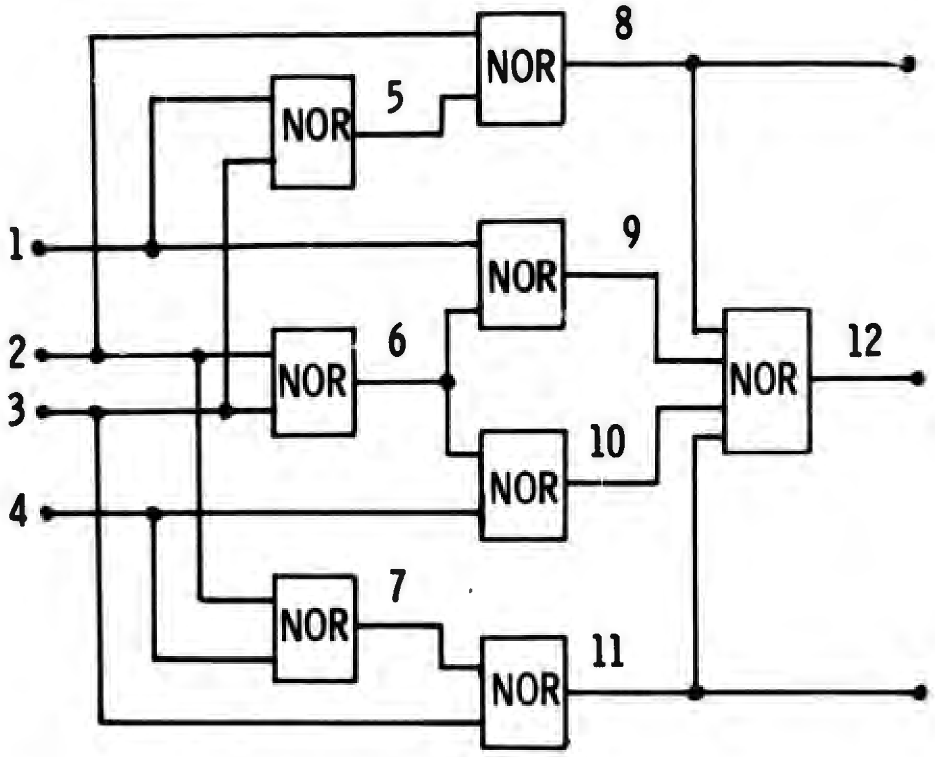


Fig. 3.3 Example illustrating the D-Algorithm (DALG-II).

$$\begin{array}{lll}
 & 000x1Dx0\overline{D}xxx & tc^{1} \\
 & xxx0xDxxx\overline{D}xx & pdc^{2} \\
 & x0x0xx1xxxxxx & \\
 & xxxxxx1xxxx0x & I^{2} \\
\sqcap & xxxxxxx0\overline{D}\overline{D}0D & \\
\hline
\end{array}$$

$tc^{2} = 00001D10\overline{D}\overline{D}0D$

The D-chain has reached the primary output so the consistency operation would normally be started. However, in this example all blocks with outputs 0 or 1 have their signals already accounted for by their inputs. Thus tc^{2} is c(T,F), with input test pattern T= 0000.

It is informative to examine what happens if block 12 in A', instead of block 10, is selected for forming pdc^2. Then pdc^2 = xxxxxxx0$\bar{D}$00D and the following initial implications may be noted; the 0 on line 11 forces a 1 on line 7 (line 3 is fixed at 0) and the 0 on line 10 forces a 1 on line 4 (line 6 is currently a D). When block 7 is examined for implications it is found to be inconsistent: the NOR gate cannot have an input value 1 and an output value 1.

The lack of success in the second part of the example serves to illustrate that, for the given failure, the only test that exists depends on having the failure signal propagate through a chain which reconverges at block 12, "sensitizing" any single path will <u>not</u> produce a test!

The results of these examples will be used to illustrate the key features of DALG-II and point up contrasts with DALG-I. Although the D-chain in the successful test passes through three blocks (9,10, and 12), only two decisions had to be made; implication automatically extended the D-chain through block 12. The implications process used during the D-drive of DALG-II really amounts to a mixture of D-drive and consistency as defined for DALG-I. Next to the use of implication, the biggest change in strategy is the method in which the choices are made during the extension of the D-chain to the output. Because DALG-II always selects the lowest numbered block in A (which it has not found to lead to an inconsistency) and because the block numbers are ordered as previously mentioned, the extension of the D-chain is an orderly level by level progression to the outputs. Further, DALG-II stops as soon as any segment of the D-chain reaches a PO since leaving any other segments of the D-chain uncompleted cannot result in any contradictions to the test as currently developed. By leaving such segments uncompleted we give them don't care status (they may or may not reach other PO's depending upon what is done in consistency.).

An alternate strategy for selecting blocks from A would be to pick the highest numbered one. This would amount to giving priority to a rapid "plunge" to the outputs along a sensitized path. At this point in the development it is now known which of the two strategies is more efficient in circuits with fan-out. In circuits without fan-out the methods are identical since A will only contain one block number at any given time.

3.4.2 The APL Rendition of DALG-II

Appendix 1.1 contains a detailed APL [FI, 66; IVERSON, 62; FIS,64] specification of DALG-II and its support functions together with a glossary of key terms. Here we will just summarize the major functions and the tasks they perform. This is followed by an example illustrating how the interactive computation provided by these APL functions facilitate diagnostic test generation.

A "stand-alone" function, CIRCUIT, is used to provide interactive communication for entry of the information necessary to define the circuit. The actual APL rendition of DALG-II is viewed through the function DALG-II, which reads in the failure information, establishes the pdcf's, calls upon six other functions to execute the actual algorithm, and then prints the results. The six major functions used to compute the test have a hierarchy as shown in Fig. 3.4. They use 13 subordinate functions also listed in Appendix 1.1. DALG controls the D-drive to the primary outputs. DB — DRIVE is the function which carries out all implications that can be made in the direction of forcing signals back towards the primary inputs during the D-drive. DF DRIVE is a similar function for forcing implied signals forward towards the primary outputs. CD DRIVE is used to perform the consistency operation whenever DALG en-

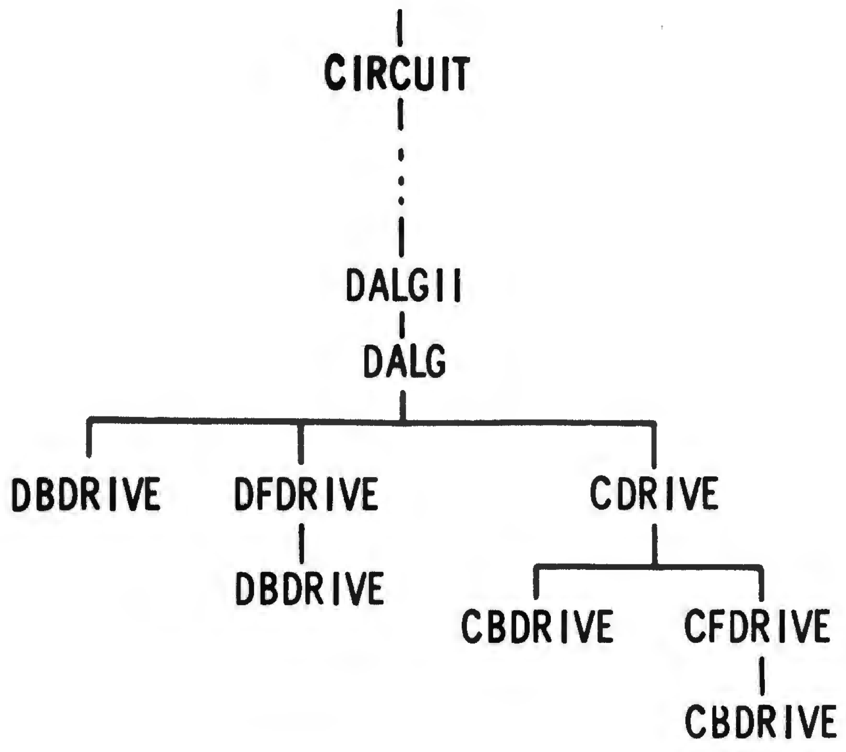


Fig. 3.4 Function structure of DALG-II.

counters a primary output in D-drive. It uses backward (CB DRIVE) and forward (CF DRIVE) implication functions in a manner similar DALG. The chief differences between the DB DRIVE, CB DRIVE and the DF DRIVE, CFDRIVE functions will be discussed later.

Fig. 3.5 shows the computer-user communication generated during a typical APL run for the example circuit used in the previous section. When CIRCUIT is executed, the function issues a command followed by the ⎕: symbol denoting that it is ready to accept the requested information from the user. Through this back-and-forth communication the circuit structure is defined in the computer. Next, DALGII is executed and through two more interactions the list of failures are defined. Then the algorithm is sequentially executed on the failure list just entered. For each, it prints out the failure followed by a sequential listing of all line values it specified during the test development, i.e. the contents of the final test cube tc. If a test does not exist for the given failure, the statement NO TEST appears.

Thus with the relative ease provided by preplanned interactive computation support a designer can enter his circuit definition and rapidly obtain diagnostic tests for whatever set of failures he desires. The actual CPU time required by DALGII to generate the six tests shown in Fig 3.5 was one minute. Based on preliminary experimental results (for circuits of 100 blocks or less) it appears that the DALGII execution time (in seconds) per test is roughly equal to twice the number of lines in the circuit. However, there is a marked tendency for the data to be more scattered for large circuits, indicating that a dependence on circuit structure may alter the linear relation.

```
         CIRCUIT
ENTER PRIMARY INPUTS.
⎕:       1 2 3 4
ENTER PRIMARY OUTPUTS.
⎕:       8 11 12
ENTER LOGIC.
⎕:       8ρNOR
ENTER PREDECESSORS FOR THE INDICATED BLOCK.
BLOCK 5
⎕:       1 3
BLOCK 6
⎕:       2 3
BLOCK 7
⎕:       2 4
BLOCK 8
⎕:       2 5
BLOCK 9
⎕:       1 6
BLOCK 10
⎕:       4 6
BLOCK 11
⎕:       3 7
BLOCK 12
⎕:       8 9 10 11
END OF CIRCUIT DEFINITION.

         DALGII
ENTER LINES TO BE TESTED.
⎕:       6 6 1 2 3 4
ENTER STUCK VALUES FOR THESE LINES.
⎕:       0 1 0 0 0 0
TEST FOR LINE 6 STUCK-AT-0
 0 0 0 0 1 D 1 0 D̄ D̄ 0 D
TEST FOR LINE 6 STUCK-AT-1
 0 1 1 0 0 D̄ 0 0 D D 0 D̄
TEST FOR LINE 1 STUCK-AT-0
 D 0 0 X D̄ 1 X D 0 0 X X
TEST FOR LINE 2 STUCK-AT-0
 X D 0 0 X D̄ D̄ X X D D X
TEST FOR LINE 3 STUCK-AT-0
 0 0 D X D̄ D̄ X D D X X X
TEST FOR LINE 4 STUCK-AT-0
 X 0 0 D X 1 D̄ X 0 0 D X
```

Fig. 3.5 Typical computer-use intercommunication during a DALG-II execution.

3.4.3 Multiple-Failure D-Algorithm. In an LSI or even MSI (M for Medium) environment the occurance of any one of several possible failures is sufficient to require replacement of an entire circuit module. Thus, it would be desirable to have a means whereby the previously described D-Algorithms could be modified so as to develop one test T which will diagnose the occurance of any one of a set of failures; the set usually, but not necessarily, being associated with a single replaceable module. This would be of use in reducing the number of tests required for diagnosis.

The multiple-failure D-algorithm or m.f.DALG described here does the following: given a category $\underline{\underline{F}}$ of failures of an acyclic logic circuit, m.f. DALG computes a single test T, if one exists, which will detect any one of this category $\underline{\underline{F}}$ of failures; if no such test exists, it will so ascertain.

The trick of this variant of D-algorithm can be understood by considering the case where $\underline{\underline{F}}$ consists of two failures, say A and B. Let TA denote a Test Cube (formerly denoted TC) being developed for failure A; concurrently being developed is a Test Cube TB for failure B. Along-side of each is a third test cube TC which keeps track of the constraints being "interleavedly" imposed upon the response pattern of the correctly functioning circuit by the development of both TA and TB. If conflicts occur in this "parallel" development (of TA and TB) then the algorithm "steps back" in the decision procedure - either for the TA-procedure or the TB-procedure or, possibly, both and proceeds again along this "2D branching procedure. TC is explicitly computed according to the following formulae:

$$\partial_A TA \sqcap \partial_B TB \equiv TC$$

where $\partial_A TA$ is an "operator" consisting of replacing each coordinate having value D with a 1, each coordinate having value $\bar{D}$ with a 0, all other coordinates remaining fixed. Similarly for $\partial_B TB$. And $\sqcap$ is the "D-cube intersection". [cf. section 3.3].

3.5 TEST DETECT

TEST-DETECT is a kind of converse of the D-Algorithm:

given a test T it computes the set of <u>all</u> failures detected by T. A method for ascertaining all failures F detected by T would be to construct c(T,F) for each F and assay whether or not any PO - coordinate had value D or $\bar{D}$. TEST-DETECT is a substanially refined version of this procedure.

3.5.1 Test Detect Algorithm. If TC denotes the vector of signals assigned by the test T to each line of the logic circuit, then, if TC_i has the value a= 1 or 0, T cannot test for the failure "line i stuck-at-a" since the good and failing circuits would appear identical. Thus if F_i denotes the failure line i stuck-at-$\bar{a}$, we need only concern ourselves with investigating such F_i. Further, the set of failures detected by T can now be uniquely specified by a vector FD of lines on which T detects a failure: if $i \in FD$ and $TC_i = a$ then T detects the failure F_i.

We now describe a method for effectively computing $c(T,F_i)$, hence for computing whether or not i can be placed in FD. We do this by constructing, ala the D-algorithm, the D-chain emanating from line i. The activity vector A and its use in producing an orderly level by level march to the PO's is as before. However, all line signal values are fixed so that we are only concerned with whether or not the D-chain propagates through the given block, i.e., whether or not the inputs correspond to the input section of a pdc which has a D or $\bar{D}$ at its output coordinate. For the sake of storage efficiency, c(T,Fi) is never explicitly generated. Instead we record that a line in TC has a D or $\bar{D}$ on it by placing the lines number in a list DL. Thus, if $i \in DL$ and $TC_j = 1$ (or 0) then coordinate j in $c(T,F_i)$ is D (or $\bar{D}$), otherwise TC and $c(T,F_i)$ agree. Since we are only interested in those lines for which active computation is being carried out, the definition of DL (and its use in the APL program) will be modified so as to exclude those lines which have already been processed and do not have successors in A.

When the D-chain reaches a PO, $DL \cap PO \neq \emptyset$, we can conclude that T detects F_i and stop the generation of $c(T, F_i)$. If the D-chain dies, $A = \emptyset$, before this occurs, T does not detect F_i. These are the two basic criteria for stopping computation. If the lines are examined for inclusion in FD in numerical order, starting with the highest numbered one, the status of line $j (j > i)$ has been determined before that of line i. Then the following lemma can be used to develop a third criterion for stopping which truncates the formation of $c(T, F_i)$ and greatly increases the efficiency of TEST-DETECT.

Lemma 3.1 If at any stage in the computation of $c(T, F_i)$ there is only one line j in DL, then i is in FD if and only if $j \in FD$.

Proof For the portion of the circuit corresponding to those blocks with number greater than j, the inputs (0, 1, D or $\bar{D}$) are exactly the same as when j was originally examined since, by hypothesis, j is the only line in DL. Hence $c(T, F_i)$ must agree with $c(T, F_j)$ in these positions and any conclusions made regarding j being in FD will also hold for i.

As an illustration of TEST-DETECT, let us examine the circuit in Fig. 3.6 wherein the test T assigns the value 1 to all the PI's. For all lines, the appropriate entry in TC is shown as the number above the line. When the first two lines, 12 and 11, are processed, placing their numbers in DL automatically satisfies the test $DL \cap PO \neq \emptyset$, i.e. if $i \in PO$ it can immediately be placed in FD. Line 10 has only one successor, 12, and when this entry in A is examined it is found that no pdc for block 12 exists which has a 0 at coordinate 9 and a D or $\bar{D}$ at coordinate 12. Hence A becomes empty and 10 cannot be placed in FD. Similarly line 9 is found not to be in FD while line 8 is. When line 7 is processed, DL = 7 and A = 11. A pdc with the required values (a $\bar{D}$ on line 7, a 1 on line 3, and a D on line 11) exists for block 11. Thus DL = 11 and A = 12. Since DL has been reduced to a single entry which is in FD, we conclude, using Lemma 3.1 that 7 must be in FD.

Investigating line 6 means that initially DL = 6 and A = $\{9, 10\}$. Since appropriate pdc's exist for both 9 and 10, two iterations in the in the D-chain extension will produce DL = 9, 10 and A = 12. The cube 1DD1D is a pdc (involving coordinates 8 through 12 respectively) for block 12 so the D-chain can be extended through 12. Now DL = 12 and Lemma 3.1 can again be applied to show line 6 is in FD.

Proceeding in a similar manner, lines 4 and 2 are found to be in FD while lines 5, 3 and 1 are not.

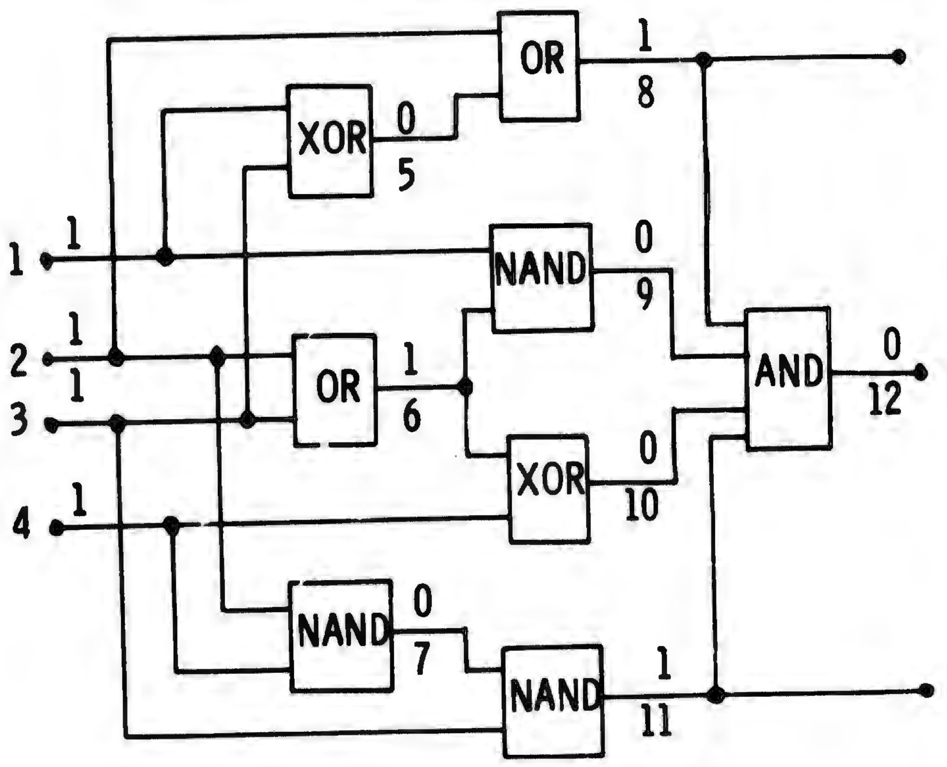


Fig. 3.6 Example used to illustrate TEST-DETECT.

3.5.2 Use of APL Rendition   The APL rendition of TESTDETECT and its support functions are listed in Appendix 1.2 together with a glossary of pertinent terms. Fig. 3.7 illustrates the computer-user interaction required to execute these functions for the circuit in Fig. 3.6. Since the only difference between the circuit used as an example in Section 3.4.2 and the circuit used here is that the logic of each block is different, there is no need to reexecute the function CIRCUIT (the two examples were run consecutively); only the logic is respecified. Then the support function PATTERNS is used to enter the values of TC[PI] for which TESTDETECT runs are requested. In this example all $2^4 = 16$ possible inputs are being used. Executing DETECT causes TESTDETECT to be run for each pattern entered. The results are displayed in tabular form as shown with each row representing one run. The left portion of the table gives the input pattern. Column i of the right portion of the table indicates the status of line i for the given test according to the following rules:

0 =	tested for stuck-at-0
1 =	tested for stuck-at-1
(blank) =	not tested.

```
      L←0,0,0,0,XOR,OR,NAND,OR,NAND,XOR,NAND,AND

      PATTERNS
ENTER PRIMARY INPUT PATTERNS, ONE AT A TIME.
TYPE  ALL  IF ALL 2*ρPI PATTERNS ARE DESIRED.
TYPE  END  TO STOP DATA ENTRY.
⎕:
      ALL


      DETECT
 1 2 3 4 | 1 2 3 4 5 6 7 8 9 0 1 2
---------+-----------------------
 0 0 0 0 | 1 1 1   1     1     0 1
 0 0 0 1 | 1 1 1   1     1     0 1
 0 0 1 0 | 1   0   0   0 0     1 1
 0 0 1 1 | 1 1 0   0   0 0     1 1
 0 1 0 0 | 1 0 1 1   0   0 0 0 0 0
 0 1 0 1 |   0   0   0   0   1 0 1
 0 1 1 0 |     0 1     0 0     1 1
 0 1 1 1 |   0   0   0 1 0   1 0 1
 1 0 0 0 | 0   1 1 0     0   1 0 1
 1 0 0 1 | 0 1 1 0 0 1   0 0 0 0 0
 1 0 1 0 | 0 1 0   1   0 1     1 1
 1 0 1 1 | 0 1 0   1   0 1     1 1
 1 1 0 0 | 0   1         0 1   0 1
 1 1 0 1 |   0       0   0     0 1
 1 1 1 0 |   0 0 1     0 0     1 1
 1 1 1 1 |   0   0   0 1 0     0 1
```

Fig. 3.7 Computer-user interaction for a TESTDETECT execution on the circuit of Fig. 3.6 .

The CPU time consumed during the running of this example was three minutes. Experimental results on circuits with less than 100 lines indicates that the execution time, in seconds, per test is roughly equal to the number of lines in the circuit. As with DALGII there is a tendency for the data to become more scattered as larger circuits are used, indicating a dependency on circuit structure.

3.5.3 Strategies Two objectives of an efficient diagnostic test generation procedure are to develop rapidly a small number of tests which will detect all failures in the circuit and then develop a somewhat larger set which will be capable of locating the site of the failure down to the smallest replaceable module. Several strategies may be employed in utilizing DALG-II and TEST-DETECT in such a procedure. These strategies must all provide a means for assigning values to the unspecified PI's (i.e. the ones that are x) in the tests produced by DALG-II since the present version of TEST-DETECT requires all PI's to be 0 or 1. (A version of TEST-DETECT without this restriction is under development). They should also make use of the greater speed of TEST-DETECT relative to that of DALG-II.

Three strategies for assigning the x's in the tests generated by DALG-II will be discussed. It is possible to assign values to the x's and simultaneously to reduce the number of tests by replacing a subset of tests by their intersection. For example, let T^1 = xx011 and T^2 = 100xx. Then their intersection, 10011, can be used in their stead. The difficulty with this strategy is that finding a minimum set of intersections which can replace a large number of tests, results in a "covering" problem [Roth, 59].

A second strategy would involve assigning the x's in such a manner that tests become rotations of each other, again reducing the number to be stored. In the previous example the test 00011 is obtained from T^1 by assigning the x's the value 0. If the two x's in T^2 are assigned the values 0 and 1, T^2 is a 1-bit right rotation of 00011.

Finally, the concept of "deadening" can be used to remove x s from the tests developed in DALG-II. In deadening, an attempt is made to place at least two 0 s on the inputs of each AND (NAND) gate with a 0(1) output and at least two 1's on the inputs of each OR (NOR) gate with a 1 (0) output. This could be accomplished in a post precessor using an algorithm similar to consistency. The effect of deadening is to tend to make the given test insensitive to other failures that may also be present.

A strategy for combining DALG-II and TEST-DETECT which appears to be efficient involves using DALG-II to develop tests for primary inputs. Then the above three strategies are used to produce a set of tests, with no x;s, which can be run through TEST-DETECT to determine all the failures which are detected. When the D-chains used in DALG-II to develop these PI tests do not have points of reconvergence, TEST-DETECT will discover that, at the very least, all blocks on these chains are also tested. If any failures have not been detected, DALG-II is used to produce tests for them and the process repeated until all failures have been detected (or DALG-II returns information stating that no test exists).

The fault location problem can be approached by developing a fault location table [GNR, 64] with each failure as a row, each column as a Test-Response pair, and a mark at a row-column intersection if the test produces the corresponding response with the given failure. It is necessary to add tests to the set used for detection until each row has a unique pattern of marks. When circuits are packaged in modules, two rows may have the same pattern of marks if they correspond to failures in the same module.

3.5.4 Test-Response Algorithm. TEST-DETECT is set up (and programed) solely to determine which of a category $\underline{\underline{F}}$ of failures are detected by a given test T; no information is specified as to which outputs signal the presence of the failure, i.e., contain a D or $\overline{D}$. For purposes of diagnosis or failure location, as contrasted with detection alone, it is important to have the discrimination obtainable from knowledge of the actual response (cf. section 3.5.3). A small modification to TEST-DETECT can provide such response information.

In the present TEST-DETECT one has, for each test T and line l, a single symbol

$$FD(T,l) = \begin{cases} x \text{ (or blank)} & \text{if T does not detect a failure on line } l \\ \alpha (= 0 \text{ or } 1) & \text{if T detects the failure line } l \text{ stuck-at-}\alpha \end{cases}$$

The Test-Response algorithm would store a vector TR(T,l) of length ρPO with the ith entry containing the value of FD(T,l) for the circuit viewed as having only the ith primary output available for use in the TEST-DETECT algorithm. The modifications to TEST-DETECT to achieve this new algorithm are the following:

1. When the D-chain defined by DL reaches a primary output the FD computed for this PO is entered into the appropriate TR position <u>and the TEST-DETECT procedure continued until all branches of the D-chain "die out" (A becomes empty) or the primary outputs are reached.</u>
2. When the conditions of Lemma 3.1 are satisfied, DL $\equiv$ line j, the non-x entries in TR(T,j) are entered into TR(T,l).

Condition 1 insures that all outputs which signal the presence of the failure, not just the first one encountered, are discovered. The second condition merely expands the amount of information one can obtain from Lemma 3.1 when TR(T,l) is available.

3.6 Cyclic Circuit Diagnosis

This section is primarily concerned with showing how the previously specified D-calculus and acyclic circuit diagnosis algorithm, DALG-II, can be extended to develop a Cyclic circuit Diagnosis Algorithm, to be called CD-Algorithm I. A brief review of the present state-of-the-art in cyclic diagnosis and several examples illustrating how cyclic circuits are diagnosed by CD-Algorithm I are also included.

3.6.1 State-of-the-Art

The classical approach to cyclic or sequential circuit diagnosis [Hennie, 64] uses state diagram or state table models of the machine to be tested [Moore, 56; Huffman, 54; Mealy, 55]. Time sequences of inputs, derived from the state table, are sought which place the good machine in some known state. Then a checking sequence of inputs is developed which, when preceeded by the state synchronizing sequence, allows the outputs to distinguish the good and failed machines. A second, and more difficult approach is to seek a sequence of inputs which never allows the user toactually know what state the machine is in but does cause the good and failed machines to generate different output sequences (c.f. Example 1, Section 3.6.3).

There are three basic difficulties in applying these techniques in practice. First, in almost all computer designs the circuit is not specified in terms of classical state diagrams or tables and generating them proves to be an enormous computational task. Second, the classical algorithms to date work well for machines with a very small number of states, say less than ten, but encounter tremendous combinational problems and very long test sequences when practical circuits are tested; here practical implies more than a thousand states as in any circuit with only 10 independent bits of storage. Finally, it is impossible to guarantee finding a test for a circuit failure from a state diagram or table alone without exhausting all possible state transitions and input sequences; the actual circuit implementation must be known to avoid exhaustion. The algorithm to be developed here will use a logic block predecessor list description of the circuit and only require state table or state diagram specifications for relatively small subportions of the circuit (e.g. the latch configurations defined in section 3.6.2).

A different attack on cyclic circuit diagnosis which has found wide use in practice is the socalled SCAN technique [CMPR, 64,;

HS, 65]. Here enough extra inputs and outputs are added to the cyclic circuit to provide the ability to scan data into the storage elements of the circuit so that all states become known. Then the test is applied and the results scanned out for analysis. In effect, the circuit is rendered acyclic for diagnostic purposes. The main disadvantage of this technique has been its heavy use of extra input and output lines, something not appealing when pin-limited technologies are used, e.g. LSI.

In general, the problem of failures which produce new feedback loops have been avoided. Races and hazards, either already present in the model or introduced because the actual circuit doesn't obey the model, also prove not to be amenable to most diagnostic algorithms. In this connection it may be that the good circuit with proper inputs does not exhibit races or hazards but when the failure occurs or illegal (say bad parity) inputs are applied to it, such problems are manifested, e.g. a failure or illegal input could cause an RS flip-flop to receive both SET and RESET inputs causing, at best, an unknown output. The CD-Algorithm I to be developed also excludes failures which introduce new feedback loops and unspecified race conditions.

<u>3.6.2 The Synchronous Model</u> To develop a cyclic circuit diagnosis algorithm which will be an efficient useful tool, the nature

of the cyclic circuits actually being designed should be incorporated in the model upon which the algorithm is based. The majority of the circuitry is clocked synchronous logic. Synthesizing such circuits is straightforward, thereby allowing the average designer to carry out an implementation in a short period of time with a minimum number of errors. Since the structure of such circuits is readily apparent, engineering changes are easily executed on a local basis something far more difficult in classical asynchronous design. By controlling the transfer or time of propagation for data at well defined interfaces, the problem of correctly treating ill-defined component and wiring delays (which may not even be known until final layout) can be reduced to a local basis wherein it becomes more amenable. Control over data propagation will also be required in ultra-reliable computers to prevent the proliferation of erroneous data [c.f. Chapter 5]. The two major drawbacks of synchronous design are the slower function execution and the slight increase in circuitry over an asychronous implementation.

The typical synchronous circuit in Fig. 3.8 will be used in all subsequent examples in this chapter. Thus, it merits some discussion. Depending on the value of SR, either b_1, b_2 (SR = 1) or b_2, b_3 (SR = 0) are brought to points a_1, a_2. When control lines K_5 is set to 1 these values are then entered into the R-S flip-flops

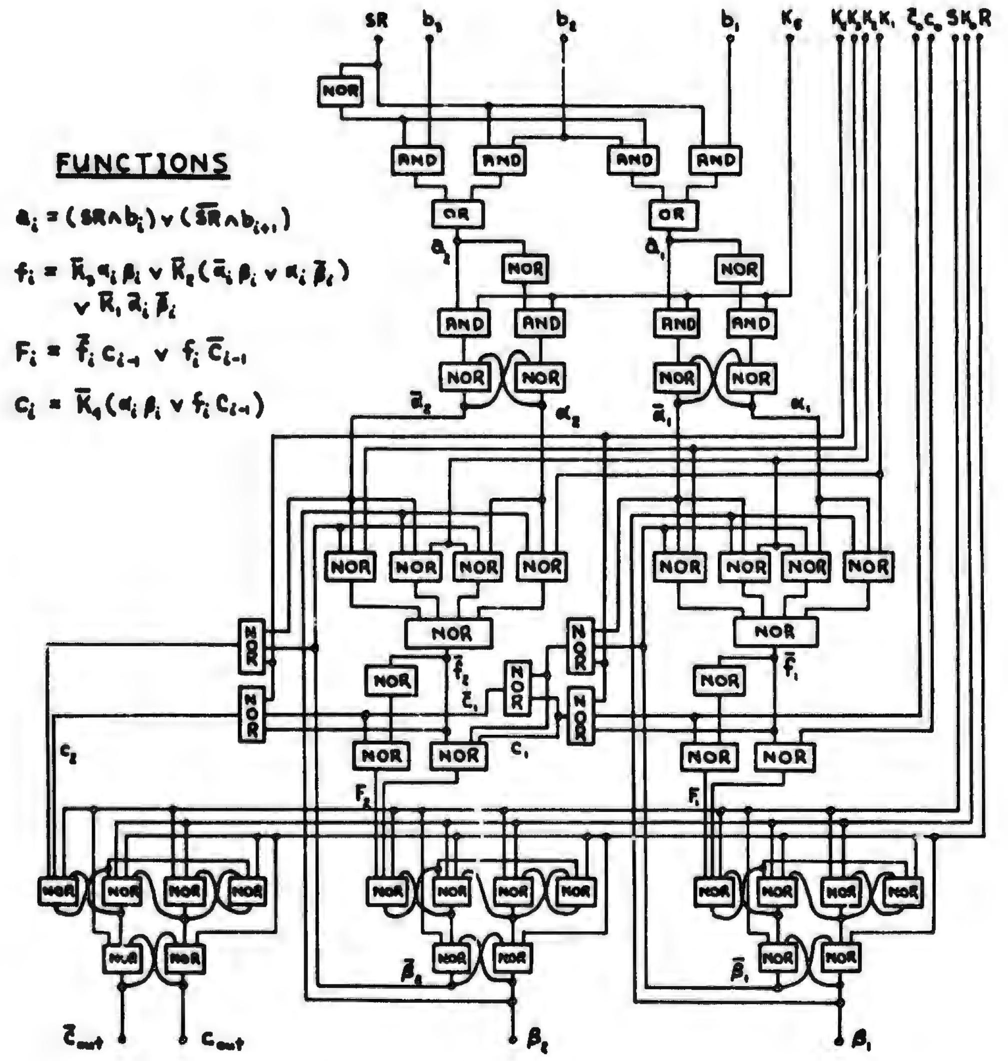

FUNCTIONS
$a_i = (SR \wedge b_i) \vee (\overline{SR} \wedge b_{i+1})$
$f_i = \overline{K}_3 \alpha_i \beta_i \vee K_2(\overline{\alpha}_i \beta_i \vee \alpha_i \overline{\beta}_i) \vee K_1 \overline{\alpha}_i \overline{\beta}_i$
$F_i = \overline{f}_i c_{i-1} \vee f_i \overline{c}_{i-1}$
$c_i = \overline{K}_4(\alpha_i \beta_i \vee f_i c_{i-1})$
SR
b_3
b_2
b_1
K_5
$K_4 K_3 K_2 K_1$
$\overline{c}_0 c_0$
SK_6
R
NOR
AND
OR
a_2
a_1
$\overline{\alpha}_2$
α_2
$\overline{\alpha}_1$
α_1
$\overline{f}_2$
$\overline{f}_1$
$\overline{c}_1$
c_1
c_2
F_2
F_1
$\overline{\beta}_2$
$\overline{\beta}_1$
$\overline{c}_{out}$
c_{out}
β_2
β_1

Figure 3.8 Typical Synchronous Cyclic Circuit: A Function Accumulator.

as α_1, α_2. The control lines K_1, K_2, K_3 permit f_i to become any one of 8 possible functions of α_i and β_i. The value of C_i is controlled by K_4; when $K_4 = 1$, $C_i = 0$; when $K_4 = 0$ and $f_i = \bar{\alpha}_i \beta_i \vee \alpha_i \bar{\beta}_i$, C_i is the carry out of the ith stage of an adder. The value of F_i can be made equal to f_i by setting the carry lines to 0 ($K_4 = 1$, $C_0 = ,0$) or to the exclusive -OR of f_i and C_{i-1}. When control K_0 drops from 1 to 0 the current value of f_i is entered into the output register, becoming the new β_i. Thus the circuit is a functional accumulator, one of whose functions is addition. Set and reset lines are also provided on the accumulator register.

Before developing the circuit model to be used in the CD-Algorithm I, several fundamental terms will be defined. Liberal use will also be made of the existing terminology in the classical literature.

The notion of what constitutes a test must be expanded, for cyclic circuits. A test T will now be defined to be a time sequence of primary input values $T = \{T^1, T^2, \ldots, T^n\}$ such that after the application of the complete sequence, the good and the failed machines have different primary output values. Superscripts will denote the time sequencing and subscripts will be used to denote distinguish different tests.

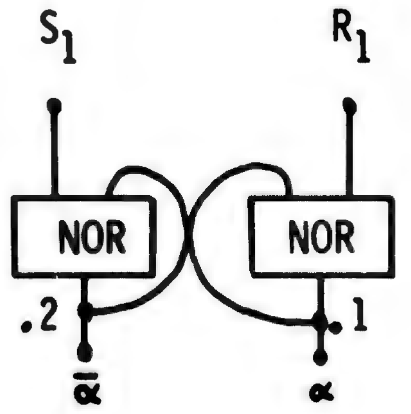
Consider a circuit composed of logic blocks interconnected to possess feedback loops such that information storage or memory is provided. Two types of control over this circuit's storage capability will be defined. We will say injection control exists if, by appropriate selection of primary input values, we can store known data into the feedback loops. Inhibition control exists when the primary inputs can be appropriately chosen so as to inhibit or block the propagation of data through the feedback loop at some point in the loop. For example, in the circuit of Fig. 3.8, inhibition control is provided over all feedback loops by primary inputs K_0 and K_5, the clock lines for the upper and lower registers respectively. Injection control for the upper register is obtained through the combinational logic trees defining a_1 and a_2, and for all other feedback loops by primary inputs S and R.

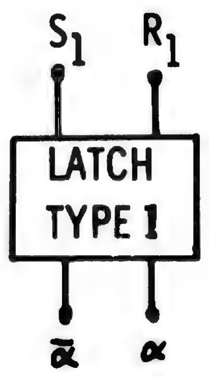
A latch configuration (LC) is an ensemble of logic blocks interconnected so as to possess information storage capability and a well specified transfer and state transition function. In order to give an LC the status of a functional storage "atom", it is customary to restrict the amount of circuitry being called an LC by requiring that it should not be partitionable into disjoint LC's which also have a well defined functional meaning within the confines of the circuit. At the same time the circuitry which facilitates effective injection and/or inhibition control is usually included in the definition of the LC. Thus, exactly what the designer chooses to call an LC is subjective, based more on functional intent than invariant constraints. Since the cyclic diagnosis algorithm to be developed next will view latches as indivisible functional units, just as DALG II viewed AND, NAND, etc., blocks, a complete functional specification (e.g. a state table) will be required for every LC. Keeping this specification compact is the major motivation for restricting the size of the LC's, to storage atoms.

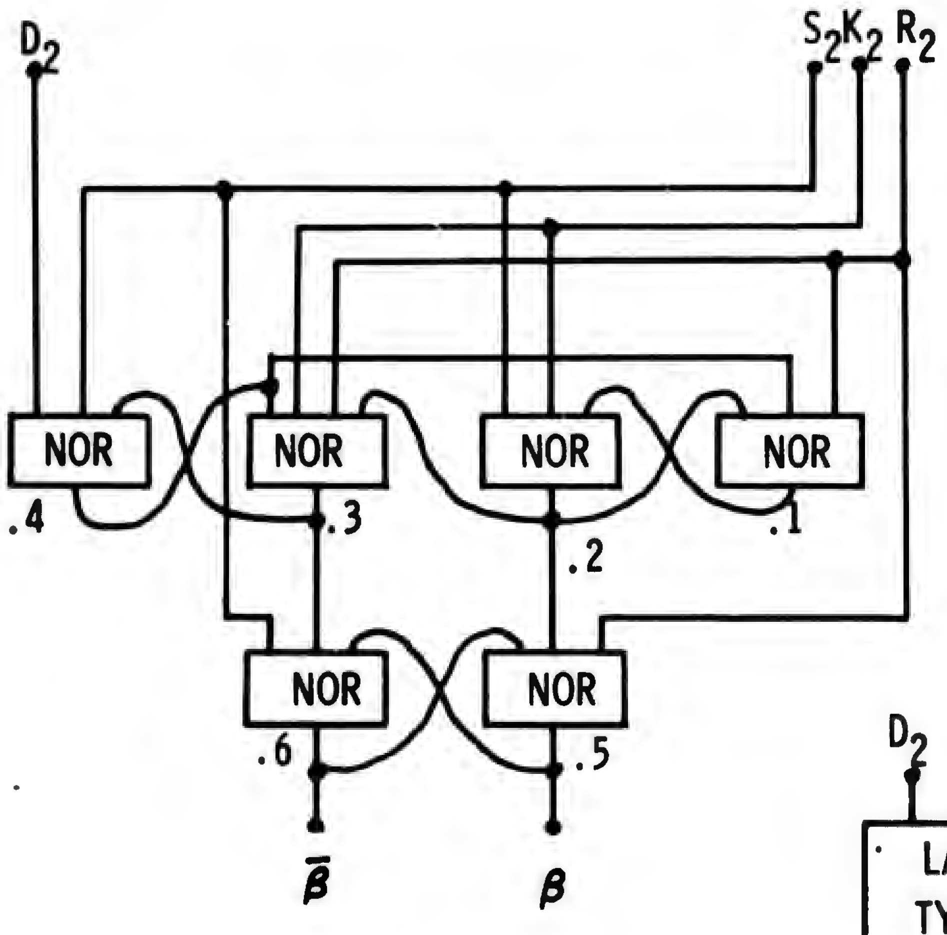
The two configurations shown in Fig. 3.9 are functionally meaningful LC's for the circuit of Fig. 3.8. Since it is possible to select the primary inputs so that S_1, R_1 input combinations of 0,1 , 1,0 and 0,0 can occur on all type 1 latches, both injection and inhibition control exists. The second latch type was defined to include a complete cluster of three interconnected RS flip-flops since it is at this level that the true functional characteristics appear: when $S_2 = 0$ and $R_2 = 0$, K_2 provides inhibition control by gating D_2[2] into the latch as K_2 drops from 1 to 0, and S_2 and R_2 provide injection control. Inserting these latch definitions into the circuitry results in the diagram of Fig. 3.10.

---

[2] In Fig. 3.9 the single data line D_2 can be replaced by a set of lines, with the result that the data gated into the latch is the OR of these signals.







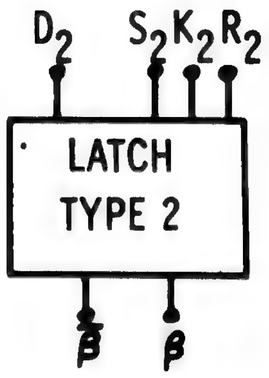


Fig. 3.9 Latch configurations for circuit of Fig. 3.8.

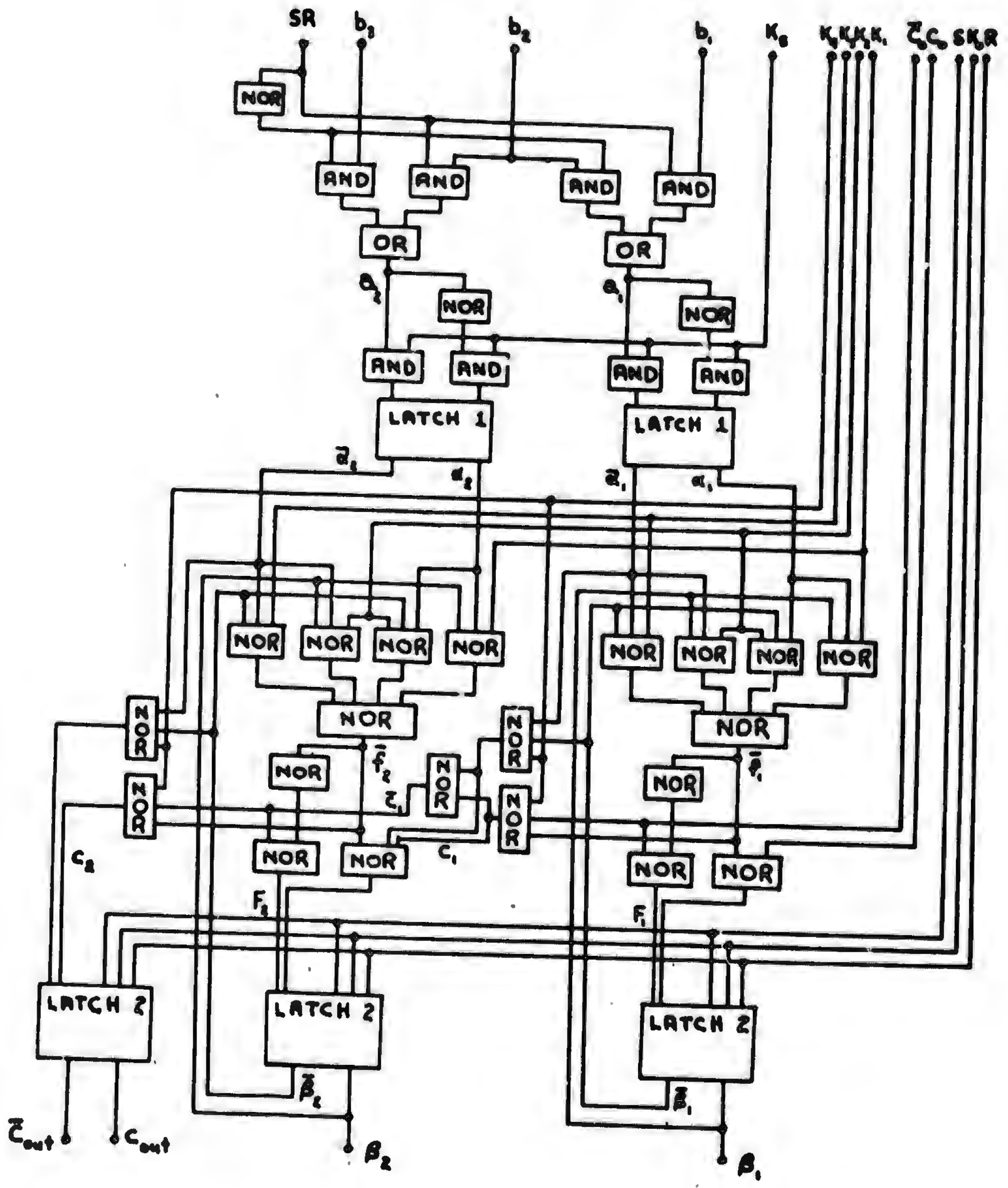


Figure 3.10 Function Accumulator with Latches Inserted.

It is relatively easy to provide the transfer function, state transtion table, pdc's, and pdcf's for these two latches: Table 3.2 illustrates the required quantities for latch type 1. The only new concepts entering the picture are that of next state variables, denoted with * superscripts, and time sequences of variables, denoted by a time occurance indication with $t_2 > t_1$. The last two rows in the transition table describe the latch operation subject to the normally forbidden input 1,1. By itself this input does not cause trouble, it only develops outputs which are not complements of each other, but when followed by a 0,0 input an unknown output results. The pdcf's are generated by locating a one or two step sequence of inputs which will force the output in the good and failed circuit to differ (as before D means 1 in the good circuit, 0 in the failed circuit). When testing either of the two outputs for a stuck-at-0 failure it is mandatory that an output of 0D or D0 be generated even though this means uncomplementary outputs will exist in the failed circuit.

The pdc's shown in Table 3.2c are the type which always present complementary values on the latch outputs, something the designer generally expected the circuit to have. In most cases these will be the only pdc's used. They are developed by placing combinations of 0's, D's and $\overline{D}$'s on the inputs of the latch and comput-

S	R	α	$\bar{\alpha}$	α^*	$\bar{\alpha}^*$
0	0	b	b	b	$\bar{b}$
0	1	x	x	0	1
1	0	x	x	1	0
1	1	x	x	0	0
0	0	b	b	x	x

b = 0 or 1

TRANSITION TABLE

(a)

	S	R	α	$\bar{\alpha}$	α^*	$\bar{\alpha}^*$	Time
α s-a-1	0	1	x	x	$\bar{D}$	D	t_1 or t_2
	0	0	$\bar{D}$	D	$\bar{D}$	D	t_2
$\bar{\alpha}$ s-a-1	1	0	x	x	D	$\bar{D}$	t_1 or t_2
	0	0	D	$\bar{D}$	D	$\bar{D}$	t_2
α s-a-0	1	0	x	x	D	0	t_1 or t_2
	0	0	D	0	D	$\bar{D}$	t_2
$\bar{\alpha}$ s-a-0	0	1	x	x	0	D	t_1 or t_2
	0	0	0	D	D	$\bar{D}$	t_2

(b) PDCFs

S	R	α	α^*
D	$\bar{D}$	x	D
$\bar{D}$	D	x	$\bar{D}$
D	0	0	D
$\bar{D}$	0	0	$\bar{D}$
0	D	0	0
0	$\bar{D}$	0	0
D	0	1	1
$\bar{D}$	0	1	1
0	D	1	$\bar{D}$
0	$\bar{D}$	1	D
D	0	D	D
$\bar{D}$	0	D	0
0	D	D	0
0	$\bar{D}$	D	D
D	0	$\bar{D}$	1
$\bar{D}$	0	$\bar{D}$	$\bar{D}$
0	D	$\bar{D}$	$\bar{D}$
0	$\bar{D}$	$\bar{D}$	0

(c) PDCs

S	R	α	$\bar{\alpha}$	α^*	$\bar{\alpha}^*$	
1	D	x	x	$\bar{D}$	0	t_1
0	D	$\bar{D}$	0	$\bar{D}$	D	t_2
0	0	$\bar{D}$	0	x	x	

(d) EXTENDED PDC

Table 3.2 Functional description of LATCH TYPE 1.

ing the next state for all possible present states. The first row of Table 3.2d is an example typical of a set of extended pdc's characterized by a possible 1,1 input in either the good or failed machine. If a 1D input occurs at the time t_1 the resulting latch outputs become $\bar{D}0$. If the next input, at time t_2, is 0D a stable output of $\bar{D}D$ is reached. However, if 00 is applied at t_2 the output becomes unknown. Inclusion of these extended pdc's has the advantage of extending the number and type of tests which can be developed. The disadvantages are that 1) an abnormal condition of identical latch outputs occurs, something which may cause subsequent difficulties in completing the test since the designer probably didn't expect this, 2) when the next input is applied the latch output becomes, at best, unknown and 3) the number of pdc's becomes very large [3], e.g. even with the normal pdc's in Table 3.1c one must now provide for the fact that the previous state latch outputs are not necessarily complements. In the algorithm described in section 3.6.4 it will be proposed that one first try obtaining a test using the normal pdc's and only when no test can be found with these, resort to the extended pdcf's.

The cyclic circuit diagnosis algorithm of the next section re-

[3] There exists a total of 166 pdc's for latch type 1 if extended pdc's are used.

quires that the circuit meet the following restrictions:

1. Each feedback loop contains at least one LC which has injection control.

2. Each feedback loop contains at least one LC which has inhibition control.

3. The state transition function, pdc's, pdcf's and all forbidden inputs are specified for each LC.

4. All forbidden circuit inputs, if any, are specified.

An optional requirement which would make any programmed version of the algorithm more efficient would be the prior specification of those lines which have the special status of "control lines", e.g. clock lines, latch pulse lines, etc.

Almost all synchronous computers meet restrictions 1 and 2 governing the ability to bring data into the loops and the prevention of uncontrolled data flow through them, a condition characteristic of races. The designer constructs his circuit using a relatively small number of basic LC's for which he has available well defined transition and transfer functions and a list of forbidden LC inputs. Using these functions, it is a simple, though time consuming, process to generate the pdcf and pdc tables. Fortunately this need only be done once, say when the LC is added to the library of basic logic modules available to the designer. Specification of all forbidden circuit inputs is required to prevent generation of a test which could never be applied in the actual machine, e.g. involve illegal sequencing of certain primary inputs. These restrictions will be used to determine inconsistencies during the algorithm execution.

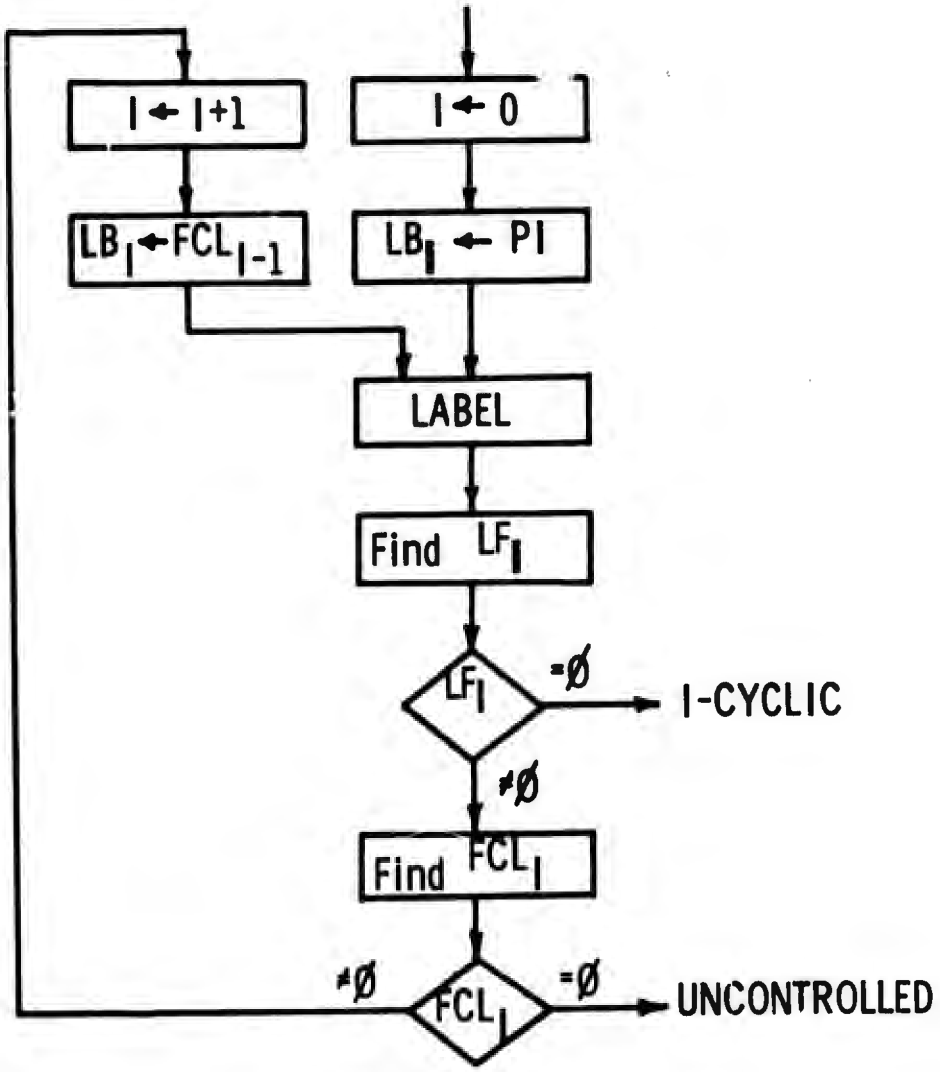
An algorithm , called RELABEL, for sequentially numbering the blocks in the circuit while simultaneously testing the circuit for satisfying restrictions 1 and 2, is outlined in Fig. 3.11. First the primary inputs are assigned integer labels from the set $\wr\rho$PI. These lines are then placed in the set of initially labelled logic blocks LB_0; LB_I is defined to be set of blocks labelled during the I th algorithm interation. Then the set LB_0 is expanded by the sub-algorithm LABEL which implements the following labelling condition:

"A logic block (line) can be labelled if, and only if, all its predecessors are labelled."

The I th label frontier, LF_I, is defined to be the set of all unlabelled blocks which have at least one labelled predecessor. If, after termination of the LABEL process for $I = 0$, LF_0 is empty the circuit is acyclic. When LF_I is not empty, all controllable latches in LF_I are located and placed in a set of Frontier Control Latches FCL_I. When FCL_I is empty the circuit does not meet the restrictions previously outlined and the circuit is rejected. Otherwise

FCL_I is labelled and placed in LB_{I+1}, I is incremented by 1, and the LABEL procedure reexecuted. The process proceeds until the whole circuit is labelled or determined not to be controllable.

With the current definition of control, i.e. controllable directly from the primary inputs through LB_0, when LF_1 is not empty FCL_1 is automatically empty. This somewhat parochial view of latch control was imposed to facilitate the implementation of the first cyclic diagnosis algorithm. If the definition of injection and inhibition control used in generating FCL_I is expanded from use of LB_0 to all of LB_0, LB_1, ---, LB_I, a larger class of circuits can be treated. An example of the topology of such a circuit is indicated in Fig. 3.12 where each block contains an indication of its status. Diagnosing such a circuit greatly increases the algorithms computational complexity since data can only be injected into the second loop by prior injection into the first loop; the combinations involved in this process are great.



LB_I - blocks labelled during I th iteration.

LF_I - labelled forntier of the labelled subgraph the I th iteration.

FCL_I - controlled latches contained in LF_I.

I-CYCLIC - exists a loop which is controlled through I-1 other loops by the PIs.

Fig. 3.11 RELABEL algorithm.

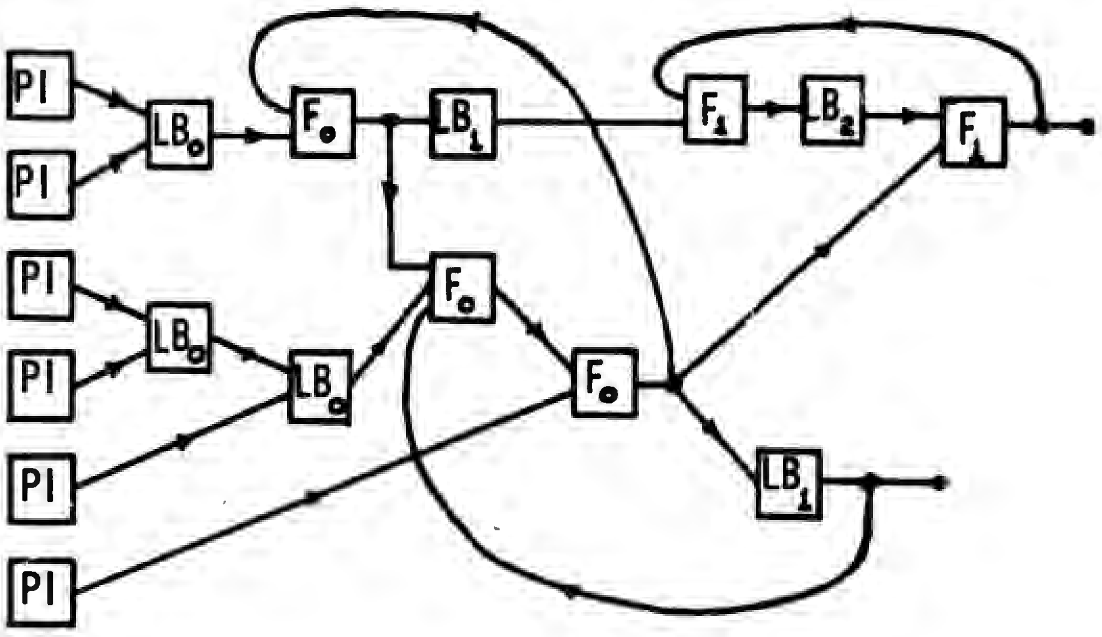


Fig. 3.12 Example of extended control in circuit.

Because of the present state, next state specification of the latch's functions, pdc, and pdcf, it will be necessary to introduce the concept of having successive time intervals or time frames, t_i, during which the test is developed. Corresponding to each such time frame t_i we associate a test cube TC^i. When a test is developed by CD-Algorithm I it will appear as the $TC^i[PI]$ values for the time sequence of TC^i developed in the algorithm.

<u>3.6.3 Examples</u> Three examples illustrating how DALG-II can be extended to diagnose circuits modelled as in the previous section will be presented. The basic attack will be to use DALG-II on all combinational circuitry (i.e., those logic blocks not in LC's) in an attempt to drive D's to the PO's or to a controlled latch's input (i.e. a block in FCL_0) where it can be inserted and stored for use in the next time interval.

When the numbering algorithm is applied to the circuit of Fig. 3.10 we obtain the renumbered circuit in Fig. 3.13. The sets which this algorithm developed, in terms of the new numbers, are:

$$
\begin{aligned}
&LB_0 \leftarrow 27\\
&F_o \leftarrow (27 + \lambda 13),\ 43,\ 44,\ 45,\ 46,\ 52,\ 53\\
&FCL_o \leftarrow 28\ \ 29\ \ 30\ \ 31\ \ 32\\
&LB_i \leftarrow 27 + \lambda 26\\
&F_1 \leftarrow \lambda 0
\end{aligned}
$$

The forbidden input combinations are those which set $TC[1,3]$ to 1 and those which do not put complementary values on $TC[4]$ and

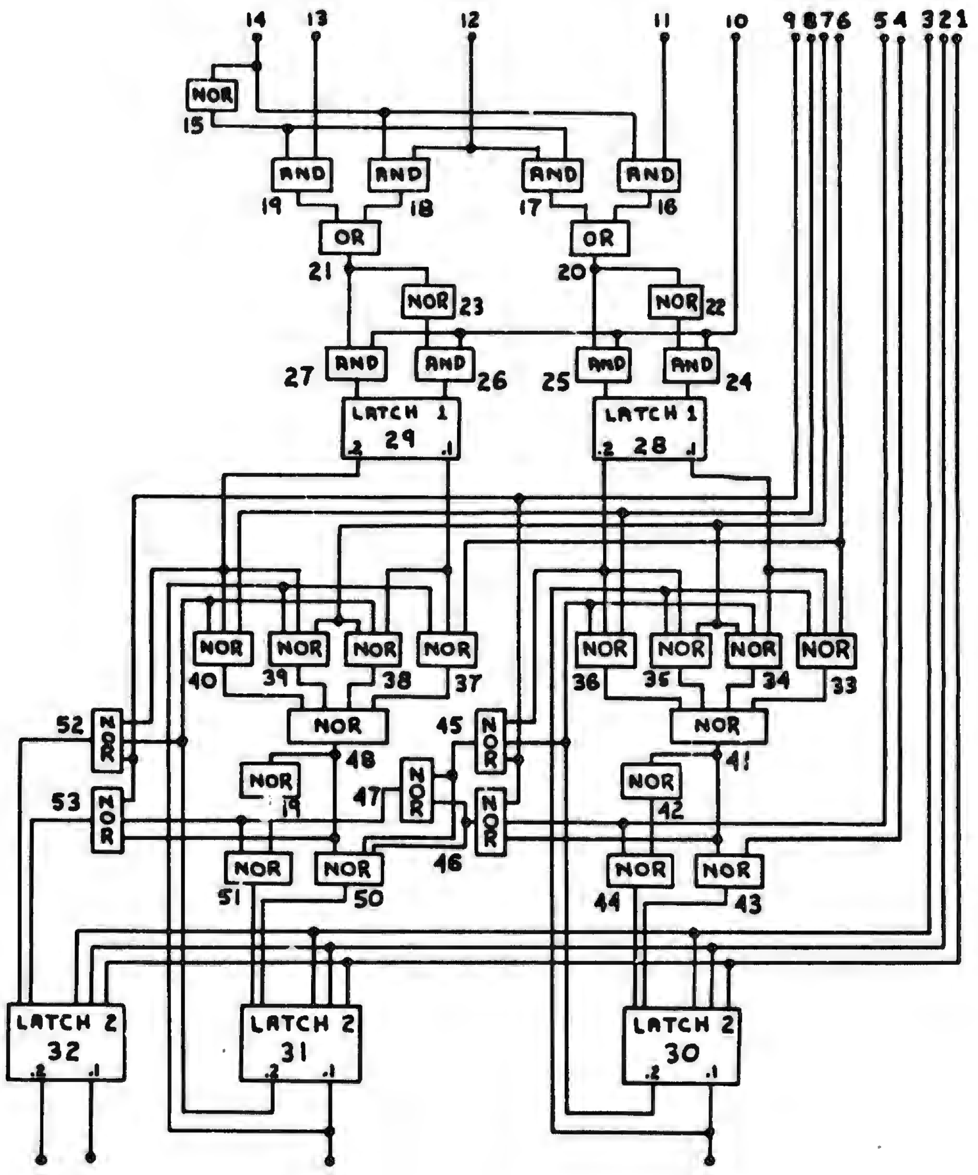


Figure 3.13 Relabeled Function Accumulator Circuit.

TC [5] in the good circuit.

For the first example consider the failure line 11 stuck-at-0. The decision sequence used in obtaining a test is shown in Table 3.3 Steps 1 through 6 illustrate the TC^1 assignments made due to pdc intersection and resulting implication as DALG-II extends the D chain from line 11 through 16, 24, 33 and 43. At step 5 the initial decision was to drive the D through block 35 (the lowest entry in the activity vector) but this caused the D-chain to "die" at block 41. Step 7 is used to inset the D into the latch. To do this requires a second time interval and thus a second test cube TC^2. Since the D has reached a primary output, no further D-drive by DALG-II is required on either TC^1 or TC^2. The only remaining task is to perform consistency on the TC^1[30.1] value of 0. Examination of the state table for latch type 2 shows this is accomplished by setting TC^1 [1,3] to 1,0 i.e. by resetting the latch. The test for this failure is

TC^1 [ι14] ←— 11001 01XX1 DXX1
TC^2 [ι14] ←— 000XX XXXXX XXXX

Next consider the failure line 28.1 stuck-at-0 (i.e. the α_1, output of latch 28 stuck-at-0) which results in the formation of the diagnostic test generation decision sequence in Table 3.4 The pdcf used in the first step is obtained from Table 3.2 The implications resulting from this pdcf are shown in the second part of step 1 and the remaining TC^1 assignments generated during the D-drive through blocks

1.	$TC'[11] \leftarrow D$	
2.	$TC'[14, 16, 15, 17, 19, 20, 22] \leftarrow 1, D, 0, 0, 0, D, \bar{D}$	
3.	$TC'[10, 24, 25, 28.1, 28.2] \leftarrow 1, \bar{D}, D, D, \bar{D}$	
4.	$TC'[6, 30.1, 33, 30.2, 34, 36, 45] \leftarrow 0, 0, \bar{D}, 1, 0, 0, 0$	
5.	$TC'[7, 35, 41, 42, 44] \leftarrow 0, D, 0, 1, 0$ $\left\{\begin{matrix}\text{D-CHAIN} \\ \text{DIES}\end{matrix}\right\}$	
5.'	$TC'[35, 41, 7, 38, 39, 42] \leftarrow 0, D, 1, 0, 0, \bar{D}$	
6.	$TC'[4, 43, 5, 44, 46, 47, 51, 53] \leftarrow 0, D, 1, 0, 0, 1, 0, 0$	
7.	$TC'[2] \leftarrow 1$	$TC^2[1, 2, 3, 30.1, 30.2]$ $\leftarrow 0, 0, 0, \bar{D}, D$
8.	$TC'[1, 3] \leftarrow 1, 0$	

Steps 1 - 6 : D-Drive by DALG-II.
Step 7 : Transfer of D into latch for use at next time interval by CD-Algorithm.
Step 8: Consistency.

Table 3.3 First Example of CD-Algorithm.

1.	$TC'[24, 25, 28.1, 28.2] \leftarrow 1, 0, D, 0$ (pdcf)	
	$TC'[10, 22, 20, 16, 17] \leftarrow 1, 1, 0, 0, 0$	
2.	$TC'[6, 30.1, 33, 30.2, 34, 36, 45] \leftarrow 0, 0, \bar{D}, 1, 0, 0, 0$	
3.	$TC'[35, 41, 7, 38, 39, 42] \leftarrow 0, D, 1, 0, 0, \bar{D}$	
4.	$TC'[4, 43, 5, 44, 46, 47, 51, 53] \leftarrow 0, \bar{D}, 1, 0, 0, 1, 0, 0$	
5.	$TC'[2] \leftarrow 1$	$TC^2[1, 2, 3, 30.1, 30.2] \leftarrow 0, 0, 0, \bar{D}, D$
6.	$TC'[11] \leftarrow 0$	
7.	$TC'[15, 14, 19] \leftarrow 0, 1, 0$	
8.	$TC'[1, 3] \leftarrow 1, 0$	

Steps 1 - 4 : D-Drive by DALG-II.
Step 5 : Transfer of D into latch for later use in CD-Algorithm.
Steps 6 - 8 : Consistency.

Table 3.4 Second example of CD-Algorithm.

1.	$TC^{'}[2, 43, 30.1, 30.2] \leftarrow 1, 1, 0, 1$ (pdcf) $TC^{'}[4, 41, 5, 34, 36, 42, 44, 45, 46, 47, 51, 53] \leftarrow 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0$	$TC^{2}[2, 43, 30.1, 30.2] \leftarrow \bar{D}, 1, D, \bar{D}$
2.	$TC^{'}[33, 6, 28.1, 28.2, 35, 25, 24, 10, 22, 20, 16, 17] \leftarrow 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0$	$TC^{2}[1, 3] \leftarrow 0, 0$
3.	$TC^{'}[11] \leftarrow 0$	
4.	$TC^{'}[15, 14, 19] \leftarrow 0, 1, 0$	
5.	$TC^{'}[1, 3] \leftarrow 1, 0$	

Step 1 : PDCF entry.
Steps 2 - 5 : Consistency.

Table 3.5 Third example of CD-Algorithm.

33, 41 and 43 are shown in steps 2, 3 and 4 respectively. At step 5, the D at the input of latch 30 is gated through into the next time interval, thereby fo.ming a new TC^2. Since the D has reached a primary output, only consistency on blocks 16, 17 and 30 need be performed; the results are as shown in steps 6-8 respectively. This completes the test.

The netwo-k formed by lines 11 through 21 is similar to those used in Chapter 5 for buss reconfiguration. It is interesting to note that in such applications, a further restriction might be placed on what is to constitute a legal test, i.e. at some times SR will always be a 1 and b_3 will be uncontrollable, and at other times SR will always be 0 and b_1 with then be uncontrollable. Problems now arise wh-n consistency is performed on blocks 16(which requires 11 or 14 to be zero) and 17 (which requires 12 to be 0 or 14 to be 1). The only way to meet both the input restrictions and the consistency requirement is to use $b_1 = 0$ when SR = 1 and $b_2 = 0$ when SR = 0 (b_1 is uncontrollable he e). Clearly the need to store more than one test depending on the state of the SR line is costly in both test generation time -nd test storage.

Finally, consider the case wh-re the latch control K_o (line 2) is stuck-at-1. The pdcf shown in step 1 of Table 3.5 spans two time intervals, and, in fact, places a D on a PO in TC^2 so that only consistency need be performed. The remaining steps illustrate the consistency decisions for blocks 16, 17 and 30 together with their implications. In this example it was the correct use of the pdcf which permitted diagnosis of a control line failure, something usually quite difficult to do.

Suppose that in the previous example no Set or Reset line (3 and 1 respectively) existed. Then it would be impossible to find a test using CD-Algorithm I even though one exists. The reason for this is that the only tests depend upon examining the latch 30 output in two successive time intervals during which the output must exhibit a change in value. What the two values are is not important and can, in fact, never be established. (One test requires that $F_i = \overline{\beta}_i$, which can be obtained by setting $c_{i-1} = 1$ and $f = \beta_i$ or $b_3b_2b_1K_5\overline{K}_3K_2K_1c_{i-1} = 1$, then a change from 1 to 0 in the control line is attempted. If the β_i output switches value the circuit is good; if not, it has failed.) This type of test is not within the confines of those obtainable through the D-calculus.

3.6.4 CD-Algorithm I With this background of definitions and examples it is now possible to state in more detail exactly how DALG-II is extended to develop the cyclic circuit diagnosis algorithm, CD-Algorithm I.

The first task in the algorithm is the numbering process for developing the circuit model as outlined in Fig. 3.11. One of the phases of this task is the location of the controllable latches in LF_0 which are to be used in forming FCL_0. This requires examining the state tables for the $LC \epsilon LF_0$ to find which latch input patterns, if any, are required to provide inhibition and/or injection control. Then a consistency operation is performed to see if the circuitry in LB_0 can yield an output to match this latch input pattern. If so, the LC is placed in FCL_0; if not it is left in LF_0. It is highly desirable to save the results of these computations for later use in CDALG-I since they can shorten many of the consistency operations.

The next task is the selection of the pdcf for the given failure. If the failure occurs in the combinational circuitry, the pdcf is selected as in DALG-II. Otherwise, the pdcf is selected from the list which is provided for the failing LC.

The major tasks performed by CD-Algorithm I are essentially those of DALG-II; drive the D-chain from the site of the failure to a primary output and verify that all line assignments are consistent. Again during both D-drive and consistency there will be places where decisions must be made as to which of several paths the algorithm should follow, e.g. when the activity or consistency vectors or pdcf list have more than one entry. As in DALG-II the order in which these paths are examined is a strategy which sets up a decision tree for the algorithm to follow. Upon encountering an inconsistency, CD-Algorithm I backs up the decision tree to the next highest mode having an unexplored path.

Perhaps the major innovation of CD-Algorithm I over DALG-II is that if a D cannot be driven to a PO and it is possible to drive it to the input of an LC wherein it can be stored, this stored D together with a new set of circuit signal values can be used in a new attempt to drive the D-chain to a PO. This brings up the prospect of having D's combining which actually arose from manifestations of the failure at different time intervals. The result is an enhancement of the types of tests which can be obtained. Unfortunately, a corresponding increase in the combinations is involved in obtaining the test results.

When it is not possible to reach a PO with the D-chain or store a D in a latch, CD-Algorithm I cannot find a test for the given failure.

Since every loop has injection control, we know it is always possible to enter data into the loops, for testing purposes from the primary outputs. The inhibition control which exists in each major loop usually exists because the synchronous machine has clock or control lines for gating signals at the interfaces. Inhibition control over the loops is used to prevent the uncontrolled looping of D's back onto the site of failure or the point of injection. It further serves to cut down the combinatorics by localizing the computations.

A whole new spectrum of strategies for setting up the algorithm's decision tree now arise. Most are based on the new concept of having multiple test cubes; one for each time interval.

1) It appears clear that if a test can be obtained using a D-chain

existing in only one time interval then this is most desirable. The problem is to store all the decision paths that lead to injecting a D into an LC as they are encountered. This will save massive recomputation when, at a later point in the algorithm, it is discovered that it is not possible to get a one time interval test, thus requiring D's to be "latched", i.e., stored in a latch.

2) In order to prevent an uncontrolled time segment growth or oscillations of the type where $TC^i = TC^j$ $i \neq j$, it is necessary to impose a restriction on the maximum number of time intervals which will be generated.

3) When a new time interval t_{i+1} and associated test cube TC^{i+1} are developed during the D-drive, a decision must be made to either continue the D-drive on TC^{i+1} or perform consistency on TC^i. It appears that D-driving TC^{i+1} is preferable since what is done here helps restrict what constitutes a permissible consistency in TC^i. However, this preference is not absolutely clear and must be subjected to experimentation.

4) When a test is initially sought without using the extended pdc's then, if no test was found,a second pass must be made using these new pdc's if one is to expand the domain of failures for which the CD-Algorithm can develop a test.

5) A new restriction on what constitutes a permissible test can

arise when it is known that each test must be exactly some specified length, e.g. if a four phase clock is used and the output is examined only at the fourth interval every test must be of length four; D's on the outputs at earlier times are lost.

6) Suppose tests for failures in the latches are obtained first. Then these tests must contain the information as to how a D is driven from a latch to the primary outputs. Now when tests are being sought for failures outside the latches and a D from the D-chain is stored into a given latch, it is possible to "look-up" how the D was driven from this latch to the PO's when its test was generated. If this method of reaching the PO is consistent with the failure outside the latch now being considered, the same input sequence can be used to complete this test, thereby saving large amounts of computation.

7) Assume that when the tests are actually applied to the hardware they are executed in some sequence one right after another. Then if test T_i tests the set of latch lines Λ_i, the values on these lines at the completion of the test T_i could be used to shorten the next test T_{i+1}, i.e. these values might help by not requiring special Sets and/or Resets to be given for the latches. The values T_i leaves on latches outside those actually tested are not available for use since they may not be correct.

The class of circuits which the CD-Algorithm I can diagnose is restricted to those which fit the control model; most synchronous circuits fall in this category. The restrictions are used strictly to circumvent the problem of having to know timing and delay information in a circuit which has races. It should be possible to run an extension of DALG-II on any race-free circuit. However, the consistency problem of finding a previous state and input pattern which can result in the current state would tend to be a heavy computational burden in asynchronous circuits.

Also restricted is the category of failures which can be tested, i.e. failures which introduce new feedback loops are, in general, not allowed. Tests which require examination of a sequence of outputs, as was mentioned in the last example of the previous section, are not allowed either.

4. Reliability / Evaluation

4.0 Summary

4.0.1 Need for Reliability Evaluation and Prediction

The ARC specifications (see section 5.1) require that the probability of the ARC functioning correctly for 10,000 hours be .997. A reliable comparable simplex commercial computer which is expected to fail four times a year will function correctly for only 6.59 hours with a probability of .997 (section 5.4.3). 'Standard' TMR'd computers appeared initially not to have sufficient survival time, and in section 5.4.3 this guess is verified. (The predicted survival time lies between 180 and 270 hours.) Forbes, et. al. [FRST, 65] have shown that an extension of reliability techniques is necessary if ultra-reliable computers are to be built. The question is, which mix of self-repair techniques will achieve a specified availability. A second related question is, which self-repair techniques will be best for a given application in the ultrareliable computer specification. Guidelines are needed for estimation and prediction. It must be ascertained that work on the project is proceeding in the right direction. Knowledge of what is required for any success (lower load estimates) is needed as well as upper load estimates - what was not to be done because of limitations.

Such questions can best be answered by constructing mathematical models and formulae which will show the relationships of various parameters to increased availability. These models can be used to produce quantitative estimates of reliability improvements. Only by performing this evaluation and prediction can stringent goals be met by an efficient, not over-designed, computer organization. To perform this efficiently, these models must be used in procedures which may be used interactively by the designers. Devising and properly using automated support procedures is the first task to be performed in design of such computers. The importance and necessity of these procedures can't be overemphasized. Human beings themselves cannot perform the amount of accurate, complex, detailed work required. The possibility of an error in a support program inducing a failure in the computer system is very real, and must be guarded against. However, this is much less than the probability of human error in design.

4.0.2 Brief Review of Previous Work

4.0.2.1 Use of Mathematical Statistics

As machines and electrical equipment became more complex, the problems of reliability (and overloading) were faced. Typically it was the telephone companies which first felt the problem seriously enough to call for predictions of reliability and overloading. Engineers tackled these problems empirically - and in general obscurity. Theoretical bases for what is now called queueing theory seem to begin with Khertichine [Khertichine, 32; BP, 65]. The mathematical description of renewal theory with applications to replacement problems began with Feller [Feller, 41] and is well described by him in [Feller, 57; Feller, 67].

In 1952 Davis [Davis, 52] published a paper presenting failure data and the results of several goodness-of-fit tests for various computing failure distributions. These data, and the engineering explorations, seemed to give a distinct edge tc the experimental distribution. In 1951 Epstein and Sobel [ES,53] began work in the field of life-testing, and provided further impetus for the exponential distribution. "The fundamental reason for the popularity of the exponential distribution and its widespread exploration in reliability work is that it leads to addition of failure rates and makes possible the compilation of design data in a single form" [BP, 65]. It should be emphasized, following Davis [Davis, 52], that this property follows naturally from the type of data considered, and the mathematical model is natural and a good fit, as well as mathematically convenient.

With the introduction of complex missile and space systems, interest in reliability and availability has risen. Reports have poured out, with an increasingly mathematical/statistical impact, and with less and less concern for engineering or direct mathematical modeling of physical processes.

4.0.2.2 Computer Reliability Work

Tube computers were unreliable, and as soon as the "gee-whiz" stage of computers was passed, reliability became a favorite topic. In those days, the idea of diagnostic programs were new, so many papers treated reliability with a diagnostic program title. The 1953 IRE Symposium [SYMP, 53] contains papers by J. P. Eckert, M. U. Nilkez, G. Estsin, L. R. Walters and N. L. Daggett describing their experiences, together with discussions from the floor. The 1953 EJCC [Eachus, '53] had a similar discussion, and Wheeler and Robertson [WR, 53] discussed the Illiac. The major problem was with tubes, and using computers to check and diagnose themselves. (Epstein and Sobel [ES, 53] were also studying tubes.) No mathematical techniques were used except MTBF, number of failures per week and simple definitions of availability.

The 1957 WJCC returned to this theme, [WJCC, 57] with A. W. Boldyriff, Willis, Wave, H. T. Glantz and B. K. Smith discussing actual experience with little mathematics and no prediction. However, H. J. Zoger [Zoger 57] discussed the evaluation of failure data (using [Feller, 57; preliminary edition] but not [Davis, 52]). Joan Rosenblatt [Rosenblatt, 57] presented an interesting paper in which she attempted to predict system performance in a detailed way from component failure data, showing many of the difficulties. Fein [Fein, '57] while talking about real-time computers introduced the term 'self-repair', used replacement of modules explicitly and discussed the need for error-detection, error-diagnosis, and error-repair.

Basic work in combining these unreliable components was published by von Neumann[vonNeumann,56] and Moore and Shannon [MS, 56]. This initiated the work on unit/component redundancy. This work was continued and resulted in the 1962 Symposium on Redundancy Techniques for Computing Systems held by the Office for Naval Research [Tryon, 62; GMR, 62; Kautz, 62; Kemp, 62; Mann, 62].

However, interest in computer reliability from an enginerring point of view began dying out with the introduction of the transistor and the performance of the TRADIC [Felker, 54]. Commercial reliability problems seemed solved.

Mathematically-oriented work continued [Flehinger, 58; Lewis, 64] and models derived from mathematical statistics were used in the analysis of idealized computers.

Lewis showed that, to the accuracy of the observed data, the exponential failure rate is valid.

The recent work with the complex systems for defense and space has shown that reliability and availability are once again problems. New work from an engineering viewpoint has begun; for example, three sessions at the First IEEE Computer Conference (1967) will be devoted to computer reliability.

4.0.3 Summary of Newly Developed Formulae

In order to determine the relationships between parameters of the ARC, mathematical models had to be devised which contained specific and valid representations of the parameters. It was necessary to be able to choose the amount of detail required, and the accurracy available.

To facilitate the discussion, the following notation has been adopted:

λ	-	failure rate during power-on.
μ	-	failure rate during power-off.
K	-	λ/μ (usually > 1).
m	-	number of spares initially available.
q	-	number of modules required to be operating at all times (power-on).
n	-	total number present $(m + q)$.
f	-	number of failures each unit can withstand before malfunctioning.
T	-	mission time.
c	-	failure coverage, or probability, that given a failure occurs, it is detected by the checking and/or diagnostic tests and that successful recovery occurs.
D	-	duty cycle or fraction of time the machine is required to be on.

The parameters which effect the form of the reliability equations are m, q, c, and f. In order to display the status of these parameters each reliability equation will be written as ${}^{f}_{c}R^{q}_{m}$. When any of the parameters q, c, or f are elided, convention will be used to establish the values $q = 1$, $c = 1$, and $f = 0$.

A new recursive formulation of an integral equation to express reliability with space modules with power turned off was devised. Solutions were found to express R^{q}_{m} for any λ, μ, m, q, and D in closed form. Then ${}_{c}R^{q}_{m}$ was proved to be expressable as a sum, with coefficients powers of c multiplied by binomial coefficients of ${}_{c}R^{q}_{j}$ for j varying between 0 and m. Approximations, valid for the region of interest, were found for ${}^{f}R^{q}_{m}$.

Dividing a multiplexed computer's unit into segments increases its reliability. New formulae were developed for such divisions. It was also shown that while division into equal parts is best, small deviations from equality produced smaller reductions in reliability increase. There must be explicit use of parameters and an interaction between the designer, his design and the mathematical model. A program, REL, using these models ${}^{f}_{c}R^{q}_{m}$, with explicit entrance of parameters, has been written in APL (A Programming Language derived from the Iverson Notation) and run on on-line consoles by the system designers. Methods of using REL are described in this chapter (and results in chapter 5).

In order to give general guidelines for desirable self-repair techniques, analyses were run of several well-known techniques.

1) It was proved that for $\lambda T \leq .1$, and equal 1, using two spares gave greater reliability than TMR as long as the extra switching required for two spares was less than 248.3% of the total original equipment.

2) It was proved that it is always worthwhile for reliability (as well as power consumption) to turn unused spares off. If $K = \lambda / \mu$ then for $m = 2$ half the ultimate improvement is reliability is reached by $K = 1.7$ and 3/4 by $K = 3$. Experimental measurements seem to show K varying between 5 and 10. In the case of m spares the ultimate reliability

$$R_m(\lambda, \mu\ ) \rightarrow 1 - \frac{1 - R_m(\lambda, \lambda)}{m!}$$

3) It is shown that if the coverage decreases from 1 to .9, then the length of survival time is decreased from T to values varying from .35T to .75T. The fewer the spares the greater the effect.

4) If X circuitry has reliability such that its probability of not malfunctioning for 10,000 hours is .997, then by using two spare modules, it is shown that .85X circuitry will satisfy the same requirements if c=1. As $c \rightarrow .9$, the amount of circuitry available drops to .42X.

5) The reliability ${}_cR_2$ and the reliability of TMR'd computers were compared. If $c \geq .9$, then for $\lambda T > .02$, ${}_cR_2$ is greater and for $\lambda T < .02$, the maximum difference of reliability values is 8×10^{-5}.

If $c \geq .9$, ${}_{c}R_{2}$ is greater for $\lambda T \geq .003$, and for $\lambda T < .003$ the maximum difference of reliability values is 5×10^{-6}.

Methods of deriving models for Markov processes using state transfer methods were derived. These methods are simple to apply to reliability situations. They were used to aid in doing the models for ${}^{f}R_{m}^{q}$ with $f \geq 1$.

4.1 Introduction

The major problem to be considered is evaluating the ability to sustain information processing capability, i.e. providing the ability to maintain a certain minimum system information handling function which is capable of doing the critical task (defined by the user) in the presence of malfunctions. A malfunction may cause data or procedures to be destroyed as well as incapacitating, temporarily or permanently, the ability of part of the system to process information. This section will consider preparing the hardware design techniques necessary to overcome loss of function. Information contamination will be considered in a later phase.

For correct continuation in the presence of malfunctions, the following system functions are required:

1) Detection of malfunctions
2) Neutralization of the effects of the propagation of erroneous signals
3) Localization of the malfunctioning element
4) Determination of the relative time of occurrence of the malfunction
5) Bypass, by repair or masking, the malfunctioning element.

These functions may be fulfilled by the introduction of protective redundancy* into the system. A computer system contains protective redundancy if the effects of component malfunctions can be tolerated because of the use of additional components or algorithms, or the use of more time for information processing. These additional components,

* This term was introduced by A. Avizienis, [Avizienis, 66].

algorithms, and time are not required by the system in order to execute the specified tasks as long as no failures or transient malfunctions occur. The techniques of protective redundancy may be divided into two major categories: masking (sometimes called massive) redundancy and stand-by (sometimes called selective) redundancy.

In the masking redundancy approach the system functions necessary for correct continuation in the presence of malfunctions are performed by the instantaneous masking of the effect of malfunctions by permanently connected and concurrently operating replicas of the unit one of whose elements is malfunctioning. The level at which replication occurs merges from individual circuit components to entire systems. The principal classical techniques of masking redundancy are:

1) Replication of circuit components; eg "quadded" logical elements [Tryon, 62].
2) Replication of logic signals ; eg use of multiple channels and voting elements[MS, 56] or use of error correcting codes [Peterson, 61].
3) Replication of entire systems with comparison and voting or diagnosis at system level; eg the classical comparison of unit output (originally UNIVAC I).

In the stand-by redundancy approach, all of the system functions for continuation must be individually performed. The principal classical techniques of stand-by redundancy are:

1) Replacement of the faulty element or unit by a stand-by spare (self-repair)[Fein, 57, Craveling, 56].

2) Reorganization of the system into a different computer configuration; eg multiprocessors.

The use of these two methods presupposes that in addition to the circuitry and procedures to carry out the continuation functions there are switches and status registers to implement the replacement or reconfiguration and keep track of the present configuration.

In the general comparison of stand-by and masking redundancy which follows, the basic theme is hardware versus procedures or hardware versus time.

The advantages of the masking redundancy approach are the following:

1) The presence of concurrently operating replicas means that the system functions for continuation are performed by hardware and the system runs without interruption until catastrophic breakdown. Carrying out the continuation functions in the stand-by case takes time, complex procedures and careful design.
2) During normal operation there is no need for diagnosis. After breakdown, however, reconstruction and repair are very difficult unless still more hardware has been added. Effective diagnosis is a problem in any system.
3) All parts of the system are equally protected, there is no "hard core" which must work to perform the switching and reorganization functions.
4) The conversion of a non-redundant design to a masking redundant one is relatively straightforward. The design of stand-by redundancy demands novel design techniques.

The advantages of the stand-by redundancy approach are the following:

1) Power requirements are less because only one copy of each replaceable unit requires power at a time.
2) The replacement switch provides fault isolation between subsystems. Such isolation is essential for recovery after catastrophic or multiple failures.
3) Masking redundancy assumes independent single failures, which are not probable if the failures are externally induced. Moreover, unless additional hardware is added transient failures (4 to 10 times as frequent as solid failures) will disrupt masking redundancy recovery far more than stand-by redundancy recovery.
4) Theoretically, all spares can be utilized in stand-by redundancy, only a majority in masking redundancy.
5) The designs of individual units, and their number, may be more easily altered in stand-by than massive redundancy.
6) Masking redundancy makes premission checkout and "burn-in" more difficult. The system must be exercised until all components have passed their initial high failure rate.
7) The replication in masking redundancy tends to increase fan-out and fan-in requirements which decrease circuit tolerance and increase power requirements.
8) The failure rate for components in the stand-by units is less than the failure rate in the units with power on.

4.2 Evaluation Techniques

As the requirements for reliability become more stringent, deciding upon the best design parameters becomes more difficult. In order to do this, mathematical models must be devised for each of the types of redundancy. The relationship of each unit to the others needs to be known and predictable. There must be explicit use of parameters and an interaction between the designer, his design and the mathematical model. This has been done by developing a series of mathematical formulations which can be applied to different designs. A program, REL, using these models, with explicit entrance of parameters, has been written in APL (A Programming Language derived from the Iverson Notation) and run on on-line consoles by the system designers. In this section (and associated appendices), the mathematical models will be discussed and derived, REL will be described and results of the parameter studies shown.

As an aside, this is one of the basic automated support procedures needed for self-repairing computer design. Devising and properly using such support procedures is the first task to be performed in design of such computers. The importance and necessity of these procedures can't be overemphasized. Human beings themselves cannot perform the amount of accurate, complex, detailed work required. The possibility of an error in a support program inducing a failure in the computer system is very real, and must be guarded against. However, this is much less than the probability of human error in design - see section 5.4.7

4.3 Mathematical Models

4.3.1 Power On

This set of models should give a general set of guidelines. It is assumed that all systems are put through a rigorous system test long enough so that all components pass the period of rapid initial failure and most cold soldered points and poor connections are found (shock and vibration tests are a necessity). Under these assumptions, the exponential failure distribution (Poisson) for components and units can be assumed. In addition, for masking redundancy the voters will be assumed to be distributed so that the amount of equipment between voters is approximately equal (following the paper by Gurzi [Gurzi, 65]. Because failures in the voting circuits only produce errors in the driven channel, voter failures are included in with those in driven channel. The extra checking circuitry and switching within the stand-by units which perform equivalent functions cannot be disregarded when comparisons are made.

With these assumptions, the reliability of a single unit which malfunctions after a single failure is given by

$$R = e^{-\lambda T}$$

where $1/\lambda$ is the mean time between failure. The reliability of a triple modular redundancy system [von Neumann, 56] is,

4.1) $$RTMR = e^{-2\lambda T}(3-2e^{-\lambda T})$$

An improvement proposed recently consists of adding detection circuits on the multiplexed output lines, and when a unit is in the minority discarding the failing unit and one of the other units. This is called the TMR/Simplex system. The reliability RTS of this system may be calculated using the methods due to B. J. Flehinger [Flehinger, 58]. RTS is equal to the reliability with no unit failures (three good) plus the integral from 0 to T of the probability that the surviving unit will last from time t to time T.

4.2) $$RTS = e^{-3\lambda T} + \int_0^T \frac{\partial}{\partial t}(1-e^{-3\lambda t})\, e^{-\lambda(T-t)}$$
$$= e^{-3\lambda T} + 3e^{-\lambda T}\int_0^T \lambda e^{-2\lambda t}dt$$
$$= 3/2\, e^{-\lambda T} - 1/2\, e^{-3\lambda T}$$

In general, if $2N+1$ units are used in a voting circuit,

$$RV(2N+1) = \sum_{i=0}^{N} \binom{2N+1}{i} R^{2N+1-i} (1-R)^i$$
$$= R^{N+1} \sum_{i=0}^{N} \binom{N+i}{i} (1-R)^i$$

as proved in Appendix 2.1.2.

The behavior of the reliability estimates for λT small enough so that it is valid to approximate $e^{-\lambda T}$ by a truncated power series is indicative of more precise results. In this case,

4.3) $$RTMR \approx 1 - 3(\lambda T)^2$$

4.4) $$RTS \approx 1 - 3/2(\lambda T)^2$$
$$RV(2N+1) \approx 1 - \binom{2N+1}{N}(\lambda T)^{N+1}$$

These approximations approach the actual values from below.

In addition, if $\lambda T \neq 0$,

$$RTS - RTMR = 3/2\, e^{-\lambda T} (1-e^{-\lambda T})^2 > 0$$

$$RTS - R = 1/2\, e^{-\lambda T} (1-e^{-2\lambda T}) > 0$$

$RTMR - R = (1-e^{-\lambda T})(2e^{-\lambda T}-1) > 0$ only when

$e^{-\lambda T} > 1/2$, or $\lambda T < .68$.

In the case of M stand-by spares (M + 1 units) assuming that all units have the same failure rates, the reliability is given by

$$RS(M) = 1 - (1 - e^{-\lambda T})^{M+1}$$

$$\simeq 1 - (\lambda T)^{M+1}$$

The improvement in reliability by adding one stand-by unit at the $(M-1)^{st}$ stage is shown by

$$RS(M) = RS(M-1) + e^{-\lambda T}(1-e^{-\lambda T})^{M}$$

We have investigated a method for healing and sparing which uses four replicas of logic, four voters and triplication of lines. The mode of operation is TMR using three circuit units. When a circuit unit fails, the spare one is cyclically switched in. In this case the approximation becomes

$$R_{QTMR} = 6R_S^2 - 8R_S^3 + 3R_S^4 \cong 1 - 4(\lambda T)^3$$

where $R_S = e^{-\lambda T}$ is the reliability of a single unit.

If there are n units of logic, then the same techniques can be used, and

4.5) $R_{nTMR} = 1 - (1-R_S)^{n-1}(1+(n-1)R_S)$

4.6) $R_{nTMR} \simeq 1 - n(\lambda T)^{n-1}$

4.3.2 Power On or Off

These models for reliability using spares employ equations which assume that the spares deteriorate as rapidly as the units being used. This is too pessimistic, so a more realistic equation which employs two failure rates; λ, the failure rate during power-on use and μ , the failure rate during power-off idleness will be developed. To facilitate the discussion, the following notation has been adopted:

λ	-	failure rate during power-on.
μ	-	failure rate during power-off.
K	-	λ/μ (usually >1).
m	-	number of spares initially available.
q	-	number of modules required to be operating at all times (power-on).
n	-	total number present $(m+q)$.

f - number of failures each unit can withstand before malfunctioning.

T - mission time.

c - failure coverage, or probability, that given a failure occurs, it is detected by the checking and/or diagnostic tests and that successful recovery occurs.

D - duty cycle or fraction of time the machine is required to be on.

The parameters which effect the <u>form</u> of the reliability equations are m, q, c, and f. In order to display the status of these parameters each reliability equation will be written as ${}^{f}_{c}R^{q}_{m}$. When any of the parameters q, c, or f are elided, convention will be used to establish the values $q = 1$, $c = 1$, and $f = 0$.

A recursive formulation can be given to derive formulae for

R^{q}_{m} and ${}_{c}R^{q}_{m}$.

$$R_0(T) = e^{-\lambda T}$$

The probability of successful operation with n spares during the time interval $0 \leq t \leq T$ is equal to the probability of successful operation with (n-1) spares during the time interval plus the probability of a failure in the (n-1) spare system at time $t < T$ multiplied by the probability that the nth spare lasts from t to T. Let $R_m(t)$ be the probability of successful operation with m spares, and assume an exponential failure rate. Then the probability of successful operation of a single unit is $e^{-\lambda t}$ and the probability of successful shelf life is $e^{-\mu t}$. The formulation may be written mathematically for fixed λ, μ as:

$$R_m(T) = R_{m-1}(T) + \int_0^T e^{-\mu\tau} e^{-\lambda(T-\tau)} \frac{d}{d\tau}(1-R_{m-1}(\tau))d\tau$$

$$= R_{m-1}(T) - e^{-\lambda T}\int_0^T \frac{dR_{m-1}(\tau)}{d\tau} e^{(\lambda-\mu)\tau} d\tau$$

Its general solution is

4.7) $$R_m^q(T) = e^{-q\lambda T} \{1 + \sum_{i=1}^{m} [\frac{(1-e^{-\mu T})^i}{i} \prod_{j=0}^{i-1} (j+qK)]\}$$

as proved in Appendix 2.2, equation 2.2. If the effect of error coverage, c, is inserted into this formulation, the equations become

$${}_cR_0(T) = e^{-\lambda T}$$

and

$${}_cR_m(T) = {}_cR_{m-1}(T) + \int_0^T e^{-\mu t} e^{-\lambda(T-t)} c \frac{d}{dt}[1 - {}_cR_{m-1}(t)]\,dt.$$

The most concise form for stating the general solution is in terms of $R_i^q(t)$:

4.8) $${}_cR_m^q = \sum_{i=0}^{m} \binom{m}{i} c^i (1-c)^{m-i} R_i^q.$$

This formula is proved in Appendix 2.3, equation 2.14.

The improvement in $R_m^q(T)$ by adding one stand-by unit at the (M-1)st stage is given by

$$R_{m+1}^q(T)=R_m^q(T) + e^{-q\lambda T}(1-e^{-\mu T})^m \frac{qK}{m} \prod_{j=1}^{m-1} (1+\frac{qK}{j})$$

For purposes of comparative analysis it is useful to have approximations which are valid for small λ T. One is

$$4.9) \quad R_m^q(T) \simeq 1 - \frac{(q\lambda T)^{m+1}}{(m+1)!} \prod_{i=1}^{m} (1 + \frac{i}{qK})$$

This formula is proved in Appendix 2. 2. 3, equation 2.1.1 and the remainder term in the series expansion is shown to be positive..

If the coverage is considered, then 4.8) shows that terms in the power series are contributed by R_j^q, for $0 \leq j \leq m$. Equation 4.9) shows that these contributions begin with the (j+ 1)st term. In Appendix 2.2 methods are given for deriving approximation equations, and the actual approximations are derived for m = 1, 2, and 3. A typical expression may be written for m = 2 as

$$4.10) \quad {}_cR_2^q \simeq 1-b_{21}\, q\lambda T - b_{22}(q\lambda T)^2 - b_{23}(q\lambda T)^3 + b_{24}(q\lambda T)^4$$

where all the b_{2i} are positive quantities for $0 \leq c \leq 1$. In fact,

$$b_{21} = (1-c)^2, \; 2b_{22} = (1-c)\,(1-c(3+ 2\sigma/q)),$$

$$6b_{23} = (1-c)^2 - 2c(1-c)\,(1+ \sigma/q)\,(1+ 1/q) + [c^2(1+ \sigma/q)\,(1+ 2\sigma/q)]$$

$$24b_{24} = 6b_{23} + 2c(1+ \sigma/q)\,(1/q)\,\, [c(1+ 2\sigma/q) - (1-c)/q]$$

As c approaches 1, the b_{2j} approach the values in equation 4.9). A numerical study of these approximations and coefficients will be discussed later.

Since the failure rate, λ and the mission length, T, only occur as a product, halving one means the other must be doubled to maintain the same reliability and this is true regardless of the type of multiplexing employed.

As a measure of availability improvement we have taken the ratio of mission lengths attainable at constant reliability. For simplex systems this factor is also the ratio of failure rates.

4.3.3 Segmentation

It is well known [Gurzi, 65] that dividing a TMR computer into approximately equal parts and voting after each part gives increased reliability. This is also true foı duplex, triplex, and higher multiplexed computer systems. Since it is not practicable to divide a real computer into parts that are exactly equal (or rather, that have the same failure rates) we investigated the effects of asymmetric division. If a computer system is divided into two parts, of size a and b then the failure rates of the two parts are $\frac{a}{a+b}\lambda$ and $\frac{b}{a+b}\lambda$. For simplex systems

$$R = e^{-\frac{a}{a+b}\lambda T} \times e^{\frac{b}{a+b}\lambda T} = e^{-\lambda T} \cong 1 - \lambda T$$

and no improvement results. For duplex systems on the other hand,

$$R \doteq [1-(\frac{a}{a+b}\lambda T)^2] \times [1-(\frac{b}{a+b}\lambda T)^2] \doteq 1-(\lambda T)^2 \frac{a^2+b^2}{(a+b)^2}$$

and the improvement ratio is

$$I = \frac{a+b}{(a^2+b^2)^{1/2}}$$

This improvement attains a maximum value of $\sqrt{2}$ when a = b; i.e., when the two parts are equal. If the parts are unequal, the improvement factor varies slightly for a duplex system, and more strongly for a higher multiplexed system. The variation is not sufficient to impose a severe restriction on the computer design.

For a system with m spares,

$$R \simeq 1 - (\lambda T)^{m+1}$$

and the improvement ratio is

$$I = \frac{a+b}{(a^{m+1}+b^{m+1})^{1/m+1}}$$

with maximum value of $2^{\frac{m}{m+1}}$ when the two parts are equal.

If a unit is divided into S segments, then λ is replaced by λ/S in the basic reliability calculation (using equations 4.7 or 4.8) and this reliability is then raised to the S^{th} power.

In general, for m spares and S segments, using equation 4.9),

$$4.11)\quad R_m^q(T) \simeq 1 - \frac{(q\lambda T)^{m+1}}{S^{m+1}(m+1)!} \prod_{i=1}^{m} \left(1+\frac{iG}{q}\right)^5$$

The improvement factor due to this dividing becomes

4.12) $$I_n^q \approx S^{\frac{m}{m+1}}$$

If the coverage is c, and equation 4.10) is used,

4.13) $$_cR_m^q(T) \approx \left[1-\left(\frac{q\lambda T}{S}\right)\sum_{i=0}^{m}\left[\binom{m}{i}c^i(1-c)^{m-i}\frac{(q\lambda T)^i}{S^i(i+1)}\left(\prod_{j=1}^{i}\left(1+\frac{i}{q}G\right)\right)\right]\right.$$

If K is the first term so that the coefficient of $(q\lambda T)^K$ is not negligible $(0 \leq K \leq m)$, then, using 4.12), the improvement factor is

4.14) $$_cI_m^q \approx S^{\frac{K}{K+1}}$$

4.3.4 Duty Cycle

When the computer is to be run under a duty cycle which has a periodicity much much less than the mission time T, it is possible, as a very good approximation, to replace λ in all of the previous equations by

$$\lambda\left[D+\frac{(1-D)}{K}\right].$$

4.4 Computational Evaluation

In order to guide the design of an ultra-reliable computer system, the formulae developed in the last section were used in special cases.

4.4.1 TMR vs. Two Spares

TMR was until recently considered a 'standard' for reliable computers. A first calculation was made to determine under what conditions the reliability using two spares is superior to the TMR reliability or the reliability of a single unit. It was assumed that the failure rate of systems using spares would be $r\lambda$, $r \geq 1$ since extra checking and switching circuitry would be incorporated. The voting circuitry was disregarded. If R_s is the reliability of the single unit and R_T is the reliability of the TMR unit, then $F = R_2^1 - \max(R_T, R_S)$ is a function of λT, r, and λ/μ. It can be shown that $\frac{\partial F}{\partial r} < 0$, $\frac{\partial F}{\partial(\lambda/\mu)} > 0$, so if $F(r, \lambda/\mu, \lambda t)$ is greater than zero for $0 \leq \lambda t \leq \lambda T$, then F is greater than zero for all smaller values of r and larger ratios of λ/μ. Because the failure rate of passive components (connections, resistors), is unaffected by shelf life whereas the failure rate of active components is affected, μ was computed by assuming that x% of the unit circuitry had a failure rate λ/ρ, $\rho > 1$, and the remainder had a failure rate of λ. The results are shown on Graph 4.1. Each curve is labeled with values for (r, ρ). Clearly the length of the interval $0 \leq \lambda t \leq \lambda T$ increases most rapidly with a decrease in r. If the amount of checking circuitry can be kept small the potential rewards are great. As a maximum, if $r \leq 3.483$ (i.e., the added circuitry is 2.483 times as much as the original for each unit, 10.449 times as much total equipment), then F is greater than zero for $0 \leq \lambda t \leq .1$. This shows the superiority of the sparing technique assuming perfect error detection and recovery.

4.4.2 Effect of Variation of λ/μ on R_m^1

Since sparing appears to be the most desirable approach for ultra reliability, the effect of variations in λ/μ on values of R_m^1 should be evaluated.

As proved in Appendix 2.2.2, equation 2.11

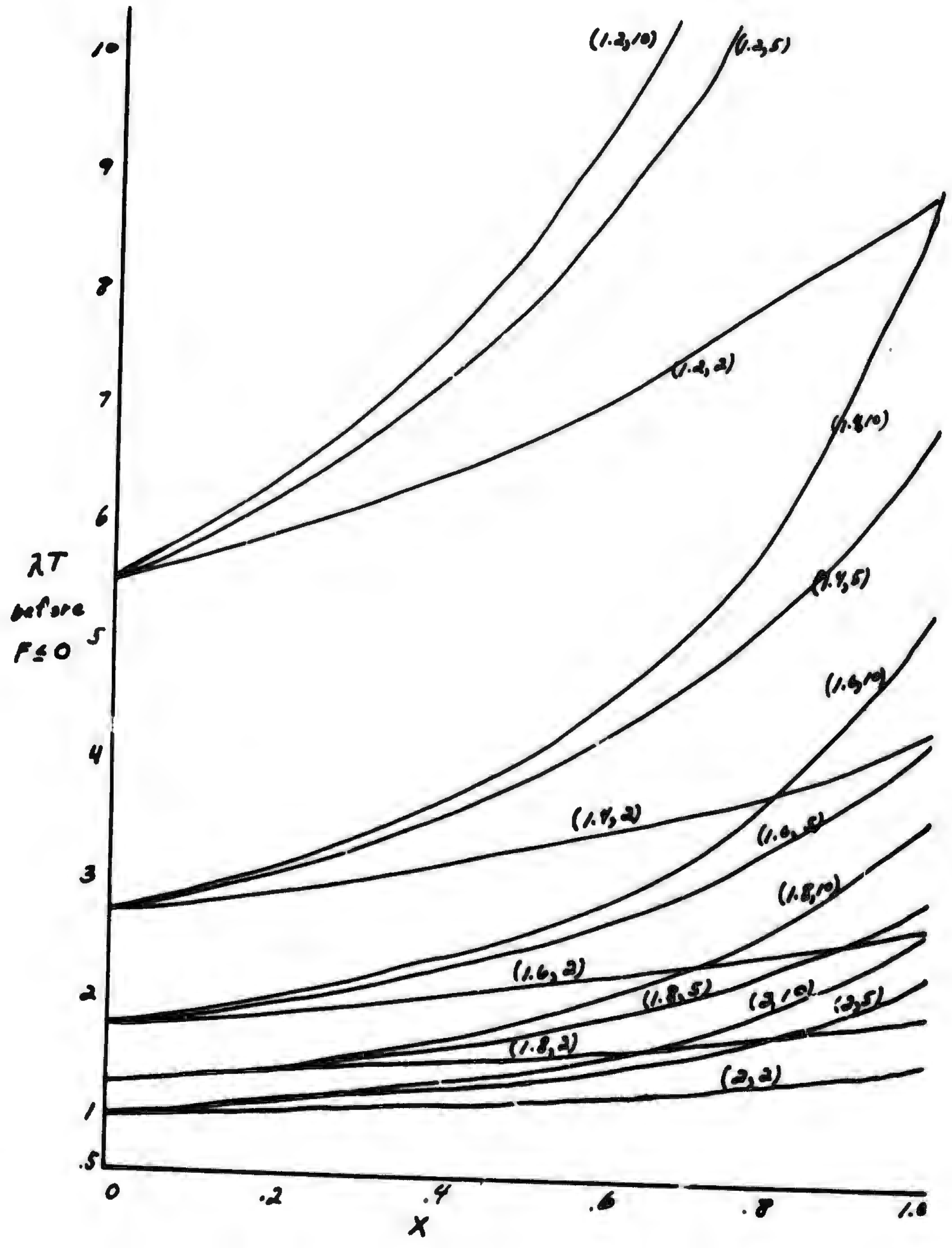


Figure 4.1 R_2^1 Comparison with R_T.

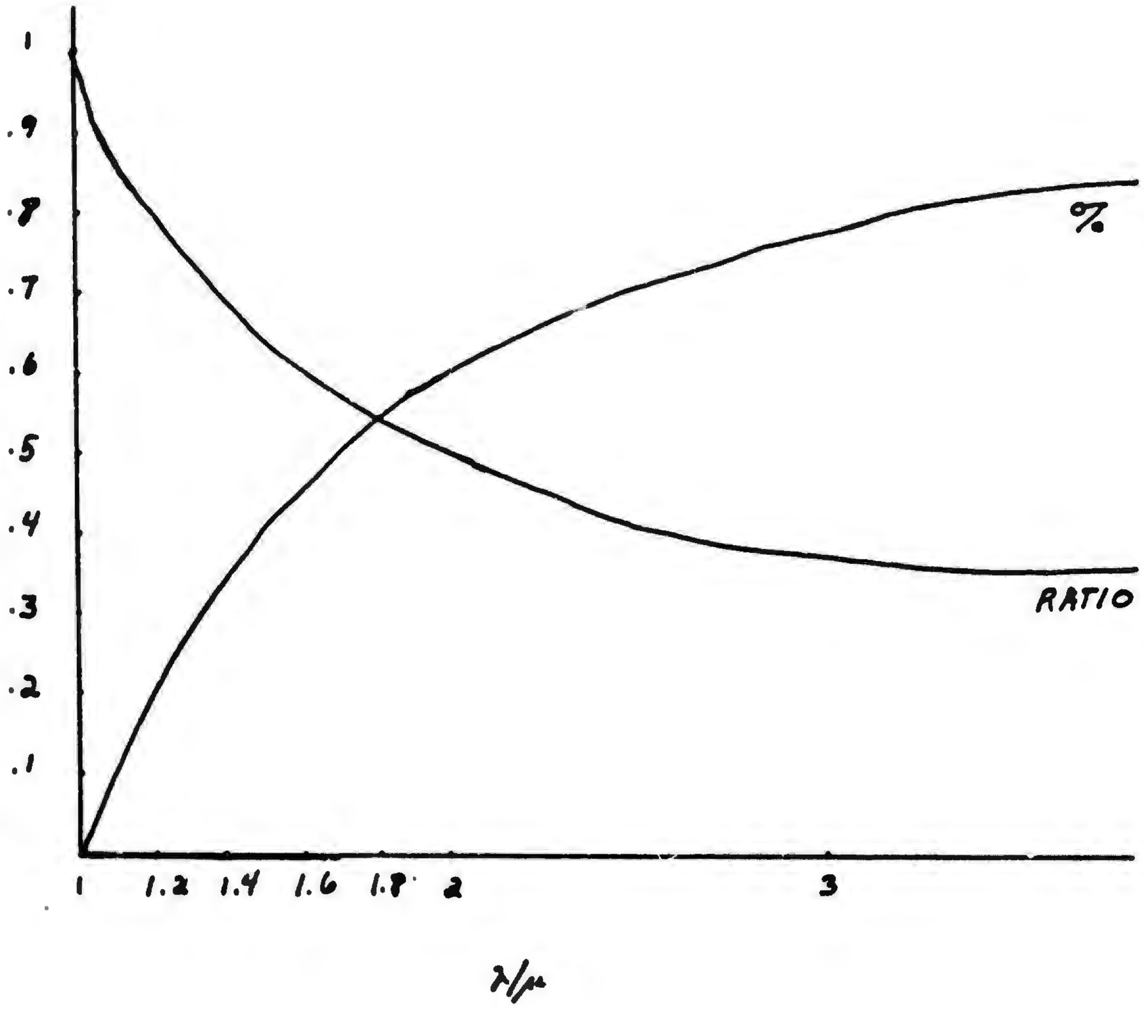


Figure 4.2 R_2^1 Variation with λ/μ.

$$R_n^1(\lambda,\mu,t) = 1 - \frac{(\lambda t)^{n+1}}{(m+1)} (1 + 1/\lambda/\mu)(1+ 2/\lambda/\mu)\ldots$$

$$(1+ n/\lambda/\mu) + \ldots$$

thus

$$\frac{1 - R_n^1(\lambda,\mu,t)}{1 - R_n^1(\lambda,\lambda,t)} = \frac{1}{(n+1)} (1+1/\lambda/\mu)(1+2/\lambda/\mu)\ldots(1+n/\lambda/\mu)$$

$$+ \text{ higher order terms}$$

$$\frac{1}{(n+1)} \text{ as } \lambda/\mu \text{ increases.}$$

The extra reliability due to the lower failure rate for unused spares can be seen by plotting

$$\frac{1 - R_2^1(\lambda,\mu,t)}{1 - R_2^1(\lambda,\lambda,t)}$$

This value is labeled 'Ratio' in Graph 4.2.

From the formula the extra reliability is a function of λ/μ. For design purposes it is useful to determine the percentage of the total improvement which has been reached by a given value of λ/μ. This graph is labeled '%' in Graph 4.2

Another set of calculations were made to determine the effect of sparing and the ratio λ/μ on the length of missions with a prescribed probability of success. The results are shown in Table 4.1. The results show that the higher the desired probability of success, the more benefit is obtained from additional spares. Less benefit is obtained from having the spare elements have a slower rate of failure but once again the most benefit is derived when trying to design to a higher specified probability of

Length of Mission with Assigned Probability of Success R_o

		λ/μ					
R_o	# spares	1	2	5	10	100	1000
.997	1	2816.4	3248.7	3630.6	3791.7	3956.9	3974.6
	2	7787.3	9769.4	11826.9	12810.8	13911.6	14036.2
	3	13330.9	17678.0	22734.9	25403.2	28631.3	29015.3
.99	1	5268.0	6070.9	6781.9	7082.4	7390.8	7424.0
	2	12131.8	15183.2	18358.8	19881.1	21587.1	21780.4
	3	19006.5	25112.1	32228.4	35992.6	40556.9	41100.8
.9	1	19006.5	21790.7	24291.6	25358.2	26458.6	26577.2
	2	31195.8	38634.0	46473.5	50271.0	54559.9	55048.2
	3	41315.7	53804.2	68197.9	76349.0	85956.2	87107.8

Normalized Length of Mission with Assigned Probability of Success

		Ratio	λ/μ					
R_o	# spares		1	2	5	10	100	1000
.997	1	1	1	1.153	1.289	1.346	1.404	1.[illegible]
	2	2.76	1	1.254	1.518	1.645	1.786	1.6[illegible]
	3	4.73	1	1.325	1.705	1.905	2.117	2.17[illegible]
.99	1	1	1	1.152	1.287	1.344	1.402	1.[illegible]
	2	2.36	1	1.251	1.513	1.633	1.779	1.7[illegible]
	3	3.61	1	1.321	1.695	1.893	2.133	2.1[illegible]
.9	1	1	1	1.146	1.278	1.334	1.392	1.3[illegible]
	2	1.64	1	1.238	1.489	1.611	1.748	1.7[illegible]
	3	2.17	1	1.302	1.657	1.817	2.030	2.1[illegible]

Table 4.1

success. As these figures show, the percentage of improvement in mission length increases more slowly than reliability as λ/μ increases.

4.4.3 Effects of Coverage, c, on ${}_cR_m^q$

The equation

$$ {}_cR_m^q = \sum_{i=0}^{m} \binom{m}{i} c^i (1-c)^{m-i} R_i^q . $$

shows that as the coverage c decreases, the reliability decreases. In fact, as $c \to 0$, ${}_cR_m^q \to R_0^q$. The effect of c can be seen by comparing these approximate expressions for the two spare case

$$ R_2 = 1 - \frac{(\lambda T)^3}{3!} (1 + \frac{1}{K}) (1 + \frac{2}{K}) $$

$$ {}_cR_2 = 1 - (1-c)^2 \lambda T - 2c(1-c) \frac{(\lambda T)^2}{2!} (1 + \frac{1}{K}) $$
$$ - c^2 \frac{(\lambda T)^3}{3!} (1 + \frac{1}{K}) (1 + \frac{2}{K}) $$

The equation for ${}_cR_2$ contains terms involving λT and $(\lambda T)^2$, something we tried to avoid by going to a two-spare system. Figures 4.3 and 4.4 are plots of

$$ \frac{1 - {}_cR_2}{1 - R_2} \quad \text{vs. } c $$

which show just how much and how fast the probability of failure increases as the failure coverage c decreases from 1. As to be expected, these curves show that when the modules are small (small λT) or when the off-time failure rate is small (large K)

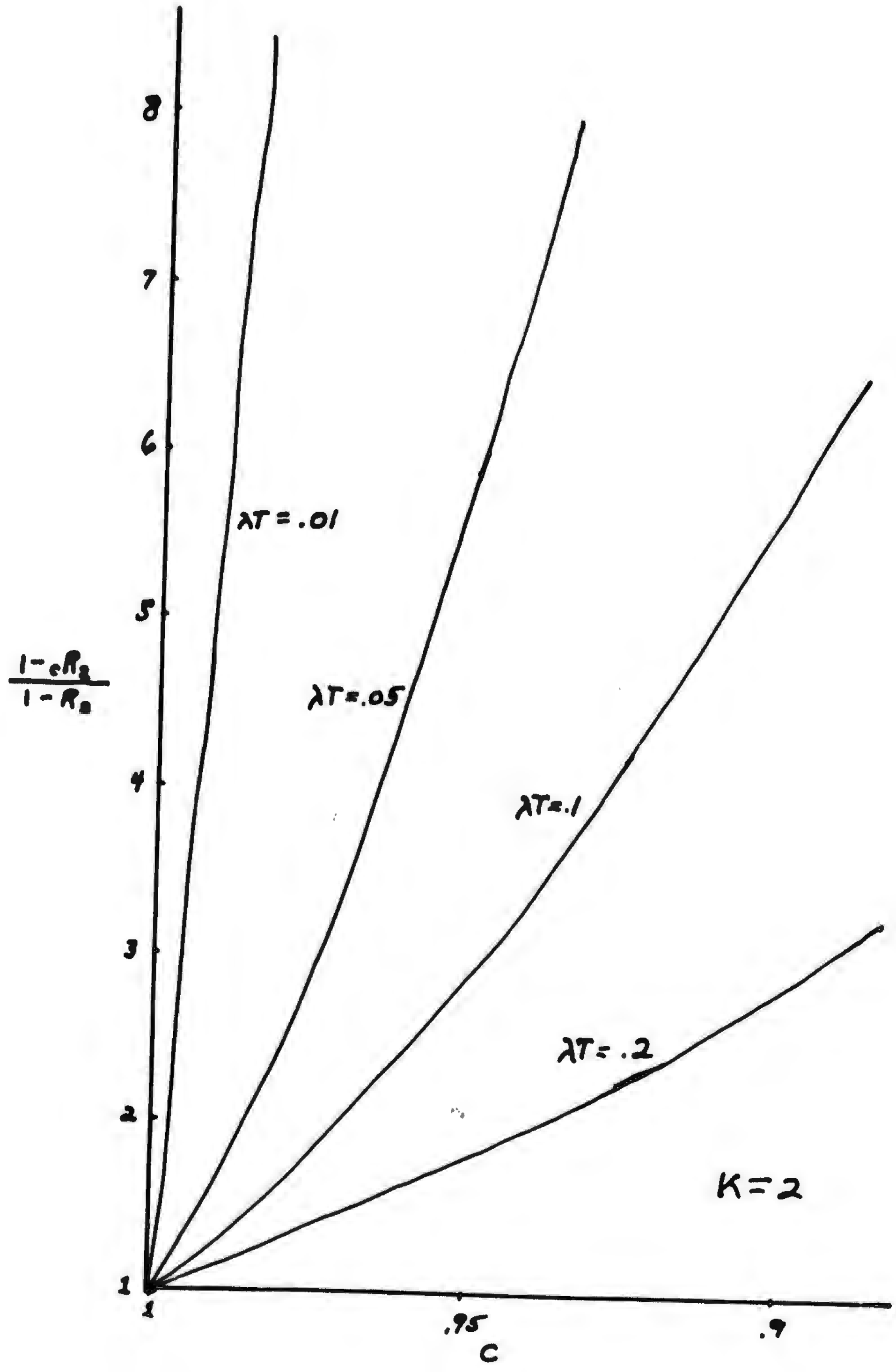


Figure 4.3 Hardware Increase by Triplexing

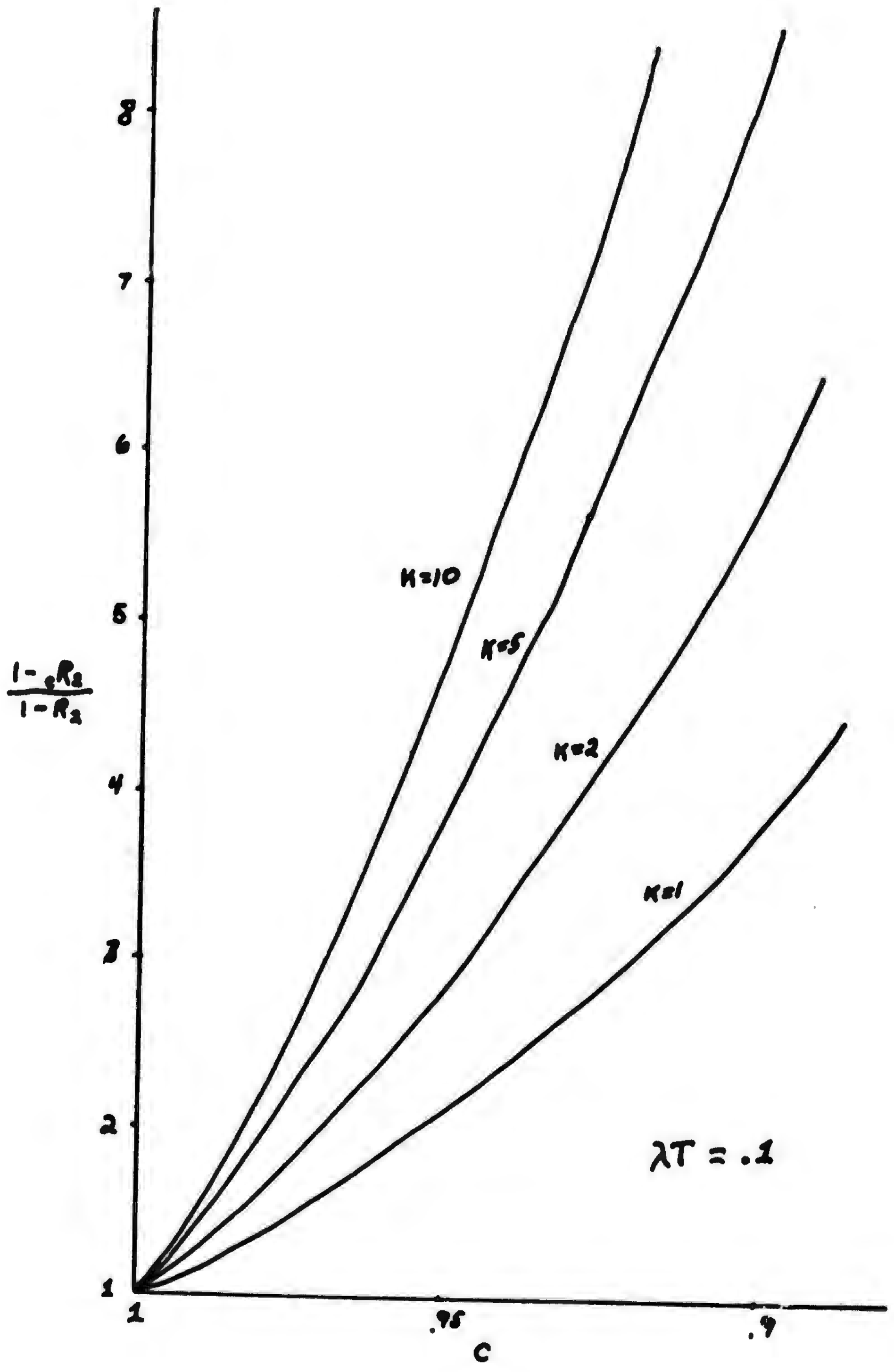


Figure 4. 4 Effect of Coverage on ${}_cR_2(\lambda T = .1)$ for Various K.

the relative change in reliability is greatest.

The general expansion formulae 2.16-2.18 in Appendix 2.2.3 supply further results.

It is easily seen that if $c > 0.9$, then the coefficients b_{mj} of all powers of $(q\lambda T)^j$ for $1 \leq j \leq m+1$ have the same sign, and the coefficients of $(q\lambda T)^{m+1}$ and $(q\lambda T)^{m+2}$ have opposite signs. This says that, since the rest of the terms in the series have alternating signs, using the first $m+1$ terms in the series for ${}_cR_m^q$ gives a reasonable approximation, with errors less than the first discarded term (with $q\lambda T$ raised to the $(m+2)^{nd}$ power). The effect of c on mission times is a reasonable way of obtaining its effect in practical cases. In Figure 4.5 the ratio of mission times is plotted for c varying between .9 and 1.0. The triplets which identify the graphs are (m,q,G). It is clear that the number of spares is most important, and that the effect of poor coverage is most impressive for the case of one spare. The reasons for this can be seen from Figure 4.6 in which the coefficients b_{mj} defined in equations 16-18 are plotted. As the coverage increases, the coefficients of terms of powers of $q\lambda T$ less than $m+1$ approach zero, but the $(m+1)^{st}$ term increases. If all terms are small - as in the case of 3 spares - the variance of ${}_cR_3^q$ will be less with c.

4.4.4 Amount of Hardware Available for use if Stand-By Redundancy is used to Achieve Reliability

The first comparison considers the following question. Assume a specific reliability (.997) is desired. This can be met by using a certain amount of hardware in a simplex fashion; i.e., $\lambda T \approx .003$). However, this amount of circuitry will certainly not be enough to construct a good computer system. How much more

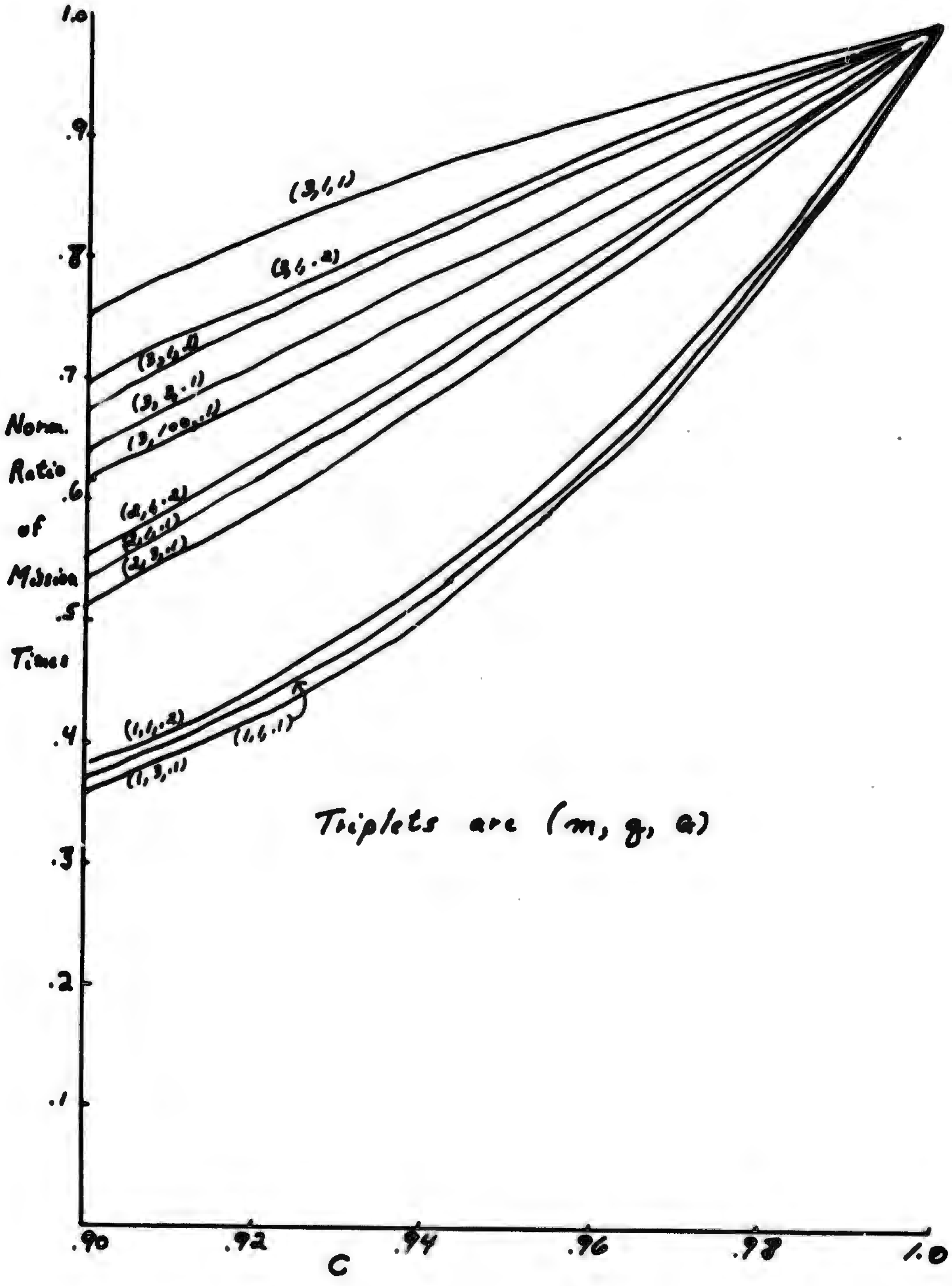


Figure 4.5 Mission Times vs. Coverage.

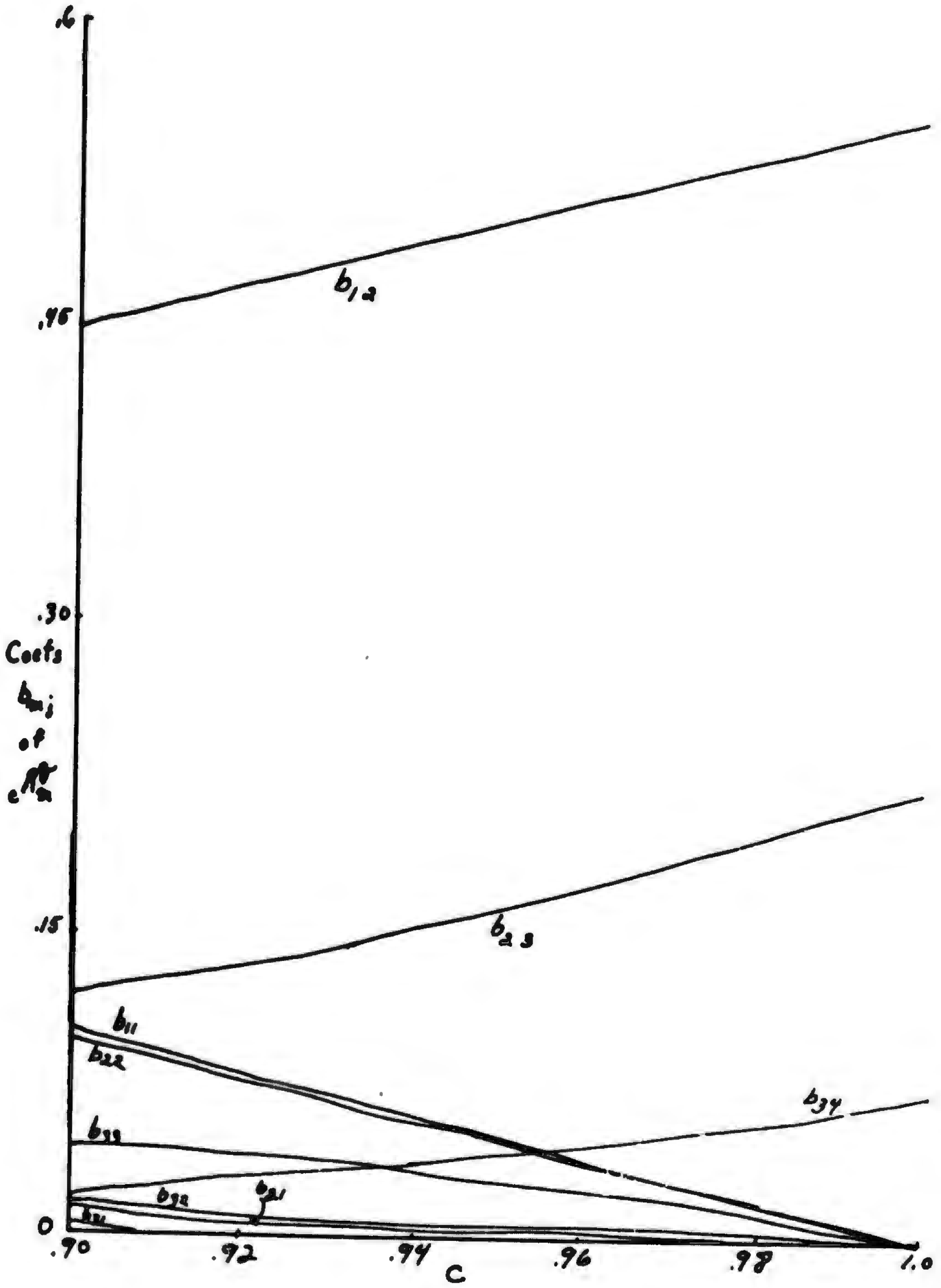


Figure 4.6 Equation Coefficients vs. Coverage.

circuitry will be available if the modules are duplexed or triplexed? Let each module have DEL more equipment than the basic system, so that $(\lambda T)^{\text{module}} = (.003) \times \text{DEL}$. First the duplex case. Let c and $k = \lambda/\mu$ vary, and determine DEL so that ${}_cR_1^1 = .997$. The results are shown in Table 4.2. The amount of extra hardware available depends strongly on c and k (as our previous graphs would indicate).

Now consider the triplex case, as shown in Table 4.3. In this case, the dependence on c and k is not as strong, as expected.

4.4.5 Comparison of TMR Reliability With ${}_cR_2^1$

The comparisons between TMR and sparing which were discussed in the previous section are extended to include the failure coverage effect. The reliability RTS for TMR/simplex is

$$RTS = \frac{3}{2} e^{-\lambda T} - \frac{1}{2} e^{-3\lambda T}$$

Let

$$g(T) = {}_cR_2^1 - RTS$$

and note that

$$g(0) = 0$$

$$\frac{\partial g}{\partial T} = -\lambda(1-c)^2.$$

In general, the function g has the form shown in Fig. 4.7

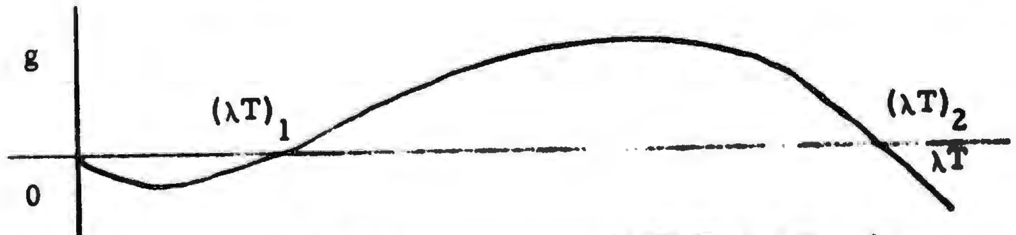


Figure 17 - Comparison of TMR and Sparing

Duplex

	C ← 1	C ← .98	C ← .95	C ← .9	C ← .8
	SEEK	SEEK	SEEK	SEEK	SEEK
K =	1	1	1	1	1
INC =	18.776	15.717	12.130	8.291	4.765
R0 =	0.945	0.953	0.964	0.975	0.985
R1 =	0.9969	0.997	0.998	0.999	0.999
K =	2	2	2	2	2
INC =	21.658	17.637	13.126	8.633	4.827
R0 =	0.937	0.948	0.961	0.974	0.985
R1 =	0.997	0.997	0.998	0.999	0.999
K =	5	5	5	5	5
INC =	24.203	19.241	13.887	8.867	4.866
R0 =	0.929	0.943	0.959	0.973	0.985
R1 =	0.997	0.998	0.998	0.999	0.999
K =	10	10	10	10	10
INC =	25.277	19.892	14.179	8.951	4.879
R0 =	0.926	0.942	0.958	0.973	0.985
R1 =	0.996	0.998	0.999	0.999	0.999

Table 4.2

Triplex

	C ←1	C ←.98	C ←.95	C ←.9	C ←.8
	SEEK	SEEK	SEEK	SEEK	SEEK
K =	1	1	1	1	1
INC =	51.915	47.534	41.241	31.739	17.641
R0 =	0.8558	0.8669	0.8835	0.9191	0.9484
R1 =	0.9792	0.9823	0.9864	0.9917	0.9973
R2 =	0.9970	0.9976	0.9984	0.9992	0.9998
K =	2	2	2	2	2
INC =	65.120	58.639	49.601	36.599	18.921
R0 =	0.822	0.838	0.861	0.896	0.944
R1 =	0.975	0.979	0.985	0.991	0.997
R2 =	0.997	0.997	0.998	0.999	0.999
K =	5	5	5	5	5
INC =	78.846	69.715	57.465	40.762	19.848
R0 =	0.789	0.811	0.841	0.884	0.942
R1 =	0.971	0.977	0.984	0.991	0.997
R2 =	0.997	0.997	0.998	0.999	0.999
K =	10	10	10	10	10
INC =	85.405	74.839	60.939	42.479	20.192
R0 =	0.774	0.798	0.832	0.880	0.941
R1 =	0.969	0.976	0.983	0.991	0.998
R2 =	0.997	0.997	0.998	0.999	0.999

Table 4.3

As done previously, use $r\lambda$ $(r > 1)$ instead of λ in ${}_cR_2^1$ in order to account for the extra checking and diagnostic circuitry which a sparing system would need. The results of an APL analysis of the function g can be summarized by the following table where $(\lambda T)_1$ and $(\lambda T)_2$ are the values of λT for which $g = 0$; i.e., ${}_cR_2$ = RTS as shown on the graph.

r	c	g	$(\lambda T)_1$	$(\lambda T)_2$
≤ 2	$\geq .9$	$\geq -8 \times 10^{-5}$	$\leq .02$	$>.5$
≤ 2	$\geq .95$	$\geq -5 \times 10^{-6}$	$\leq .003$	$>.5$
2.5	.85	negative		
2	.8	$-.0022$	.155	$>.5$
1.4	.8	$\geq -.0009$	.0789	$>.5$

This indicates that high values of failure coverage c are essential (in other words the detection and recovery must be good) if sparing is to be more reliable than TMR.

4.5 Mathematical Models for Modules Which Can Tolerate Multiple Failures

4.5.1 Exact Formulae The solution of the integral equations which led to equations 4.8) and 4.9) depended upon the fact that the derivative of the probability of failure has a simple expression. As shown in Appendix 2.2, 2.3 this formula could be proved by mathematical induction and the resulting integrals could be integrated by inspection.

When each unit under consideration has some error-correcting or healing properties, f becomes non-zero. The value for ${}^fR_0$ is obtained from the first f + 1 terms of the Poisson distribution.

$${}^fR_0(T) = e^{-\lambda T} \sum_{i=0}^{f} \frac{(\lambda T)^i}{i!}$$

Taking into account the fact that the f failures could occur in both the power-off and power-on states, there exists no such simple formulation for the probability density of failure $\frac{d}{dt}(1 - {}^f_cR^{\ell}_{m-1}(t))$ where f > 1. The number of states-configuration in which the (q-1) units are when the last spare is needed depends upon q and f, and is combinational. The direct attack on this problem is to use a next state transformation analysis. This does not seem reasonable as will be shown.

The integral equation recursive formulation for c = 1, ℓ = 1, is:

$${}^fR_m(T) = {}^fR_{m-1}(T) + \int_0^T e^{-\mu t} e^{-\lambda(T-t)} \sum_{i=0}^{f} \frac{(\mu t)^i}{i!} \sum_{j=0}^{f-i} \left\{ \frac{[\lambda(T-t)]^i}{j!} \right\} \frac{d}{dt}[1-{}^fR_{m-1}(T)]\, dt$$

where

$$e^{-\mu t} \frac{(\mu t)^i}{i!}$$

is the probability that exactly i failures have occurred in the spare in its power-off state before the switch-over and

$$e^{-\lambda(T-t)} \sum_{j=0}^{f-i} \frac{[\lambda(T-t)]^{j}}{j!}$$

is the probability that that spare will survive the mission after turning its power-on.

To date, no general, closed form expression has been found for ${}^{f}R_{m}$. Even for the relatively simple case of f= 1 the task of computing ${}^{1}R_{1}$ and ${}^{1}R_{2}$ has proven to be formidable, as can be seen from the results below:

$${}^{1}R_{0}(T) = e^{-\lambda T}(1+\lambda T)$$

$${}^{1}R_{1}(T) = {}^{1}R_{0}(T) + K^{2}e^{-\lambda T}\Big\{(1+\lambda T)[1-e^{-\mu T}(1+\mu T)] + 2(1-K)[1-e^{-\mu T}(1+\mu T+\frac{(\mu T)^{2}}{2!})]\Big\}$$

$$\begin{aligned}{}^{1}R_{2}(T) = {}^{1}R_{1}(T) + K^{2}e^{-\lambda T}\Big\{&K(1-K)(1+\lambda T)(1-e^{-\mu T})^{2} \\ &+ [2K(1-K)^{2} + (1+\lambda T)(1+K^{2})][1-e^{-\mu T}(1+\mu T)] \\ &+2(1-K)(1+K^{2})[1-e^{-\mu T}(1+\mu T+\frac{(\mu T)^{2}}{2!})] \\ &-\frac{1}{4}[2K(1-K)^{2} + (1+\lambda T)(1+2K-K^{2})][1-e^{-2\mu T}(1+2\mu T)] \\ &-\frac{1}{4}[(1-K)(1+2K-K^{2}) + (1+\lambda T)(1+K)][(1-e^{-2\mu T}(1+2\mu T+\frac{(2\mu T)^{2}}{2!})] \\ &-\frac{3}{8}(1-K^{2})[1-e^{-2\mu T}(1+2\mu T+\frac{(2\mu T)^{2}}{2!}+\frac{(2\mu T)^{3}}{3!})]\Big\}.\end{aligned}$$

$$
\begin{aligned}
{}^{2}R_{1}(T) = e^{-\lambda T}\left(1+\lambda T+\frac{(\lambda T)^{2}}{2}\right) \\
+ K^{3} e^{-\lambda T} \Big\{ & \left[1+\lambda T+\frac{(\lambda T)^{2}}{2}\right]\left[1-\left(1+\mu T+\frac{(\mu T)^{2}}{2}\right) e^{-\mu T}\right] \\
& - 3(K-1)(1+\lambda T)\left[1+\mu T+\frac{(\mu T)^{2}}{2}+\frac{(\mu T)^{3}}{6}\right) e^{-\mu T}\Big] \\
& + 6(K-1)^{2}\left[1-\left(1+\mu T+\frac{(\mu T)^{2}}{2}+\frac{(\mu T)^{3}}{6}+\frac{(\mu T)^{4}}{24}\right) e^{-\mu T}\right]\Big\}
\end{aligned}
$$

4.5.2 Approximation Methods These special cases show the difficulties in deriving a general formula involve f, q, μ and λ. To derive an approximation,

expanding ${}^{f}R^{q}_{m}(\lambda, \mu)$ about $\mu = 0$ gives

$$ {}^{f}R^{q}_{m}(\lambda, \mu) = {}^{f}R^{q}_{m}(\lambda, 0) + \mu \frac{\partial}{\partial \mu} {}^{f}R^{q}_{m}(\lambda, 0) + \frac{\mu^{2}}{2} \frac{\partial^{2}}{d\mu^{2}} {}^{f}R^{q}_{m}(\lambda, 0) + \ldots.. $$

Using

$$ \frac{\partial}{\partial \mu} {}^{f}R^{q}_{m}(\lambda, 0) \approx \frac{{}^{f}R^{q}_{m}(\lambda, \lambda) - {}^{f}R^{q}_{m}(\lambda, 0)}{\lambda} $$

${}^{f}R^{q}_{m}(\lambda, \mu)$ can be linearly approximated by

4.15) $$ {}^{f}R^{q}_{m}(\lambda, \mu) = {}^{f}R^{q}_{m}(\lambda, 0) + {}^{\mu}/_{\lambda} \left[{}^{f}R^{q}_{m}(\lambda, \lambda) - {}^{f}R^{q}_{m}(\lambda, 0)\right] $$

$$ = {}^{\mu}/_{\lambda}\, {}^{f}R^{q}_{m}(\lambda, \lambda) + (1 - {}^{\mu}/_{\lambda})\, {}^{f}R^{q}_{m}(\lambda, 0) $$

Since ${}^{f}R^{q}_{m}(\lambda, \mu)$ is an increasing function of μ, ${}^{f}R^{q}_{m}(\lambda, 0)$, ${}^{f}R^{\mu}_{m}(\lambda, \lambda)$,

and ${}^{\mu}/\lambda$ i.e. betweeen 0 and 1,

$$\max \left[{}^{f}R^{q}_{m}(\lambda,\lambda)\,,\ (1-{}^{\mu}/\lambda)\ {}^{f}R^{q}_{m}(\lambda,\mu)\right] \leq {}^{f}R^{q}_{m}(\lambda,\mu) \leq {}^{f}R^{q}_{m}(\lambda,0).$$

In Appendix 2.1.1, direct calculation has shown the error to be small.

Consider ${}^{f}R^{q}_{m}(\lambda,\lambda)$. Since $\lambda=\mu$, this reliability can be expressed in terms of combinations of the basic module reliability ${}^{f}R\ (\lambda,\lambda)$. As proved in 2.1.2

$$4.16)\quad {}^{f}R^{q}_{m}(\lambda,\lambda) = ({}^{f}R\ (\lambda,\lambda)\ )^{q} \sum_{j=0}^{m} \left(\frac{q \times j-1}{j}\right) (1-{}^{f}R(\lambda,\lambda))^{j}$$

with an alternative expression

$${}^{f}R^{q}_{m}\ (\lambda,\lambda) = 1 - \left(1 - {}^{f}R\ (\lambda,\lambda)\right)^{m+1} \sum_{j=0}^{q-1} \left(\frac{m+j}{j}\right) ({}^{f}R\ (\lambda,\lambda)\ )^{j}$$

It is also shown in Appendix 1.2 that ${}^{f}R^{q}_{m}(\lambda,\lambda)$ is approximated from below by

$$4.17)\quad {}^{f}R^{q}_{m}(\lambda,\lambda) \approx 1-\begin{pmatrix} m+q \\ q-1 \end{pmatrix} \left[\frac{(\lambda T)^{f+1}}{(f+1)\ !}\right]^{m+1}$$

Increasing f is obviously an efficient way to obtain more reliability.

Expressions for ${}^{f}R^{q}_{m}(\lambda,0)$ will be derived using a next state transformation analysis and by solving a set of linear differential equations. The problems encountered are enough to discourage an immediate attack on the general problem involving ${}^{f}R^{q}_{m}(\lambda,\mu)$

The general method followed is to set up a next state transformation in the form

$$4.18)\quad ES = S\ (t+h) = S\ (t) + h\,T\ S\ (t) + o\ (h^{2})\ \ BS\ (t)$$

where S is a n dimensional state vector, T is a n x n matrix, E is a next state operator and h is a small increment of time. The probability that more than one malfunction occurs is h is $0(h^2)$[Feller, 67].

This leads to the differential equation

$$\frac{dS}{dt} = TS$$

with boundary conditions $S = S(o)$ when $t = 0$. As shown in Appendix 2.2.4 this has a solution of the following form, writing $\rho = (f+1)\,m + f\,q$

4.20) $${}^{f}R^{q}_{m}(\lambda, 0) = e^{-q\lambda T}\left[\sum_{i=0}^{(f+1)(m+f)} \frac{(q\lambda T)^i}{i!} + \sum_{i=(f+1)(m+1)}^{\rho} a_i \frac{(q\lambda T)^i}{i.}\right]$$

Where $1 > a_j > a_{j+1} > 0$ for $(f+1)(m+1) \leq j \leq \rho$.

If $a_{(f+1)(m+1)}$ is written as $1 - p_{f(q-1)+1}$, (see Appendix 2.2.4) then for small λT

4.21) $${}^{f}R^{q}_{m}(\lambda, 0) \approx 1 - p_{f(q-1)+1} \frac{(q\lambda T)^{(f+1)(m+1)}}{[(f+1)(m+1)]!}$$

The general procedure is carefully derived in Appendix 2.2.4. The following example will illustrate the procedures. Consider the assemblage with $q = 2$ and $m = 1$ whose modules tolerate one error ($f = 1$). The procedure is to consider the set of 2 active units and 1 reserve unit. The state of the set will change as malfunctions occur. If the assemblage has had exactly j errors, and the assemblage will operate correctly, call this state S_j. Represent the failed state by F. Each state may be represented by the "error list" (a, b, c,) where the i^{th} entry gives the number of failures existing in module i, and - indicates the module is a spare.

Initially there are no malfunctions. The probability that a module failure occurs in the time interval h is λh. The possible changes, occurring in the internal h, are shown by the following Fig. 4.8 where each level represents the occurrence of an error; the particular configuration of errors is shown as (a, b, c,). Any active unit is equally likely to malfunction. For each level, the corresponding error lists, state, and number of distinct paths leading to the state will be listed.

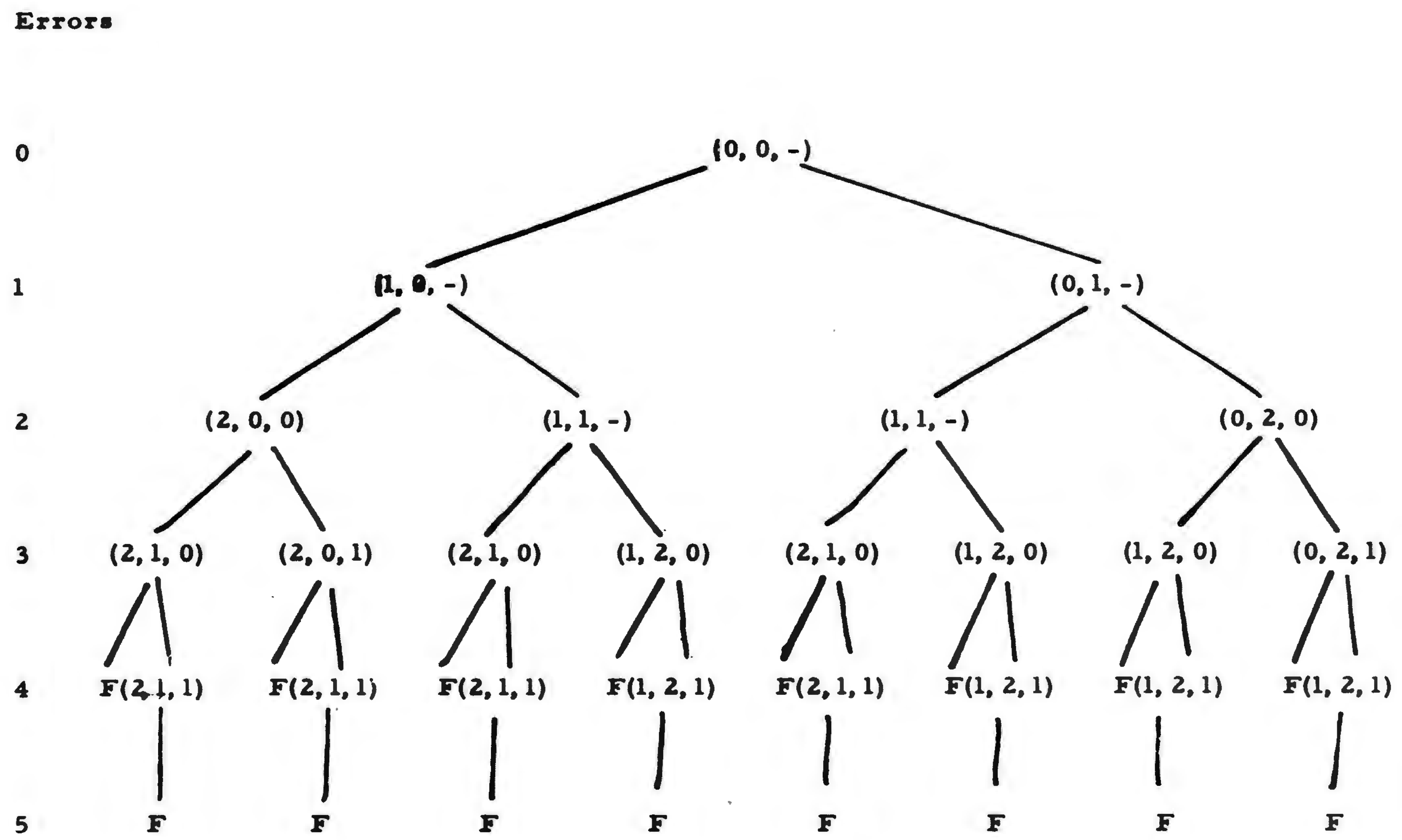

Errors
0
1
2
3
4
5
(0, 0, -)
(1, 0, -)
(0, 1, -)
(2, 0, 0)
(1, 1, -)
(1, 1, -)
(0, 2, 0)
(2, 1, 0)
(2, 0, 1)
(2, 1, 0)
(1, 2, 0)
(2, 1, 0)
(1, 2, 0)
(1, 2, 0)
(0, 2, 1)
F(2, 1, 1)
F(2, 1, 1)
F(2, 1, 1)
F(1, 2, 1)
F(2, 1, 1)
F(1, 2, 1)
F(1, 2, 1)
F(1, 2, 1)
F
F
F
F
F
F
F
F

FIG 4.8

Errors	State	Error List		Total Number of Paths
0	S_0	(0, 0, -)		1
1	S_1	(1, 0, -)		2
2	S_2	(2, 0, 0)	2	
		(1, 1, -)	$\underline{2}$	
			4	4
3	S_3	(2, 1, 0)		8
4	S_4	(2, 1, 1)		8
	F	(2, 2, 0)		8
5	F	(2, 2, 1)		8

The error list configurations and number of paths are important because the next state transformation depends upon the probability of going from one state to the next.

The next state transformations with associated probabilities are:

$$
\begin{array}{lll}
 & \xrightarrow{1 - 2\lambda h} & S_i \\
S_i & \xrightarrow{2\lambda h} & S_{i+1} \qquad 0 \leq i \leq 2 \\
 & \xrightarrow{1 - 2\lambda h} & S_3 \\
S_3 & \xrightarrow{\lambda h} & S_4 \\
 & \xrightarrow{\lambda h} & F \\
 & \xrightarrow{1 - 2\lambda h} & S_4 \\
S_4 & \xrightarrow{2\lambda h} & F \\
F & \xrightarrow{1} & F
\end{array}
$$

(λh is the probability a module failure occurs in interval h).

Equation 4.18 now becomes;

$$E\begin{pmatrix} F \\ S_4 \\ S_3 \\ S_2 \\ S_1 \\ S_0 \end{pmatrix} = \begin{pmatrix} F \\ S_4 \\ S_3 \\ S_2 \\ S_1 \\ S_0 \end{pmatrix} + h\begin{pmatrix} 0 & 2\lambda & \lambda & 0 & 0 & 0 \\ 0 & -2\lambda & \lambda & 0 & 0 & 0 \\ 0 & 0 & -2\lambda & 2\lambda & 0 & 0 \\ 0 & 0 & 0 & -2\lambda & 2\lambda & 0 \\ 0 & 0 & 0 & 0 & -2\lambda & 2\lambda \\ 0 & 0 & 0 & 0 & 0 & -2\lambda \end{pmatrix}\begin{pmatrix} F \\ S_4 \\ S_3 \\ S_2 \\ S_1 \\ S_0 \end{pmatrix}$$

and equation 4.19 is obvious. To solve 4.19 a nonsingular matrix P must be found such that

$$P\,T\,P^{-1} = J$$

where J is the Jordan normal form for T.

The initial state is represented by $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = I$

The solution for equation 4.19 is given by

4.22 $\quad P\,S = e^{tJ} \quad P\,I$ $\quad$ [CL, 55; p. 75]

The corresponding Matrices J and P are:

$$J = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2\lambda & 2\lambda & 0 & 0 & 0 \\ 0 & 0 & -2\lambda & 2\lambda & 0 & 0 \\ 0 & 0 & 0 & -2\lambda & 2\lambda & 0 \\ 0 & 0 & 0 & 0 & -2\lambda & 2\lambda \\ 0 & 0 & 0 & 0 & 0 & -2\lambda \end{pmatrix} \quad P = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$e^{tJ} = e^{-2\lambda t} \begin{pmatrix} e^{2\lambda t} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2\lambda t & \frac{(2\lambda t)^2}{2!} & \frac{(2\lambda t)^3}{3!} & \frac{(2\lambda t)^4}{4!} \\ 0 & 0 & 1 & 2\lambda t & \frac{(2\lambda t)^2}{2!} & \frac{(2\lambda t)^3}{3!} \\ 0 & 0 & 0 & 1 & 2\lambda t & \frac{(2\lambda t)^2}{2!} \\ 0 & 0 & 0 & 0 & 1 & 2\lambda t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$PS = \begin{pmatrix} F + \sum_{i=0}^{4} S_i \\ 2S_4 \\ S_3 \\ S_2 \\ S_1 \\ S_0 \end{pmatrix} \qquad PI = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The probability that the assemblage will be in state S_i may be determined by finishing the multiplication shown in equation 4.22 .

$$\begin{pmatrix} F + \sum_{i=0}^{4} S_i \\ 2S_4 \\ S_3 \\ S_2 \\ S_1 \\ S_0 \end{pmatrix} = e^{-2\lambda t} \begin{pmatrix} e^{2\lambda t} \\ \frac{(2\lambda t)^4}{4!} \\ \frac{(2\lambda t)^3}{3!} \\ \frac{(2\lambda t)^2}{2!} \\ \lambda t \\ 1 \end{pmatrix}$$

$${}^1R^2(\lambda, 0) = \sum_{i=0}^{4} S_i = e^{-2\lambda t} \left(\sum_{i=0}^{3} \frac{(2\lambda t)^i}{i!} + \frac{1}{2} \frac{(2\lambda t)^4}{4!} \right)$$

There are two feasible sets of formulas which have been derived. The first set f = 1 and derived formulas for small values of q; 1, 2, 3, 4, and 5, and arbitrary values of m. The second fixed m at small values, 1, 2, and 3, and derived approximations for small values of f (1 and 2) and arbitrary values of q. In each case, general forms were found for T, P and J which depended upon λ, q, m and f.

4.5.3 Specific Approximations Formulae

4.23) small q, f = 1, m arbitrary.

a) q = 1

$$ {}^{1}R^{1}_{m}(\lambda,\mu) = (1-{}^{\mu}/_{\lambda})\, e^{-\lambda T} \sum_{j=0}^{2m+1} \frac{(\lambda T)^{j}}{j!} + \frac{\mu}{\lambda}\left[1-(1-e^{-\lambda T}(1+\lambda T))^{m+1}\right] $$

b) q = 2

$$ {}^{1}R^{2}_{m}(\lambda,\mu) = e^{-2\lambda T}\left\{(1-{}^{\mu}/_{\lambda})\left(\sum_{j=0}^{2m+1} \frac{(2\lambda T)^{j}}{j!} + \frac{1}{2}\,\frac{(2\lambda T)^{2m+2}}{(2m+2)!}\right) + {}^{\mu}/_{\lambda}\,(1+\lambda T)^{2} \sum_{j=0}^{m} (j+1)\,[1-e^{-\lambda T}(1+\lambda T)]^{j}\right\} $$

c) q = 3

$$ {}^{1}R^{3}_{m}(\lambda,\mu) = e^{-3\lambda T}\left\{(1-{}^{\mu}/_{\lambda})\left(\sum_{i=0}^{2m+1} \frac{(3\lambda T)^{i}}{i!} + {}^{3}/_{4}\left(1-\frac{1}{3^{2m+2}}\right)\frac{(2\lambda T)^{2m+2}}{(2m+2)!}\left(1+\frac{\lambda T}{2m+3}\right)\right) + {}^{\mu}/_{\lambda}\,(1+\lambda T)^{3} \sum_{j=0}^{m} \frac{(j+2)(j+1)}{2}\,[1-e^{-\lambda T}(1+\lambda T)]^{j}\right\} $$

d) q = 4

$$ {}^{1}R^{4}_{m}(\lambda,\mu) = e^{-4\lambda T}\left\{(1-{}^{\mu}/_{\lambda})\left(\sum_{i=0}^{2m+1} \frac{(4\lambda T)^{i}}{i!} + {}^{7}/_{8}\left(1-\frac{1}{2^{2m+3}}\right)\frac{(4\lambda T)^{2m+2}}{(2m+2)!}\left(1+\frac{2\lambda T}{2m+1}+\frac{(\lambda T)^{2}}{(2m+3)(2m+1)}\right)\right) + {}^{\mu}/_{\lambda}\,(1+\lambda T)^{4} \sum_{j=0}^{m} \frac{(j+3)(j+2)(j+1)}{6}\,[1-e^{-\lambda T}(1+\lambda T)]^{j}\right\} $$

The forumlas for larger values of q will be considered only in the case of utmost necessity.

4.24) Small m, most f as shown,

a) m = 1, f = 1.

$$ {}^{1}R_{1}^{q}(\lambda, \mu) \approx 1 - \frac{q(3q-2)}{24}(\lambda, T)^{4} - ({}^{\mu}/{}_{\lambda})\frac{5q\,(\lambda T)^{4}}{24} $$

b) m = 2, f = 1

$$ {}^{1}R_{2}^{q}(\lambda, \mu) \approx 1 - \frac{15q\,(q-1)^{2} + q}{720}(\lambda, T)^{6} - \left(\frac{\mu}{\lambda}\right)\frac{q(\lambda T)^{6}}{720}\;(75q + 14) $$

c) m = 3, f = 1

$$ {}^{1}R_{3}^{q}(\lambda, \mu) \approx 1 - \frac{(105)\,(q-1)^{2}\,(q-2)\,q + 63q\,(q-1) + q}{40,\ 320}\;(\lambda T)^{8} $$

$$ - \left(\frac{\mu}{\lambda}\right)\frac{q\,(\lambda T)^{8}}{40,\ 320}\;(1050q^{2} + 567q + 902) $$

d) m = 1, f = 2

$$ {}^{2}R_{1}^{q}(\lambda, \mu) \approx 1 - \frac{q(10q-9)}{720}(\lambda T)^{6} - ({}^{\mu}/{}_{\lambda})\frac{19q\,(\lambda T)^{6}}{720} $$

These formulas have been incorporated in the REL program for calculating ${}^{f}_{c}R^{q}_{m}$ for the ARC system. Using the exact expressions derived earlier for ${}^{1}R_{1}$, ${}^{1}R_{2}$, ${}^{2}R_{1}$, R_{m} the following approximate equations were derived for small λT.

$$ {}^{1}R_{0} = 1 - \frac{(\lambda T)^{2}}{2\,!} $$

$$ {}^{1}R_{1} = 1 - \frac{(\lambda T)^{4}}{4\,!}\,[\,1 + 2G + 3G^{2}] $$

$${}^{1}R_2 = 1 - \frac{(\lambda T)^6}{6!} [1 + 6G + 21G^2 + 32G^3 + 30G^4]$$

$${}^{2}R_0 = 1 - \frac{(\lambda T)^3}{3!}$$

$${}^{2}R_1 = 1 - \frac{(\lambda T)^6}{6!} [1 + 3G + 6G^2 + 10G^3]$$

$$R_1 , 1 - \frac{(\lambda T)^2}{2!} [1 + G]$$

$$R_2 = 1 - \frac{(\lambda T)^3}{3!} [1 + 3G + 2G^2]$$

$$R_3 = 1 - \frac{(\lambda T)^4}{4!} [1 + 6G + 11G^2 + 6G^3]$$

- - - - -

$$R_5 = 1 - \frac{(\lambda T)^6}{6!} [1 + 15G + 85G^2 + 225G^3 + 274G^4 + 120G^5]$$

In general they are close to the other approximations for ${}^{f}R^{q}_{m}$ - just set q = 1 and replace G^i by G.

The ratios of the probabilities of failure,

$$\frac{1 - R}{1 - R\,(K = 1)}$$

were computed and are shown in Figure 7. Write

$$\frac{1 - {}^{f}R^{q}_{m}(G)}{1 - {}^{f}R^{q}_{m}(1)} \text{ as Ra } {}^{f}F^{q}_{m} \quad 1$$

then the limits approached in Figure 4. 9 are shown in the following table.

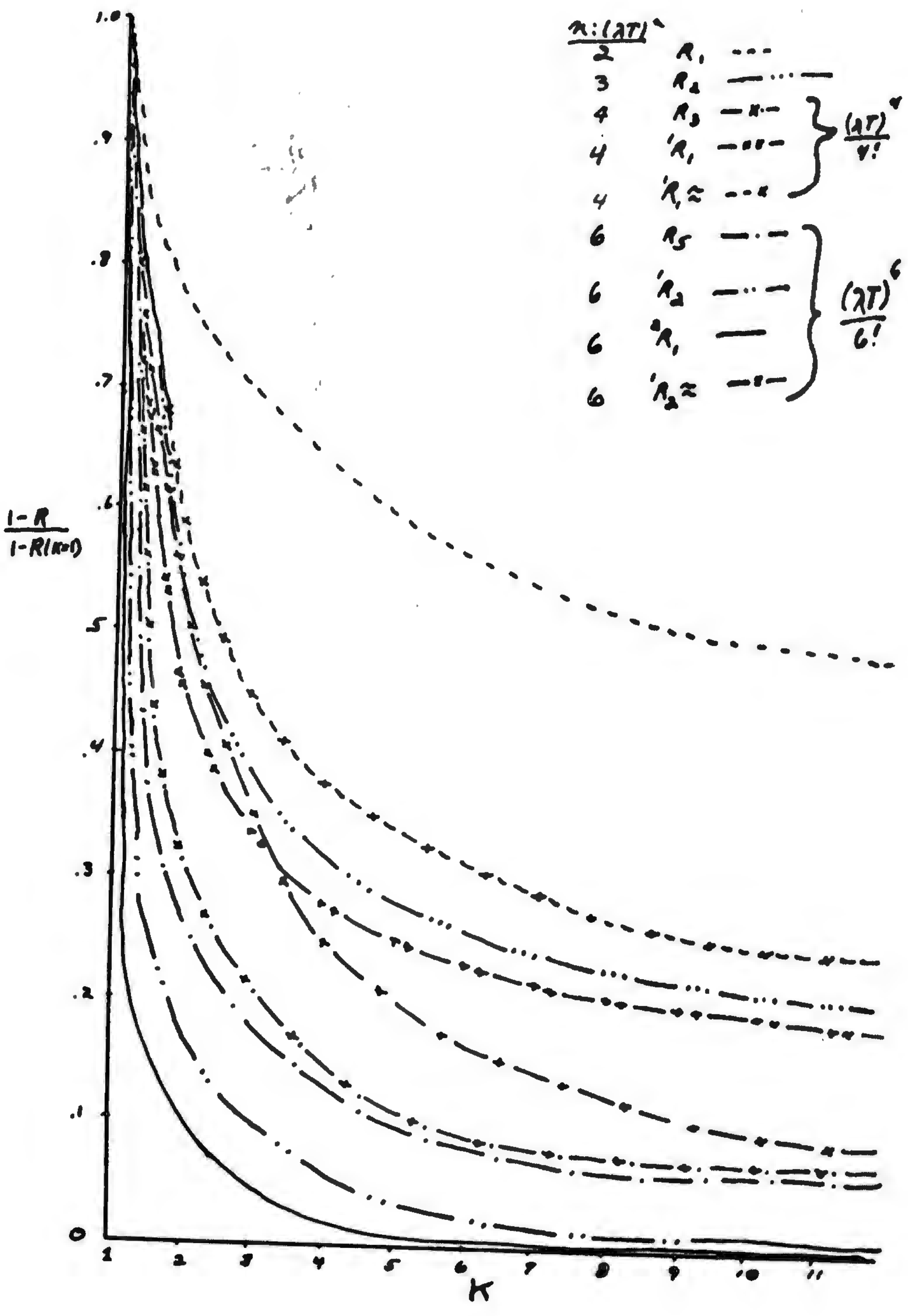


Figure 4.9 Variation of ${}^{\ell}R_m^{\,q}$ vs. K.

Ratio	Limiting Value
RA F_1	$^1/_2$
Ra F_2	$^1/_6$
Ra ${}^1F_1$	$^1/_6$
Ra ${}^1F_1$	$^1/_6$
Ra ${}^2F_1$	$^1/_{20}$
Ra F_3	$^1/_{24}$
Ra ${}^1F_2$	$^1/_{90}$
Ra ${}^1F_2$	$^1/_{90}$
Ra F_5	$^1/_{720}$

It must be remembered that these coefficient are multiplied by $(\lambda T)^n/n!$ as shown in the graph or e equations, so that variations in the approximations to ${}^fR^q_m$ are made that much smaller.

4.6 Computing System Reliabilities

The previous equations for the reliability of units and modules were augmented by approximations for the busses and status registers and incorporated into a computer program called REL[See Appendix 3]. This program was in APL language [Iverson, 62] and was put on the APL Terminal System [FI, 66] which is a time-sharing interpreter employing linearized APL programmed on an IBM System 360/50 or 67.

This REL program was used in an interactive procedure to determine the system reliabilities. Initializing is done by executing the SETUP program. Then any desired parameter or input values are changed from the base values. (For details see the program in Appendix 3). Then the RELIABILITY program is executed which does the calculation and prints out the results. The value of any variable at any time may be displayed for examination and change. The use of REL in balancing a computer system design is described in detail in [BCRS, 67]. The highly non-linear characteristics of the reliability equations make it impossible to guess at the number of spares or amount of segmentation required to "balance" a system design. "Balancing" a design consists in determining the configuration that meets the reliability, mission time, and duty cycle specifications, with each unit contributing close to its predetermined share to the probability of failure.

As an example of this, typical values for the Normalized (to 100) Probabilities Of Failure (NPOF) of the PCU are 191, 4.9 and .23 for 1, 2, and 3 spares respectively, where

$$NPOF = 100 \times (1-R[UNIT]) \div (1-R[SPEC.]).$$

The variation of the NPOF's with the change in the number of segments is not nearly so sharp as in the case when the number of spares is changed. Typical results for the NPOF's for the PCU are 15.4, 7.9, 4.9, and 3.4 for segmentation into 2, 3, 4, and 5 parts.

Neither of these last two observations is surprising, since they merely quantify the analytical results of the improvement factors of equations 4.9 and 4.11 .

4.7 Circuit Reliability Application of State Transfer Model

4.7.1 General Model A similar mathematical model has been applied to the analysis of the error process in a chain of n binary logic functions. The inputs and outputs are vector functions of logic variables, one on each line. The input for the (n+1)st element in the chain will be the output from the nth element. Let $G_o(n)$ be the good output at the nth step, $G_i(n)$ be the good input at the nth step, and $B_o(n)$, $(B_i(a))$ be the corresponding bad outputs (inputs) at the nth step. Let $P(G_o|G_i)$ be the conditional probability of good outputs given good inputs at any step. Then in matrix formulation,

$$\begin{pmatrix} G_o(n) \\ B_o(n) \end{pmatrix} = M \begin{pmatrix} G_i(n) \\ B_i(n) \end{pmatrix} = \begin{pmatrix} P(G_0|G_i) & P(G_o|B_i) \\ P(B_o|G_i) & P(B_o|B_i) \end{pmatrix} \begin{pmatrix} G_i(n) \\ B_i(n) \end{pmatrix}$$

The mathematical model gives an expression for $G_o(n)$ in terms of the characteristic roots and vectors of the matrix, and the initial states of the lines.

$P(G_o|G_i)$ etc., may be evaluated using assumptions about the independence of failures and conditional probabilities. The expression for $G_o(n)$ is dependent upon these probabilities, and this dependence seems important and has not been mentioned explicitly in the literature.

Assume, as usual, that the failures on input lines and in the logic are independent. Let the probability of failure of the logic be a in each logic function, and the probability of an initial error in the lines be μ.

In the literature a usual assumption has been that the probability of good outputs given good inputs $P(G_o|G_i)$ is $1-a$, the probability of no failures of the logic circuits. In addidion, it is assumed that bad inputs always give bad outputs. Under these assumptions the transformation matrix is:

$$M = \begin{pmatrix} 1-a & 0 \\ a & 1 \end{pmatrix}$$

and the expression for $G_o(n)$ is:

$$G_o(n) = (1-\mu)(1-a)^n$$
$$B_o(n) = 1-G_o(n)$$

Since $1-a < 1$, $G_o(n)$ approaches 0 as n takes on higher and higher values, while $B_o(n)$ approaches 1 at a rate determined by $1-a$.

In practical cases it is known that sometimes a failure in the logic will not result in a failure in the output if the inputs are good. This conditional probability of good outputs given good inputs but a failure in the logic is called P_2. Now $P(G_o | G_i)$ equals the sum of no failures in the logic unit plus P_2 times α.

Thus,

$$M = \begin{pmatrix} 1-\alpha+P_2\alpha & 0 \\ (1-P_2)\alpha & 1 \end{pmatrix}$$

and

$$G_o(n) = (1-\mu)(1-(1-P_2)\alpha)^n$$

$$B_o(n) = 1-G_o(n)$$

$G_o(n)$ still approaches zero as n increases, but at a slower rate. Since logic can be designed so that $P_2 > 0$ (for example, use all prime implicants in the construction of each logic function), a degree of reliability can be simply introduced in the logic design independent of the components.

Another assumption made in the literature is that compensating errors can occur. Using von Neumann's method, (cf. P1, page 151) the probability of good outputs given bad inputs is the probability of a failure in the logic, or α.

Now

$$M = \begin{pmatrix} 1-\alpha & \alpha \\ \alpha & 1-\alpha \end{pmatrix}$$

and

$$G_o(n) = 1/2 + (1/2 - \mu)(1-2\alpha)^n$$

$$B_o(n) = 1-G_o(n)$$

In this case it is usually assumed that $\alpha < 1/2$, so $G_o(n)$ approaches 1/2 as n increases, but at a faster rate than in the previous cases.

Examining the conditional probabilities as before, there is no reason for P_2 to be zero. In addition, the conditional probability of good outputs given bad inputs and bad logic is called P_3 and it is a conditional probability which is 1 only in an improbable limiting case.

Now,

$$M = \begin{pmatrix} 1-\alpha+P_2\alpha & P_3\alpha \\ (1-P_2)\alpha & 1-\alpha+(1-P_3)\alpha \end{pmatrix}$$

and

$$G_o(n) = \frac{P_3}{1-P_2+P_3} + (\frac{1-P_2}{1-P_2+P_3} - \mu)(1-(1-P_2+P_3)\alpha)^n$$

$$B_o(n) = 1-G_o(n)$$

In this case, with $\alpha < 1/2$, as n becomes large $G_o(n)$ approaches $\frac{P_3}{1-P_2+P_3}$ at a speed governed by $1-(1-P_2+P_3)\alpha$. The limiting value is still determined only by the assumptions about the values of the conditional probabilities and does not depend upon values of α.

So far we have considered the conditional probability of good outputs given correct input values but a failure in the logic (P_2) and the conditional probability of good outputs given a failure in the logic and some incorrect input values (P_3), and we have been assuming implicitly that the conditional probability of good outputs given correct inputs and correct logic is one. The remaining conditional probability is that of correct outputs, given good logic and some incorrect input values. Call this P_1, and, by experience, P_1 is not always zero.

In this case

$$M = \begin{pmatrix} 1-a+P_2 a & P_1(1-a)+P_3 a \\ (1-P_2)a & (1-P_1)(1-a)+(1-P_3)a \end{pmatrix}$$

The formula for the good output at the nth step, $G_o(n)$ may be written as

$$G_o(n) = \left(1-\frac{r_1}{r_1+r}\right) + \left(\frac{r_1}{r_1+r} - \mu\right)(1-r_1-r)^n$$

where

$r = P_1(1-a) + P_3 a$

$r_1 = (1-P_2)a$

a = probability of circuit failure

P_1 = conditional probability of good outputs given good logic but bad inputs

P_2 = conditional probability of good outputs given bad logic and good inputs

P_3 = conditional probability of good outputs given bad logic and bad inputs

μ = probability of bad inputs

or

$$G_o(r) = (1-\mu)(1-r_1)^n + \mu[(1-r_1)^n - (1-r-r_1)^n]$$
$$+ [(1-\frac{r_1}{r_1+r}) + \frac{r_1}{r_1+r}(1-r-r_1)^n - (1-r_1)^n]$$

Term	Function
$(1-\mu)(1-r_1)^n$	"Fail sure" term, limit as n increases is zero.
$\mu[(1-r)^n - (1-r-r_1)^n]$	Possible correction of erroneous input values.
$[1-\frac{r_1}{r_1+r} + \frac{r_1}{r_1+r}(1-r_1-r)^n - (1-r_1)^n]$	Possible correction of errors in earlier elements in the chain by later elements in the chain.

The values which may be assumed for the various conditional probabilities should be studied. The following table shows how $P\ (G_o\ |\ B_i, G_\ell)$ is computed for a single OR gate with the probability of an error or an input line being. ν .

Correct Input	Erroneous Input	P (erroneous Input)	Input State	Output State
00	00	$(1-\nu)^2$	Good	Good
	01	$\nu(1-\nu)$	Bad	Bad
	10	$\nu(1-\nu)$	B	B
	11	ν^2	B	B
01	00	$\nu(1-\nu)$	B	B
	01	$(1-\nu)^2$	G	G
	10	ν^2	B	G
	11	$\nu(1-\nu)$	B	G
10	00	$\nu(1-\nu)$	B	B
	01	ν^2	B	G
	10	$(1-\nu)^2$	G	G
	11	$\nu(1-\nu)$	B	G
11	00	ν^2	B	B
	01	$\nu(1-\nu)$	B	G
	10	$\nu(1-\nu)$	B	G
	11	$(1-\nu)^2$	G	G

P (Bad Input) = P $(B_i) = 2\nu-\nu^2$

P (Good Output, Bad Inout | Good Logic) = P $(G_o, B_i | G_\ell) = \nu - 1/2\, \nu^2$

$$P\ (G_o | B_i, G_\ell) = \frac{P\ (G_0, B_i | G_\ell)}{P\ (B_i)} = 1/2$$

The prediction parameters may be summarized as in the following table.

Output	Logic	Input	Conditional Probability
Good	Good	Good	1
Good	Bad	Bad	?
Good	Good	Bad	1/2
Good	Bad	Good	? ?

A parameter study was made by computer program for the following parameters.

P_2 = 0.0, 0.1, 0.3. 0.5

a = 0.1, 0.01, 0.001, 0.0001, 0.00001

r = 0.2, 0.1, 0.01, 0.001, 0.0001, 0.00001

μ = 0.0, 0.1

n = 1, 2, 3, 4, 5, 10, 20

The study showed the following results:

1. The dominant effect is due to the conditional probability of good output given good logic and bad inputs. If this conditional probability (CP) is greater than the probability of circuit failure (a), the probability of good output from the chain (λ_0) is high. The biggest change in λ_0 comes when CP is less than a, λ_0 is close to the 'fail-sure' value, the value taken when all conditional probabilities are effectively zero.

2. A small effect was due to changes in either the conditional probability of good outputs given bad logic and bad inputs.

3. If the a is about 1/10 of the probability of erroneous input, then the probability of the correction of erroneous input is about equal to the probability of the correction of errors in the chain.

The effect of this is to show that whenever

$$P\ (G_o | G_l, B_i) > a \approx 10^{-8}$$

the prediction results are strongly affected.

Following Feller [Feller 57] this formulation can be related to the usual reliability equations in the standard way. As Feller says,* "The difference between the (Markov) chain and our process (birth/death) lies in the fact that, with the latter, changes can occur at arbitrary times, so the number of transitions during time t is a random variable."

Replacing the conditional probabilities by functions assuming whatever the changes during (0, t), the probability approaches zero linearily with h plus terms 0(h) and the probability that more than one occurs is 0(h) (Postulates for the Poisson Process)* the following set of equations is obtained.

$$\begin{pmatrix} PG(t+h) \\ PB(t+h) \end{pmatrix} = \begin{pmatrix} 1-(1-P_2)\,ah+0(h) & (P_1(1-a)+P_3 a)\,h+0(h) \\ (1-P_2)\,a\ +\ 0(h) & 1-P_1(1-a)\,h-P_3\ a\,h+0(h) \end{pmatrix} \begin{pmatrix} PG(t) \\ PB(t) \end{pmatrix}$$

These lead to the equations

$$PB(t) = 1 - PG(t)$$

$$PG^1(t) + [(1-P_2)\,a + P_1(1-a) + P_3 a]\ PG(t) = P_1\,(1-a)+P_3 a$$

or, using previous notation,

$$PG^1(t) + (r_1 + r)\ PG(t) = r$$

If the initial conditions are $PG(0) = 1$,

then $PG(t) = (1 - \frac{r_1}{r_1 + r}) + \frac{r_1}{r_1 + r}\ e^{-(r_1 + r)t}$

which differs from the expression for $G_o(n)$ only by the term which approaches zero, as expected.

This result emphases that approximation of the conditional probabilities is important, and that the conditional probabilities approach zero as t increases. Handling these mathematically is not simple, however work will continue along these lines. Establishment of reasonable values for these conditional probabilities will enable field and laboratory results to be reconciled. At the present time the situation is described as follows in reference S1, p. 6. "Any attempt to compute the reliability of the Timing Generation and the complex logic it feeds, without making a great number of simplifying assumptions, would lead to a mathematical expression containing literally thousands of terms. For practicality, a program has been written for the IBM 7090 Computer, which is capable of evaluating the reliability of complex logic, up to and including Triple Modular Redundancy. This program was used to obtain all reliability estimates except a few special cases.

The basic operation of the program is to randomly fail component parts, both open and short, and trace the resultant logic failure through Guidance Computer or Data Adapter logic simulated in the program, to see if a system failure would result. This is accomplished many times (normally 20,000). System reliability is then computed by dividing the number of times no system failure occurred by the total number of times it was tried."

This approach gives a method of prediction–after the technology is chosen and prototypes are built.

Practically speaking, as these component failure rates are measured from the performance of components in systems, the effect of all conditional probabilities are included, and the usual reliability calculations give good results. However, component life as measured directly on the components will tend to give too pessimistic results if standard calculations are made, and these different values remain unreconciled.

Reconciliation is important, since, from our results, the component failure rates are clearly dependent upon the particular way the logic is designed. Guides are needed during computer specification and design so that reliability requirements will be met without overdesign.

Conversely, with a theoretical foundation values can be established for the conditional probabilities using results like the Saturn V data and component data. With these values, reliability predictions can be made without excessive simplification before prototypes are built.

4.7.2 TMR Reliability Application

Special cases were computed for TMR for two cases. Case 1, conditional probabilities of correction due to TMR are 1. Case 2, worst case, all conditional probabilities of good output given a failure in logic or inputs are zero. The results are as follows, where line triple refers to the three lines carrying the same signal.

		Case 1	Case 2
$G_0(n)$:	All outputs good	$(1-\epsilon)^3$	$(1-\epsilon)^3 R$
$G_1(n)$:	One error or at least one line triple	$3\epsilon(1-\epsilon)^2$	$3\epsilon(1-\epsilon)^2 R$
$G_2(n)$:	Two errors on at least one line triple	$3\epsilon^2(1-\epsilon)$	$3\epsilon^2(1-\epsilon)R$
$G_3(n)$:	Three errors on at least one line triple	ϵ^3	$\epsilon^3 R + (1-R)$

where ϵ is the probability of voter failure, and the other terms are as previously defined, and

$$R = r^2(3-2r)/[2\lambda^3(1-\epsilon + r/\lambda) - 3\lambda^2(1-\epsilon + r/\lambda) + 1 + 3r^2 - 2r^3]$$
$$\lambda = 1 - r_1 - r$$

where r_1, r were defined earlier.

5. Structure/Organization

5.0 Summary of Architecture/Organization

5.0.1 Evolution of ARC Design

The design of ARC started with dividing a standard computer for space applications into functional units to facilitate hardware controlled retry within the unit. Standby redundancy and segmentation were then assumed for each unit and the program REL used to determine the ARC I configuration and analyze variations. This technique of standby redundancy depended upon an efficient design of reconfiguration switches to effect self-repair.

Improvements in the ratio of mission time obtained to equipment employed were obtained by redesigning memories and arithmetic units which could tolerate one or more errors. The following table shows the various design configurations considered.

$R = .997$

Configuration	Time		Equipment	Time + Equipment	
	$\lambda/\mu = 1$	$\lambda/\mu = 5$		$\lambda/\mu = 1$	$\lambda/\mu = 5$
	$D = 1$	$D = .25$		$D = 1$	$D = .25$
Simplex	6.59	14.5	1	6.59	14.5
TMR	205	431	3.5	59	123
ARCI	3483	9068	3.5	995	2590
ARCII	4097	10879	2.5	1170	4340

Table 5.0 Mission Time for Various Organizations

5. 0. 2 New Design Techniques

5. 0. 2. 1 Reconfiguration Switches Reconfiguration switches to effect standby redundancy self repair techniques were designed with the property that a failure in the switch caused one of the modules being used to be unavailable. Since the exponential failure rate for components was assumed, this allowed the failure rate for circuitry for the correct fraction of the switch circuitry to be added to the module failure rate and the reliability of the computer computed in this way. The status registers can be designed to be error correcting using masking redundancy. With careful design techniques using TEST DETECT to determine the effect of component malfunctions, reconfiguration switches can be designed so that single component failures will disable only a single module and most double failures will have the same result.

By using a switching scheme which permits switching only to adjacent lines but which will overcome m failures with m spare lines, the total amount of added circuitry is reasonable when measured by the criteria of (Mission Time/Equipment).

5. 0. 2. 2 Error Tolerant Memory Unit Using the same type of reconfiguration switching it is easy to design a reconfigurable bit plane memory unit. This is superior, in circuitry needed and speed of response, to error correcting coding of memory words. A new method of dynamic storage address blocking achieved error toleration in the addressing circuitry. Using these techniques, an assemblage of memory units could withstand several circuit malfunctions with small loss of capacity and no loss of speed.

5.0.2.3 Ultra Reliable ROS Configurations The same techniques which made a memory module more reliable can be used in a simplified form to make a ROS used for computer control error tolerant. The major modification is in the output reconfiguration network with minor savings in the address blocking.

5.0.2.4 Error Tolerant Byte ALU (BALU) The BALU is designed so that its bit planes are relatively independent (this affects primarily its adder). By adding a spare bit plane and changing the carry and shifting circuitry a reconfiguration switch can be used to switch bit planes and make the BALU tolerate a single component malfunction.

5.0.2.5 TMR/Sparing Using ideas derived from the reconfiguration switching, it is possible to combine the features of TMR and standby redundancy to design error tolerant circuits called TMR/Sparing. Their reliability is approximately $1 - (\lambda T)^{n-1}$ instead of $1 - (\lambda T)^{\frac{n+1}{2}}$ for n odd and greater than three.

5.0.2.6 Designing ARC Methods of designing and operating ultra reliable computers are studied. It is shown by example how to use TEST DETECT to design completely checked combinational circuitry. The checking circuitry output could be checked by a simple diagnostic test. Operating ultra-reliable computers is discussed. The necessity for a 'sentinal' to operate during turn off periods is shown, and the similarity between computer 'being up' procedures after turn off and after an error signal is treated. The design of special hardware and the storage of tests in the ROS to facilitate these procedures is discussed.

5.1 Introduction

5.1.1 Description of Architectural System Goals

The goal of this chapter is to formulate a description of the architectural characteristics of a self-repairing computer. To achieve the high availabilities required of planned space-borne computers with the technologies available either now or in the indefinite future, it will be necessary to employ a variety of organizational techniques to handle failures in the various elements composing the computer. These failures are basically of two kinds; intermittent (transient) or solid (permanent). For intermittent failures it is only necessary to detect and overcome errors introduced by the temporary malfunctioning of a component, but when the failures are solid it it is usually necessary to repair the computer in some manner.

Since these computers will be operating in environments where it is either impossible or infeasible to have manual intervention, high availability must be attained by automatic detection of errors (hardware or program) self-diagnosis and self-repair. Several organizational techniques are available to do each of the chores of error detection, error location, error correction and repair.

Detection of errors by parity checking is an efficient and widely used technique. It has been used extensively in data storage elements and channels and to some extent in arithmetic units and control circuitry [CMPR, 64]. In a highly available computer it should undoubtedly be employed throughout. Another method of error detection is to have duplication of every organ and then do everything twice. A comparison of the two outputs then reveals the existence of errors [Kemp, 62; LC, 66 ; Saturn, 65]. A related method is the use of two rail logic. A fourth method of error detection, which is only applicable to solid type failures, is the use of diagnostic tests [CMPR, 64 ; Roth, 66 ; MA, 63].

Location of errors can be accomplished to some extent by parity checking and to a much greater extent and precision through the use of diagnostic tests.

Correction of errors is possible in many ways. Masking redundancy [Saturn, 65] has been widely used. Three ways that have in common the virtue of healing the effects of both intermittent and solid type failures are 1) the use of error correcting codes and associated error correcting circuitry [Brown, 60 ; Kautz, 62], 2) the use of triple modular redundancy together with voting elements [BH, 66] and 3) quadding [Tyron, 62] many analysis have been made of systems using replication of logic signals [Saturn, 65 ; FRST, 65]. Such systems proved inadequate to meet a stringent requirement of .997 for a mission time of 10,000 hours with a duty cycle of .25 . Not only was the number of replicated signals high, but in a practical environment the probability of errors induced by external sources become relatively great. The problems of continuation after such errors, or a sudden burst of transient errors, were difficult to solve efficiently. However, these techniques can be used to design some parts of an ultra-reliable computer.

If stand-by redundancy is used, this involves replacing the unit or module containing the failing component with a spare one. Since spares are limited, it is highly desirable to be able to differentiate between the two kinds of failures. An alternative strategy is to employ spares in a cyclic manner so that once switched out they can be switched in again.

Another method which may be used is to replace the logical transfer function of a unit by a combination of the transfer functions of other units. This is usually done by having several control paths to carry out a single instruction; and using the control path which uses correctly functioning units. If the errors are known to be the result of a transient failure, then instruction retry is a feasible method of correcting the errors. In some cases where the data has been contaminated, a program recovery method can be employed. If the failure is of a solid type, the program recovery must await a repair before it can be successful. Our study has encompassed most of these techniques, applying them to specific organs of a computer.

The basic goals of this Automatically Repaired Computer (ARC) system are:

1. Increased hardware reliability and self-repair capability through:
   a) redundant components (TMR), duplicate memory modules, and other devices ,
   b) error detection and correction coding implemented in circuitry,
   c) diagnostic procedures to determine the precise nature and extent of component malfunctions,
   d) partitioning and reconfiguration of redundant units to provide graceful degradation.

2. Increased system reliability and self-repair capability through:

   a) hardware controlled micro instruction and instruction retry, and possible alternate instruction sequencing selection,

   b) hardware controlled I/O data checking, certification, and retry,

   c) information protection by hardware/supervisor program,

   d) planned recovery, rollback, and restart procedures after catastrophic errors.

<u>5.1.2 Definition of Terms.</u> In an ARC organization, the computer consists of a number of functional units, or simply units, suitably interconnected: each unit consists of an appropriately chosen number of modules. The modules are chosen so that when a given module malfunctions it is relatively easy for the computer to detect the error and diagnose the unit to ascertain which module malfunctioned. To implement automatic repair, replicas of the modules are joined in segments. Segments are disjoint except for wires, and are composed of modules which are individually replaceable by switching. After detection of an error and diagnosis, repair is effected by automatic switching. The number of modules in a segment depends upon the reliability requirements. The wires connecting segments are called switchable interfaces. The lines currently active are defined by status registers.

The functional units of the ARC considered are shown in Figure 5.1. A more detailed description of the functions of the units will be given later. The basic logical and data path interconnections are also shown.

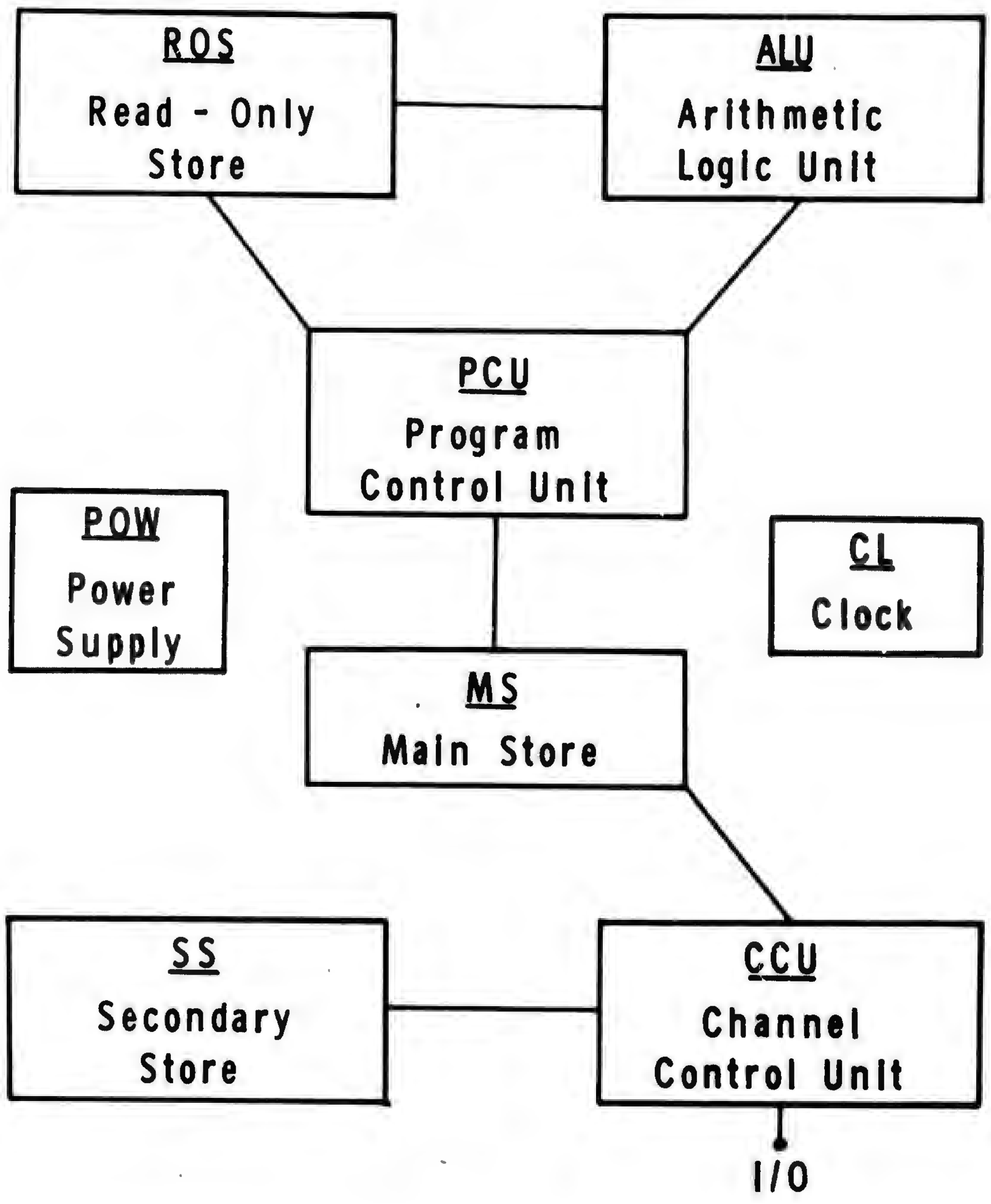


Fig. 5.1 Typical computer configuration.

PCU SEGMENTING

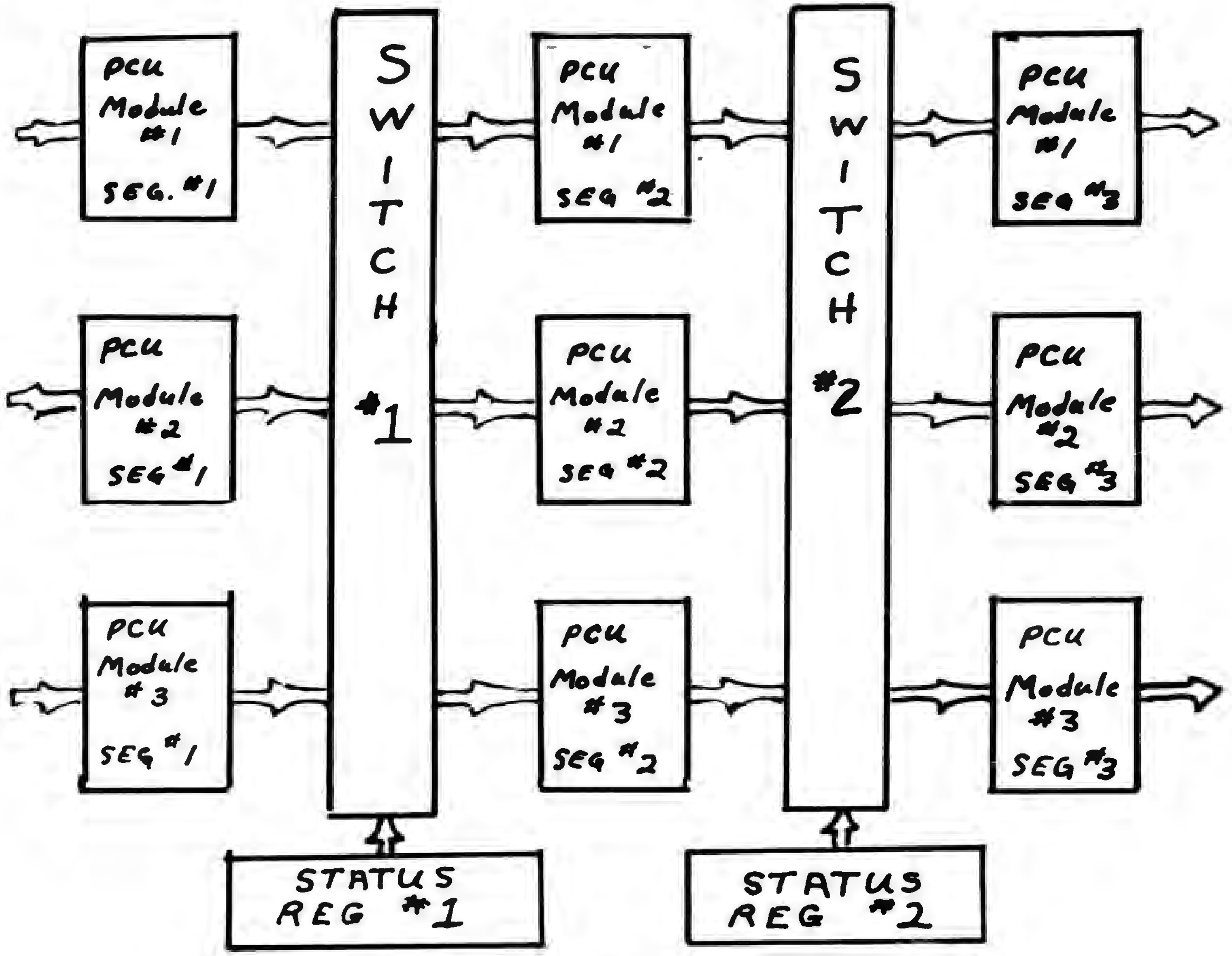


Figure 5.2

Figure 5.2 shows the segments of a particular unit - the PCU. Each segment contains three modules. The three modules in each segment perform the same transfer function. For example if data is processed by module #1, segment #1, then by module number #2, segment #2, then by module #1, segment #3 a correct result will be obtained if all modules function correctly. Clearly there are 27 possible paths through the PCU to use the correct module.

5.2 Architecture Organization

5.2.1 Overall Organization

The basic concept of this organization is one of distributed function, distributed control, and variable sequencing. The system will consist of various varieties of units of the following types:

1. Modules of main memory.
2. Modules of Bulk Memory (large capacity, for storage of files as necessary).
3. Channel Control Units (includes connections to all I/O Units) (CCU).
4. Arithmetic and Logic Units (ALU).
5. Console and Display Units.
6. Program Control Units (PCU).
7. Switching Controls, Control Memories and Multiple Busses to sequence, control and pass information among the above.

Each unit control responds to external stimulation, and the action may vary with control information, but control rests within each unit to perform given functions. Input and output interfaces are defined and standard. A block diagram of such a system is shown as Figure 5.1.

The memory orientation of the system shows clearly in the block diagram. As always, the control of this computing system lies in the set of programs, procedures and information that are used for total system control. These programs operate on their "state vector"; e.g., all values of all variables and other information which, together with the program itself, determines the further course of computation. Thus control is in the supervisory program through thick and thin, good and erroneous signals.

Recovery from an error involves:

1. Detection of the malfunction, together with the prevention of the unbounded propagation of erroneous data and/or control information.
2. Assessment of the damage to critical hardware, and possible reconstruction of critical information concerning hardware (eg. status of earlier failed modules). This is the diagnosis process.
3. Self-repair of critical hardware, if necessary.
4. Assessment of the damage to critical information (i.e., programs and data).
5. Reconstruction of critical information, if necessary.
6. Restarting of critical tasks.

This organization was chosen to facilitate carrying out these tasks. After a description of the units, methods of using these units to achieve the goals will be discussed and a rationale for choosing this organization given.

5.2.2 Description of the Units

5.2.2.1 Main Memory Modules

Essential information will be stored in duplicate with information protection bits and error detection codes (probably parity) to detect some errors. Both copies of the information can be accessed and the copies compared to detect the rest of the errors. The driver lines addressing memory can be parity checked (only one should be up) to detect address selection failures. A method of designing such decoders so that all single failures can be detected or bypassed will be given later (cf. section 5.4.7). Hardware echo checking can be done after the storage of data.

Address error masking can be done by being able to independently disable modules. The paging concept of addressing may be used to carry out this procedure with a minimum of inconvenience to the user as long as spare capacity is available. If the paging device uses an associative memory to determine the base for input selection, changing the base is easy for local control. Such changes would follow from the parity check on the devices combined with diagnosis. A more sophisticated method of bypassing errors in addressing for each Basic Operating Memory module (BOM) will also be described (cf. section 5.4.3.3).

A majority (70%) of the failures in a BOM result in an error in a single bit position of a word. Error correcting coding can be used, however a new and more efficient method will be described in section 5. 4. 3. 2 .

The distributed functions of this main memory are:

a) reading of information, with parity error detection and correction as necessary of data, with distribution of data to both the PCU and CCU.

b) storing of information with hardware echo check that information has been stored, from both the PCU and CCU.

c) detection of information protection bits.

d) validation of commands to change these bits (comparison of table names with timing update for example).

e) detection of errors in selection circuitry.

f) updating of error correction information.

g) supplying program control units when necessary with coded information describing the memory status.

h) reconfiguration of main memory to bypass some malfunctions after diagnosing a permanent error.

5. 2. 2. 2 Modules of Bulk Memory

On an extended mission, bulk backup storage of files of programs, constants and data becomes necessary. The files of programs and constants can be read only (with a section for modification due to the discovery of program errors). Roll-back information must be dynamically stored, and data gathere during flight saved for later use.

The functions to be performed are similar to those performed by the modules in main memory. However, the initiation of control is done by the CCU.

5.2.2.3 Program Control Unit (PCU)

In a nutshell, the PCU will contain the scratchpad memory (General Purpose Registers), index registers, program counters, interrupt masks and status bits, and in general the hardware necessary to contain the information necessary to control the execution of an algorithm in the presence of noise.

The real worth of the PCU is to focus attention on the program being executed by the computer. Think of the program as a tree structure with the computation and data transfer being done at the branch points. For micro reliability the branch point operations - computation and data transfer - must be locally retriable. Thus the PCU initiates instruction retry. The next problem in reliability consists of overseeing the paths joining the branch points - especially as several paths will be transversed in parallel. This control is the function of the PCU. When an error occurs, recovery or restart must be begun. Thus the PCU has the further function of maintaining data in duplicate in memory modules so that checkpoint and recovery operations may be initiated after catastrophic failures. So it must store and protect the system state.

Another basic function of the PCU is to control the diagnosis and self-repair procedures, including bringup. After an error signal, an ARC bootstrapps its way into operation. Forbes et al [ARSY, 65] have derived such a procedure. Such procedures must avoid "hard core"- defined as any modules whose failure will cause the computer to cease reliable operation. There must be at least two diagnostic sub-systems, each capable of controlling the diagnostic process, verifying status information, and setting reconfiguration circuitry.

If the entire computer does not operate continuously, then it must contain a clock and network which respond to external or clock induced stimuli. After the initiating stimulus, the procedures outlined previously must be followed. These procedures will be performed by the PCU.

In addition, the PCU will fetch instructions from main storage, decode instructions and status indicators, activate the control memory, store data in main storage, and initiate I/O and secondary storage activities.

The PCU contains a scratch pad duplicated memory, with all the error detection and correction features described before for memories. For effective program control this memory will contain locations for the following:

1) Instruction Counter (IC) - include paging and memory protection.
2) Instruction History Counter (every time a branch occurs, the former contexts of the IC are stored here).
3) Index Registers (like System/ 4π - C32 [Vandling, 66]).
4) Clock and instruction execution counter capable of interrupting program status.
5) Program Storage Word Information - I/O mask information.
6) Interrupt Registers (as needed).
7) Arithmetic Back-Up Registers:
   - data storage register (memory)
   - accumulator
   - accumulation extension
   - status bits
8) IOCC's (several).

9) IOCC History Registers.

10) Information for the Diagnostic Program.

11) ALU error status information.

12) CCU error status information.

13) Storage and I/O error status information.

14) Alternate PCU information (for use if supervisor program is using PCU).

15) Partitioning and paralleling ALU information.

In summary, the distributed functions of the PCU are:

a) fetch instructions and data from main memory, executing the paging algorithm and memory protection.

b) activate the control memory and switching controls.

c) store data in the main memory, using the paging algorithm and memory protection schemes.

d) activate CCU's, provide them with IOCC's.

e) update the contents of all registers previously listed-when done automatically by hardware control.

f) transfer program control in all branching and interrupt situations.

g) provide diagnostic and bring-up control.

h) provide fail-safe switching of program control and automatic reconfiguration upon detection of an error; a single error must not hang up the system.

Each PCU will contain or point to the information necessary to:

a) Control a program (including paging algorithms).

b) Retry macro instructions.

c) Supplement CCU and memory retry capabilities.

d) Connect ALU's in parallel, and control their action.

e) Partition the computer after error detection.

f) Initiate recovery from catastrophic errors.

g) Control and call diagnostics.

5.2.2.4 Arithmetic and Logic Units

Conventional computer instructions will be stored in main memory. These will activate variable micro instruction sequences depending upon the hardware available. The ALU's will execute the micro instruction sequences which make up the variable instructions.

During the execution of a micro instruction sequence an ALU will remain connected to a control memory. The ALU will store input information and temporary results in internal registers. It will get its input information from a PCU and will store its output there.

Each ALU will be 9 bits wide (8 information bits and 1 parity bit): for this reason, they will be referred to as Byte ALU's or BALUs. One, two or four ALU's will be able to work in parallel on a micro instruction such as ADD and will pass information from one another. The operations performed by each ALU will be checked, and error signals sent to the approrpiate PCU.

The checking of arithmetic operations is well covered in the literature. The main problem in designing such units will be in the partitioning and interconnection, and in the fffficient checking of logical and shifting operation.

The distributed functions of this control unit are:

a) addition, subtraction both 1's and 2's complement (for partitioning and combining).

b) accepting sequences of micro instructions and signals for multiplication and division from the control memory.

c) logical operations; AND, OR, XOR, NOT.

d) shifting operations.

e) comparison operations, indication of branching conditions.

f) accepting control signals from the control memory.

g) partitioning and passing signals and data for parallel processing (this is discussed under switching controls in section 5.4.2).

h) checking for the above operations.

i) passing the results of operations and status information such as overflow, number of shifts, branch conditions, occurrence of error, to the PCU.

j) retry of micro instructions after detection of error; passing of error status information to the PCU.

For higher reliability such BALU's can be designed so that they can tolerate at least one error. This design will be described later (cf. section 5.4.5).

5.2.2.5 Switching Control and Control Memories

The switching control is activated as shown on the block diagram, initiates distributed control, and monitors the distributed control. The control works through standard interchangeable interfaces in each type of unit. This switching control will be checked and will have error correcting features. It is discussed in more detail later.

Common control information is stored in the control memory. In particular, diagnostic control and tests are stored here for activation by the diagnostic interrupt. This interrupt is time controlled, so that the frequency of tests may be increased as the reliability degrades.

5.2.2.6 Channel Control Units

The commands for a CCU are the I/O Control Commands (IOCC) which it receives from a program control unit.

These commands identify the I/O operation, I/O device, core storage, address and information protection status (like System/4 π - C32 [Vandling, 66]).

The channel control unit carries out these commands, checking the information received for satisfaction of a checking algorithm (parity for example) and for reasonableness (upper and lower bounds for sensed information, for example). I/O units may be polled, and the PCU interrupted only when a significant change occurs. The new values and the time since the last change will be available. An associative memory will allow such tasks to be performed quickly and cheaply. The information in the reasonableness checking portion of the associative memory will come from the PCU and this check may be disabled upon command.

If an error is detected, the CCU will notify the PCU, store appropriate status information there, and take independent retry action, as dictated by the command and I/O unit. For example, information from I/O sensors may be reread, using an alternate connection path. Information from bulk memory can be treated similarly. The ability to use alternate paths is extremely important for recovery from errors (as well as in the diagnosis of equipment failures). If the CCU cannot obtain correct information by retries, it will notify the PCU so that recovery action may be started.

If no errors are detected, after the successful execution of one command in a chain, the control unit gets the next IOCC from the same PCU. Control information requested by the PCU will also be obtained. The console and display devices are treated as I/O units.

The distributed functions of the Channel Control Unit are:

a) input transmission of data with parity error detection, reasonableness, and significance checks on data (where applicable).

b) output transmission of data, with parity error detection.

c) set up path for input transmission of data.

d) set up path for output transmission of data.

e) remove path (or sections of path) from pool of paths available for use.

f) sense interrupts from I/O devices, with masking as directed by PCU or own logic (e.g., during retry analysis).

g) cause selected device to make its current status available.

h) retry IOCC a number of times specified by the PCU, storing status information in the PCU as directed.

i) execute diagnostic commands affecting the PCU, ALU, main memory and other CCU's as necessary.

j) poll attached I/O as required.

k) be able to be electrically disconnected from tne rest of the system in case of permanent malfunction.

5.2.2.7 Console and Display Units

These will be treated as I/O units and controlled with the CCU's. In the absence of a specific environment no further planning has been done for this.

5. 2. 3 Estimates of Circuitry Needed for ARC Implementation.

In order to estimate the reliability of the ARC, the interconnections between buss widths were estimated, the individual units were assumed segmented for reliability and the extra switching control necessary was estimated, and the number of units needed for necessary thruput were estimated.

Unit	Number Required	Number of Segments
MS	2, 3	8 K capacity
SS	2, 3	16 K capacity
PCU	1	2-8
CCU	1	2-6
ROS	1	2-4
BALU	4	1
Clock	1	4
Power	1	
Busses	Width	
PCU-ROS	92	
BALU-ROS	15	
PCU-ROS	12	
MS-PCU	36	
MS-CCU	36	
CCU-SS	9	

The amount of hardware necessary was estimated using the 4 π /EP as guide. The unit for estimating is pages of logic (475 circuits) or circuits.

The BUSSES exert a substantial effect on the system reliability: spare lines are required. The reliability formula for the busses weresplit into a buss-status register portion and a buss-line portion. The failure rate for the buss register is proportional to the number of lines and the total number of terminal modules. The failure rate for each buss line is proportional to the number of terminal modules on the buss.

Unit	EP	ARC I	Reason for Augmentation
	pages	pages × No. = total	
BALU	8	1 1/2 × 6 = 9	Checking and Operand Storage
PCU	7	10 × 3 = 30	Addition of general-purpose register for certain checking and diagnostics
CCU	4	5 × 3 = 15	Retry diagnostics
ROS	Crushed Core Stack		Divided Control
	5	6 × 4 = 24	Added bussing
MS	Core Stack	+	
	.2 × 3	.25 × 9 = 2.25	
	Core Stack	+	
	.2 × 3	.25 × 8 = 2	
Clock	1	3	
	26.2	85.25	
Buss		3 circuits × width × (sources + sinks)	Only circuits that will bring bus line down
Partitioning Switching		Parametrical - multiply number of pages by 1.1, 1.2, 1.0 .	

The failure rates used in the calculation are shown in the following table.

TECHNOLOGIES

TYPICAL FAILURE RATES / 10^6 HOURS

	1968		1970	
Page of Logic	6		3	
2K. ROS	20		14	
Power Supply	15		10	
16 K. Drum	15		9	
8 K. Core Storage				
Logic	47	59	38	49
Arrays	12		11	

POWER-OFF EFFECTS ON λ

Solid State Devices - λ/5 - λ/10

Cores, Connections - λ (No change)

FAILURE DISTRIBUTIONS -

POISSON STILL VALID

5.3 Determining ARC Configurations Using REL

5.3.1 Specifications and Parameters The computer design specifications call for a reliability (probability of success) of .997 for a mission length of 10,000 hours. The duty cycle, the assumed date for the component failure rates, and the value of the ratio of the power-on to power-off failure rate values (λ/μ) were all parameters of the investigation. The calculations centered around a duty cycle of 0.25 , a date of 1968 and a value of 5 for λ/μ . Another important parameter, which was discovered during the course of the work, is the checker and diagnostic coverage discussed in section 4.3.2.

For a given system configuration, the mission time that just meets the specified system reliability was computed as being the most informative way of seeing the significance of any parameter change.

5.3.2 Hardware Involved Four designs were considered. The first one was a simplex computer system consisting of a CPU, storage and channels with a performance and capacity about the same as that of an IBM System 360/44. As implemented in IBM 4 Pi Technology [BCJ, 67] this simplex system has the failure rate values of column 1 of Table 5.1.

The second computer was designed by simply assuming that a TMR'd version of the first was available. The TMR'd system was considered to have the same failure rates and be segmented in the same way that ARC II, the last design, was.

[Gurzi, 65]. The Saturn study [BH, 66; AES-EPO, 65] showed that in a practical case building a TMR'd computer with 7 sections took 4.5 times as much equipment as a corresponding simplex computer.

The third computer, ARC I, was designed by separating computer functions into groups for ease of implementing retry by hardware. The ALU lost its index and other registers, becoming a unit for performing arithmetic and logic functions on given operands (to facilitate micro instruction retry). Extra hardware was assumed for the retry, operand storage for retry and extended checking. This ALU lost hardware in toto, and so the failure rate decline of this and other failure rates are shown in column 2. of Table 5.1. The PCU gained the registers (index, etc.) which control the program, additional checking circuits and hardware to control diagnostics and computer turn-on and bring-up (instruction retry would be implemented here). The increment I is a parameter of the study, and is the increment added to account for the extra switching circuitry when a unit is segmented. The CCU's hardware was increased because the channel had independent retry control and additional diagnostic facilities. The ROS logic hardware is increased because of divided control and more checking abilities. The ROS is

FAILURE RATES /10^6 HOURS FOR VARIOUS ORGANIZATIONS

UNIT	SIMPLEX	ARC I	ARC II
ALU	37.6	28.2	32
PCU	32.9	47 + I	47 + I
CCU	18.8	23.5 + I	23.5 + I
ROS			
Logic	23.5	28.2	32.9 + I
Arrays	20	20	20
MSTORE			
Logic	47.94	48.175	49.35
Arrays	12	12	12
SSTORE			
Logic	47.94	48.175	49.35
Arrays	8	5.38	5.38
CLOCK	4.7	4.7 + I	4.7 + I
POWER	10	15	15
BUSSES			
Switch	.06 X No. of terminal modules		
Status Reg.	.1 X No. of terminal modules X buss width		

Table 5.1 Failure Rates/10^6 Hours for Various Organizations

is also segmented. The Main Store and Secondary Store logic were increased because of additional checking. The clock is segmented and TMR'ed.

The fourth computer, ARC II, was designed by making certain of the ARC I modules tolerate one circuit malfunction so that a minimum of two malfunctions are required before the module must be replaced. (f = 1 in the calculations). The width of the ALU modules are increased from 8 bits to 9 and one decoder is duplicated (see section 5.7.5). The increase in failure rate, and all ARC II failure rates are shown in column 3 of Table 5.1. The ROS addressing and core storage are made error tolerant, as described in section 5.4.4 and the failure rates are increased. The Main Store and Secondary Store failure rates are increased for the same reason (see Section 5.4.3).

The failure rates for the busses were estimated from the hardware necessary to switch connectors from one line to another and record the change in a status register (see Section 5.4.2).

<u>5.3.3 The Simplex Computer</u> The reliability of the simplex configuration of Table 5.2 was computed and the mission time that gave the required system reliability was 6.59 hours. This corresponds to a Mean Time Before Failure (MTBF) of 2200 hours, or 4 failures per year. When the 1970 failure rates were assumed, the mission time increased to 8.82 hours. At lower system reliability specification values the mission time went up to 2 .4 hours for R = .99 and 112.5 hours for

```
                    CONFIGURATION

UNIT       N       M        Q     100×C    S    100×I     F
 ↓       TOTAL   SPARES    REQ    COVER   SEG    INC     FAIL
ALU        1       0        1      100     1      0       0
PCU        1       0        1      100     1      0       0
ROS        1       0        1      100     1      0       0
MSTORE     3       0        3      100     1      0       0
CCU        1       0        1      100     1      0       0
CLOCK      1       0        1      100     1      0       0
SSTORE     3       0        3      100     1      0       0
POWER      1       0        1      100     1      0       0

                 YEAR=1968   LA+MU=5

             RELIABILITY CALCULATION


TOT SYS REL.=.9970     PROB OF FAIL.= 99.8(NORMALIZED TO 100)
SYS REL SPEC=.9970     MISSION DURATION=14.50 HOURS   DUTY CYCLE=0.25
       ALU     PCU      ROS    MSTORE   CCU    CLOCK   SSTORE  POWER
NPOF   7.27    6.36     4.54   45.17    3.63    0.91    30.12   1.93
RELS 3N7819  3N8091  3N8637  2N8644  3N8909  4N7274  3N0963  4N4200
POFS 3Z2181  3Z1909  3Z1363  2Z1356  3Z1091  4Z2726  3Z9037  4Z5800
```

TABLE 5.2

R = .95. This ratio of 20 improvement in mission time, as the reliability requirements were lowered to .95 is discussed later in section 5.3.5 with a much lower change obtained with the ARC I (and ARC II) organizations.

5.3.4 The Triple Modular Redundant Computer The same computer organized in a TMR manner of 15 segments has a mission time of 181 hours. If no overhead for bussing, partitioning and voting is assumed, then this value goes up to 271 hours. While these mission times are a significant improvement, they fall far short of the specified requirement of a 10,000 hours. A TMR design also requires at least four times as much power as a simplex one, which is a liability for space computers.

5.3.5 The ARC I Computer A functional discription of the ARC computer was given earlier in this section. Fig. 5.3 shows the organization of these units and their buss connections. The data paths are solid lines and the control paths are dotted lines. The small boxes below the switches represent status registers.

The process of determining an appropriate number of spare units to employ in a balanced design has been fully described in reference [BCRS, 67]. The configuration of ARC I is given in Table 5.3. This represents a hardware increase over the simplex design by a factor of 3.5. The mission time achievable by this orga-

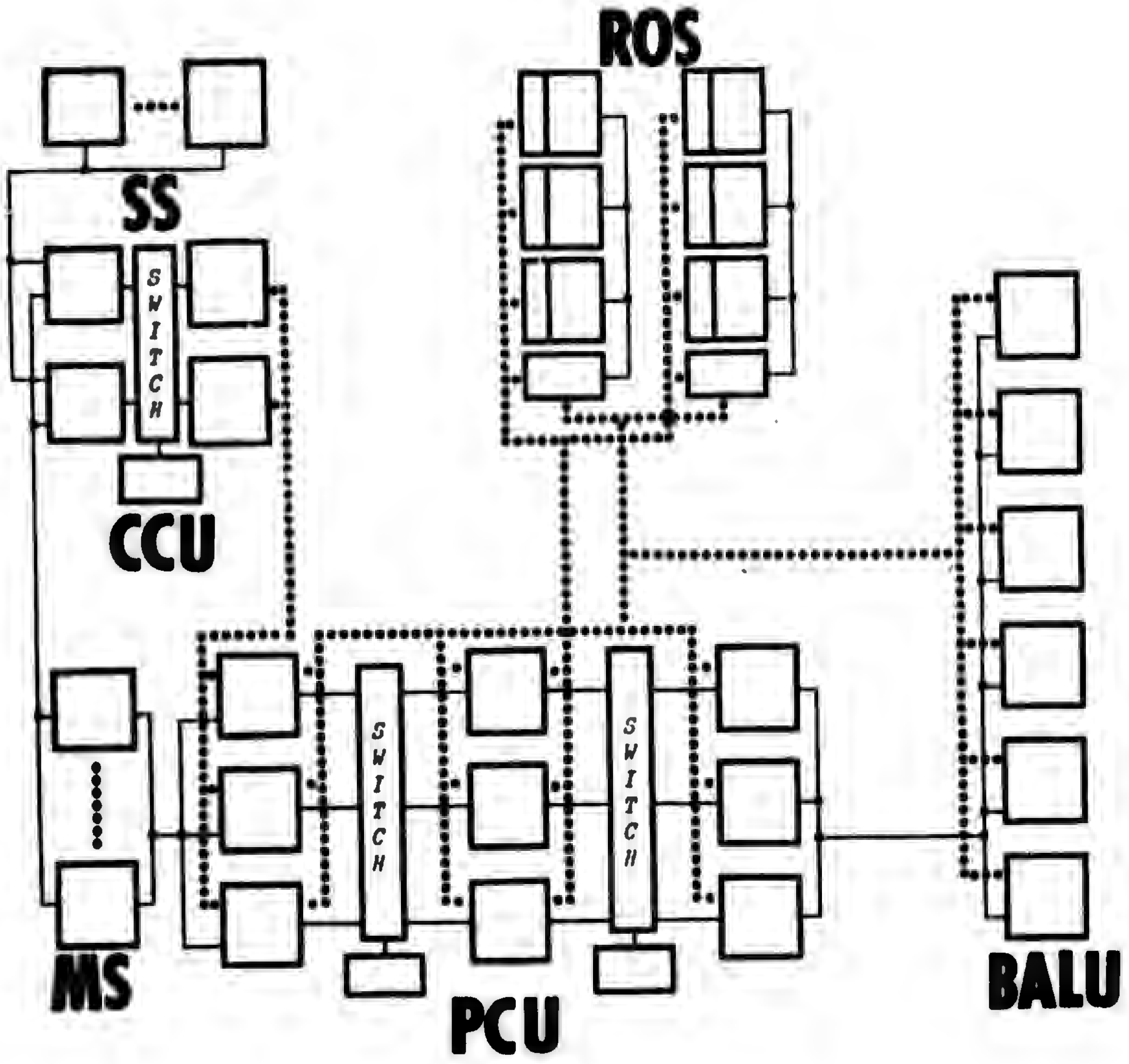


Figure 5.3 ARC I Organization

```
                               CONFIGURATION

          UNIT        N       M       Q     100×C    S    100×I    F
           ↓        TOTAL   SPARES   REQ    COVER   SEG    INC    FAIL
          BALU        6       2       4      100     1      0      0
          PCU         3       2       1      100     3     10      0
          ROS         4       3       1      100     4     10      0
          MSTORE      9       6       3      100     1      0      0
          CCU         3       2       1      100     2     10      0
          CLOCK       3       0       3      100     4     10      0
          SSTORE      8       5       3      100     1      0      0
          POWER       3       2       1      100     1      0      C

          BUSS       BPB     BPR     BPM     BBR     BCM    BCS
          WIDTH      12      95      36      15      36     36

                          RELIABILITY CALCULATION

TOT SYS REL.=.9970    PROB OF FAIL.=99.9(NORMALIZED TO 100)           YEAR=1968
SYS REL SPEC=.9970    MISSION DURATION=9068 HOURS    DUTY CYCLE=0.25    LA+MU=5

       BALU  PCU    ROS  MSTORE CCU  CLOCK SSTORE POWER  BPB   BPR   BPM   BBR   BCM   BCS
NPOF  7.67 12.73   4.60 13.55  3.28   9.02  7.04  2.53  0.07 19.43  5.11  0.20  5.11  3.89
RELS 3N769 3N617 3N862 3N443 4N015 3N702 3N788 4N242 5N779 3N417 3N846 5N401 3N846 3N883
POFS 3Z231 3Z383 3Z138 3Z557 4Z985 3Z298 3Z212 4Z758 5Z221 3Z583 3Z154 5Z599 3Z154 3Z117
```

TABLE 5.3

nization is 9068 hours. In this computer printout NPOF stands for the normalized probability of failure, REL stands for the reliability and POF (= 1 - REL) for the probability of failure. In RELs the digit before the N is the number of leading nines and in POF's the digit before the Z is the number of leading zeros .

Several parameter runs were made, the first of which was to determine the influence of the overhead required for segmenting a functional unit. The equation used to determine the failure rate of a module of a segmented unit is

$$\lambda \text{ module} = (\lambda \text{unit}/ S) \times 1 + I (S - 1)^5.$$

The calculated reliability was then raised to the S th power to get the reliability of the unit. A value of .1 for I was assumed as the best engineering approximation. For values of I of 0, .1, .2 and .3, the corresponding mission times became 9317, 9068, 8774 and 8442. Thus an estimate which is wrong by a factor of 2 of the amount of hardware required for segmenting only changed the mission time by 3.3%.

The next parameter varied was the duty cycle. For values of D of .25, .50, .75 and 1.00 the corresponding mission times became 9068, 6614, 5142, and 4189 hours.

The value of λ/μ is not known precisely so a parameter run was made varying this ratio. For values of K of 10, 5, 2 and 1 mission times became 11188, 9068, 5694 and 3483 hours. The advantages to be gained by shutting power off on the spares are very large. On the other hand, the values of λ/μ is still unknown. Also,if the phenomena involved consist solely of temperature effects, then the value might well be different in a space environment than surface measurements would indicate.

The component failure rate predictions (Table 2.1) are lower for 1970 than for 1968. A calculation employing these lower values gave an improved mission time of 12487 hours. This was about what was expected. Since the mission time T and the failure rate λ always occur as the product λT, in all formulae and approximations, a decrease by any factor in the failure rates will manifest itself by that same factor increase in the mission time.

The mission time became 12427 hours with a system reliability of .99 and 18551 hours with a system reliability of .95. The ratio of improvement of 2 contrasts with a ratio of 20 obtained for a simplex computer. This reflects the fact that in the ARC organizations, the reliability is approximated by 1 minus a constant times λT raised to a power which is 1 for the simplex computer and higher than 3 for the ARC I(see Eq. 4.17 or 5.1).

The last parameter varied was the checker and diagnostic coverage C. As C degraded becoming successively 1, .99, .98,

.95, .90, the mission time decreased from 9068 to 8572, 8945, 6348 and 3720 hours. These results illustrate the importance of achieving very high coverage in both the dynamic checkers and the diagnostics.

These results are summarized in Table 5.4.

Mission times for ARC I Paramater Changes

I	T	D	T	K	T	C	T
0	9317	.25	9068	10	11188	1.00	9068
.1	9068	.50	6614	5	9068	.99	8572
.2	8774	.75	5142	2	5694	.98	8045
.3	8442	1.00	4189	1	3483	.95	6348
						.90	3720

Y	T	R	T
		.997	9068
1968	9068	.99	12427
1970	12487	.95	18551

Table 5.4 Mission Times for ARCI Parameter Changes .

5.3.6 The ARC II Computer

Next, the value of the failure tolerance parameter F was increased to 1 for the BALU, ROS, MSTORE and SSTORE to reflect the improved designs described in sections 5.4.3, 5.4.4, and 5.4.5. Table 5.5 shows the calculated results with theARC I configuration. An examination of the NPOF values for these improved units shows that they contribute nothing to the failure of the system, indicating too many spares. Similarly the large BPR (PCU-ROS) buss value indicates a weak point in this

```
                     CONFIGURATION

UNIT      N      M       Q     100×C    S    100×I    F
 ↓      TOTAL  SPARES   REQ    COVER   SEG    INC    FAIL
BALU      6      2       4      100     1      0      1
PCU       3      2       1      100     3     10      0
ROS       4      3       1      100     4     10      1
MSTORE    9      6       3      100     1      0      1
CCU       3      2       1      100     2     10      0
CLOCK     3      0       3      100     4     10      0
SSTORE    8      5       3      100     1      0      1
POWER     3      2       1      100     1      0      0

BUSS     BPB    BPR     BPM    BBR     BCM    BCS
WIDTH    12     95      36     15      36     36

                 RELIABILITY CALCULATION

TOT SYS REL.=.9970    PROB OF FAIL.=99.9(NORMALIZED TO 100)         YEAR=1968
SYS REL SPEC=.9970    MISSION DURATION=10658 HOURS    DUTY CYCLE=0.25    LA+MU=5

      BALU   PCU    ROS   MSTORE CCU   CLOCK SSTORE POWER  BPB   BPR   BPM   BBR   BCM   BCS
NPOF  0.00  20.40  0.36  0.00   5.28  13.68  0.00   4.06  0.12 32.45  8.47  0.33  8.47  6.45
RELS 9N240 3N387 4N891 8N719 3N841 3N589 8N786 3N878 5N638 3N026 3N745 5N017 3N745 3N806
POFS 9Z760 3Z613 4Z109 8Z282 3Z159 3Z411 8Z214 3Z122 5Z362 3Z974 3Z255 5Z983 3Z255 3Z194
```

TABLE 5.5

design. The numbers of spare units was reduced to those indicated by the configuration of Table 5.6. While the amount of hardware per individual unit is up (see Table 5.1), the number of units is down and the amount of bussing required is considerably less. The total hardware required for the ARC II is 2.5 times the simplex computer and is 5/7 of the ARC I. Nevertheless, the mission time was improved, and the reason for this behavior is the smaller sized buss from the PCU to the ROS(BPR).

Parameter runs similar to those summarized in Table 5.4 were made and the results are listed in Table 5.7.

MISSION TABLES FOR ARC II PARAMETER CHANGES

I	T	D	T	K	T	Y	T	R	T
0	11450	.25	10879	10	13484	1968	10879	.997	10879
.1	10879	.50	7771	5	10879	1970	15897	.99	15537
.2	10271	.75	6022	2	6762			.95	24289
.3	9653	1.00	4909	1	4097				

Table 5.7 Mission Tables for ARC II Parameter Changes.

5.3.7 Balancing the Computer Configuration

As mentioned earlier in section 3.6, the NPOF of a unit varies strongly with the number M of spares provided. Table 5.8 lists the NPOF's of four units for both ARC I and ARC II configurations, first with one less spare each, then with the number designated in Tables 5.3 and 5.6, and lastly with one more spare each.

```
                              CONFIGURATION

          UNIT       N      M      Q    100×C     S    100×I     F
           ↓       TOTAL  SPARES  REQ   COVER    SEG    INC     FAIL
          BALU       5      1      4     100      1      0       1
          PCU        3      2      1     100      3     10       0
          ROS        3      2      1     100      4     10       1
          MSTORE     5      2      3     100      1      0       1
          CCU        3      2      1     100      2     10       0
          CLOCK      3      0      3     100      4     10       0
          SSTORE     5      2      3     100      1      0       1
          POWER      3      2      1     100      1      0       0

          BUSS      BPB    BPR    BPM    BBR     BCM    BCS
          WIDTH      12     95     36     15      36     36

                          RELIABILITY CALCULATION

TOT SYS REL.=.9970    PROB OF FAIL.=99.9(NORMALIZED TO 100)          YEAR=1968
SYS REL SPEC=.9970    MISSION DURATION=10879 HOURS    DUTY CYCLE=0.25    LA+MU=5

      BALU   PCU    ROS  MSTORE CCU   CLOCK SSTORE POWER  BPB   BPR    BPM   BBR   BCM   BCS
NPOF  0.05 21.66   9.52 12.88   5.60 14.25   2.71   4.31  0.09 21.23   2.55  0.18  2.55  2.55
RELS 5N850 3N350 3N714 3N613 3N831 3N572 4N186 3N870 5N731 3N363 4N235 5N472 4N235 4N235
POFS 5Z150 3Z650 3Z286 3Z387 3Z169 3Z428 4Z814 3Z130 5Z269 3Z637 4Z765 5Z528 4Z765 4Z765
```

TABLE 5.6

	M - 1	M	M + 1	
ARC I	183	7.67	.27	BALU
ARC II	53	.05	.00	
I	298	12.73	.59	PCU
II	424	21.66	1.15	
I	64	4.60	.33	ROS
II	229	9.54	.39	
I	71	18.55	4.63	MSTORE
II	292	12.88	.58	

Table 5.8 Effect of Failure Tolerance on ARC.

As can be seen, the variation is stronger for the improved units and less for the PCU which has fewer spares in ARC II then in ARC I. This behavior is explained by expressing Eq. 4.17 by the following simplified equation, where B is slowly varying.

Eq. 5.1 $R \simeq 1 - B(\lambda T)^{(F+1)(M+1)}$

As can be seen, when $F = 1$, as for the improved BALU, a variation of 1 in M varies the exponent by 2. Therefore, while it might have previously been thought that the configuration of ARC II was over-designed by having too many spare BALU's, in actual fact this is not the case, and the design is balanced as well as is possible. The BPR buss still needs improvement.

5.4 Engineering Specifications

5.4.1 Introduction

In this section a set of detailed engineering specifications will be given for units which posed reliability problems as shown by use of the REL program. These specifications are new, improved and more reliable ways of designing units to meet the previously outlined functional specifications. They are not complete engineering designs, but the essential particulars to solve reliability problems are detailed. The first unit specified is the switchable interface-the key to implementation of a computer using stand-by redundancy to achieve reliability. Three kinds of interfaces: module-module, module-buss and buss-module are treated. Reliability tradeoffs are shown. The second unit specified is a BOM-because of this modules relatively low intrinsic reliability. The modules are specified so that malfunctions in either core storage or addressing circuitry could be bypassed. These techniques are then applied to the ROS to achieve sufficient reliability. The nature of the ROS allows simplifications to be made. So far reconfiguration and multiplexing of equipment has been on the level of large functional units. The replacement techniques are then extended to encompass the design of the BALU.

A merger of TMR with Sparing with reconfiguration (called TMR/Sparing) is specified. This has the advantages of providing 1) a uniform treatment for transient and solid failures 2) immediate hardware detection for most failures, and 3) a reliability approaching that of sparing with reconfiguration for approximately the same amounts of hardware invested.

When assemblages of circuits are being logically designed, a question which arises is, how many malfunctions will not be caught by the checking circuits? If they are not caught, is erroneous information dessiminated? If no information is erroneous, will a second malfunction be caught, and what further implications must be considered? The sixth section shows a technique for designing decoders so that all malfunctions will be detected. To do this, extensive use must be made of the program TEST DETECT. As shown by the tradeoffs, a mixture of checking and diagnostic tests are most efficient. The final section discusses the design of hardware to generate diagnostic tests (when activated by the ROS) and thus drastically reduce the storage requirements for diagnostic tests, especially when used with circuits designed using the techniques of the previous section. The hardware designed will test parity checking circuits efficiently.

Each individual unit will now be described in detail.

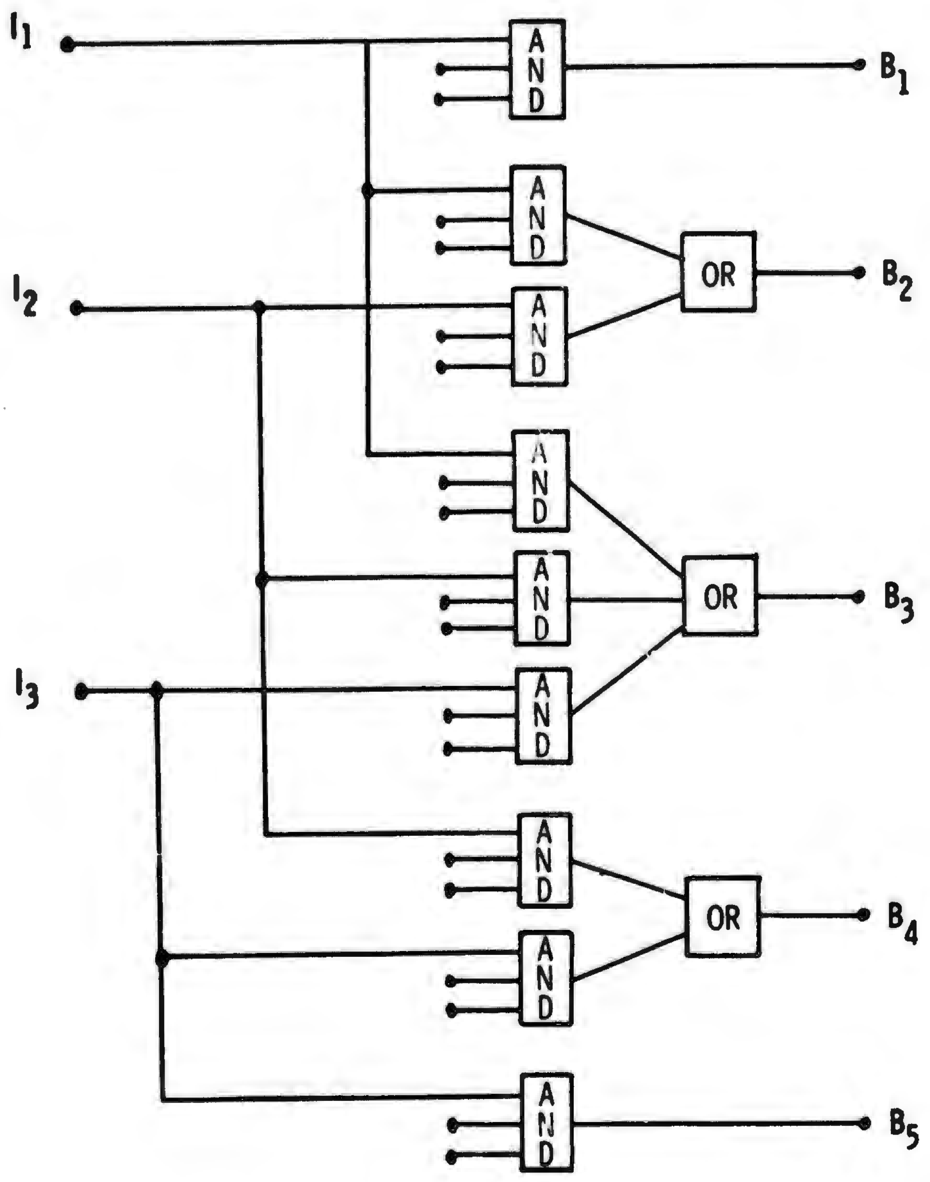
5.4.2 Switchable Interfaces

5.4.2.1 Introduction

The key concept of an Automatically Repaired Computer using stand-by redundancy is the replacement of the failed module by a stand-by spare. In order to maximally improve reliability the switches and status registers to implement the replacement and keep track of the present configuration must be efficiently designed. The fan-in/fan-out requirements are crucial. If m general purpose spare lines are used for q information lines, where each spare can replace any of the q lines, then every one of the switches controlling the action of the m spares must have a fan-in of q and a fan-out of q. Adding powering requirements to this will clearly defeat the desired gain in reliability.

The scheme used is to switch the $j^{\underline{th}}$ line to line j, $j+1$, $j+2, \ldots, j+m$, $0 \leq j \leq q$. Since m is much less than q (typically 1 or 3 against 9 to 36) this reduces the fan-out to manageable proportions and does not increase the power requirements much.

Figure 5.4 shows the case for $q=3$ and $m=2$. The top diagram illustrates a simple implementation of the switching circuit. This implementation is for explanation only, and is not an example of a reliable switch. The number of state triggers is chosen for simplicity of the switch. For example, if there is an error in B_2, then the state triggers are set to 00, 01, 01 and I_1 is connected to B_1, I_2 to B_3 and I_3 to B_4.



I_1	00	01	11		
I_2		00	01	11	
I_3			00	01	11
	B_1	B_2	B_3	B_4	B_5

$\{SR_1, SR_2\}$

Figure 5.4 Switchable Interface Illustration for $q = 3$, $m = 2$.

5.4.2.2 Basic Analysis

There are three basic ways in which information is transferred between units in ARC. They are shown in the following table:

	MAPPING			LINES
1)	Module	→	Module	1 out of $N \rightarrow N$
2)	Buss	→	Module	$M \rightarrow N$ $(M > N)$
3)	Module	→	Buss	$N \rightarrow M$ $(N < M)$

These mappings are performed by switches which are controlled by status registers. The basic ground rules for designing switches is to design the switch so that a failure in the switch will disable only the device receiving information. In this case the circuits comprising the switch may be added to the circuits in the receiving device for purposes of reliability calculations. In general, the rule for constructing the switch at a device interface is that only the source lines may be fanned out. No switch gate can have its output fanned out or a single gate failure will affect more than one module.

5.4.2.3 Simplex Design

5.4.2.3.1 Mapping Information from Module to Module

In Fig. 5.5 two identical modules in segment one, m_1^1, m_2^1, are being connected to two identical modules, m_1^2, m_2^2, in segment two. The jth output line from m_1^1 is denoted d_{1j}^1. Generally, d_{ij}^k specifies a line which is the jth output from module i in segment k. The status register has two states S_n, $n = 1, 2$, depending upon whether module n is being used. The device status register SR is a two stage shift register with states 10 and 01.

The lines d_{ij}^k, and the lines defining the states S_n, are routed into the switches in front of each module. To get information from d_{11}^1 during state S_1 or from d_{21}^1 during S_2, the line d_{11}^1 is ANDed with S_1 and d_{21}^1 with S_2 and these outputs are ORed to form the input information line I_{11}^2, the first input line for module one in the second segment.

To prevent one circuit from bringing down the system, there is no need to provide additional lines for interconnection because a line going down simply means that one of the units connected to the line cannot be used further and a spare must be switched in.

Using status registers with error correcting coding will provide more reliable mappings. However, the designer must be alert to remember which data is being transferred. Many times the grouping of terms to replace equipment will result in the fundamental ground rule being broken and the design of a switch which is part of the computer hard core.

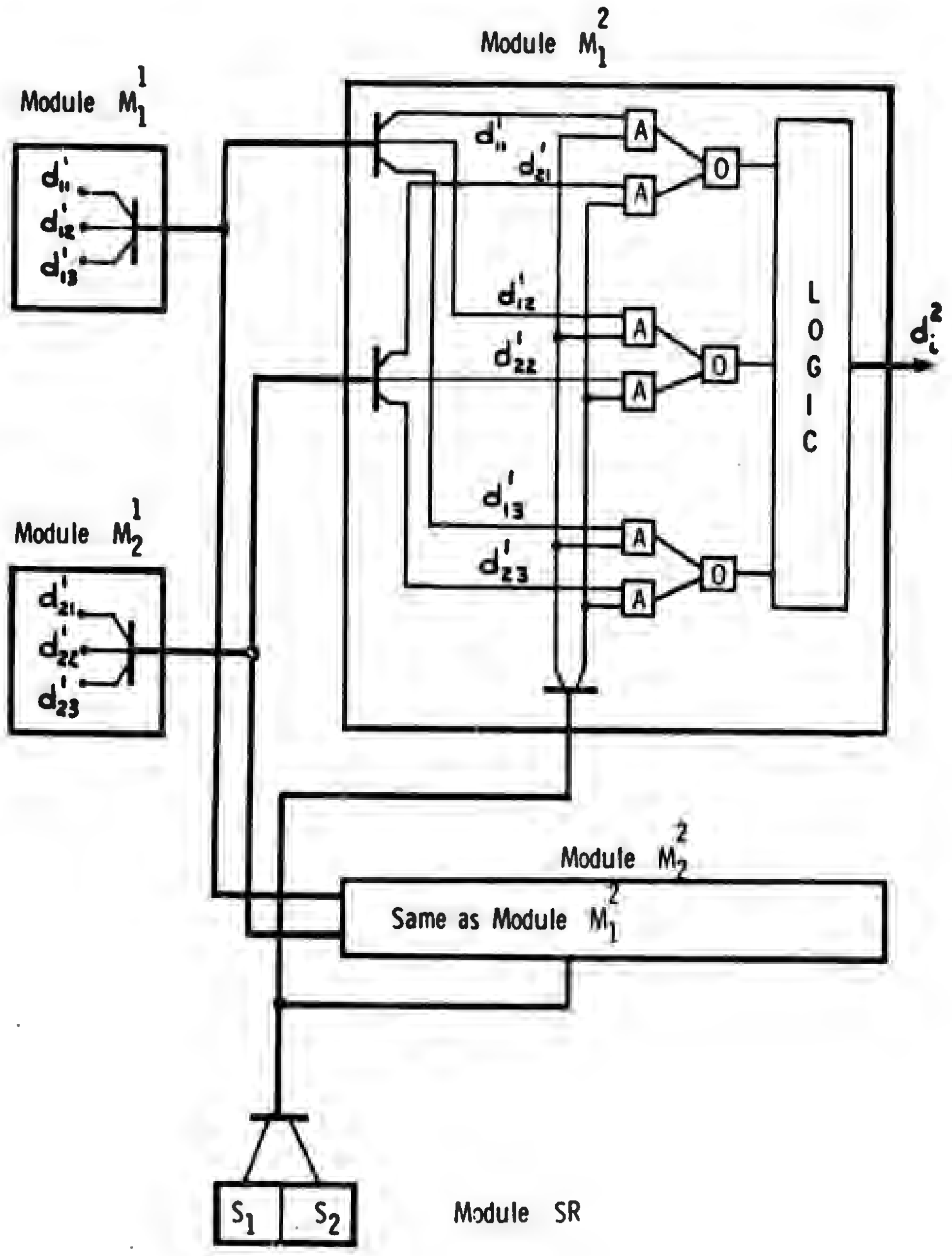


Figure 5.5 ARC 1 Reliability Calculation with Failure Tolerance.

5.4.2.3.2 Mapping Information from Buss to Module

The difference between buss interconnection and direct interconnection between units is that if a buss line goes down, all units are down, and no spares can be switched in. Thus to avoid hard core, there must be spare lines in the buss. The basic buss interconnection is shown in Figure 5.6 with three information lines and one spare.

The gating function for each information line I_j^1 (jth input information line to the first module) is

$$I_j^1 = b_j \bar{S}_j \vee b_{j+1} S_j$$

and the triggers S_j defining the states are shown below.

Buss Line Made Idle	S_1	S_2	S_3	State Trigger
b_4	0	0	0	
b_3	0	0	1	
b_2	0	1	1	
b_1	1	1	1	

If Eb_i stands for the signal showing the detection of an an error on buss-line b_i, then the setting functions for the S_i are as follows:

S_1 equals Eb_1

S_2 equals $Eb_1 \vee Eb_2$

S_3 equals $Eb_1 \vee Eb_2 \vee Eb_3$

The triggers S_j are initially reset.

A stuck at one error in the switch circuitry shown will result in the receiving device being put in a failed state.

A stuck at zero error can be circumvented by changing the buss status.

An error in the buss status register clearly limits the possible buss states.

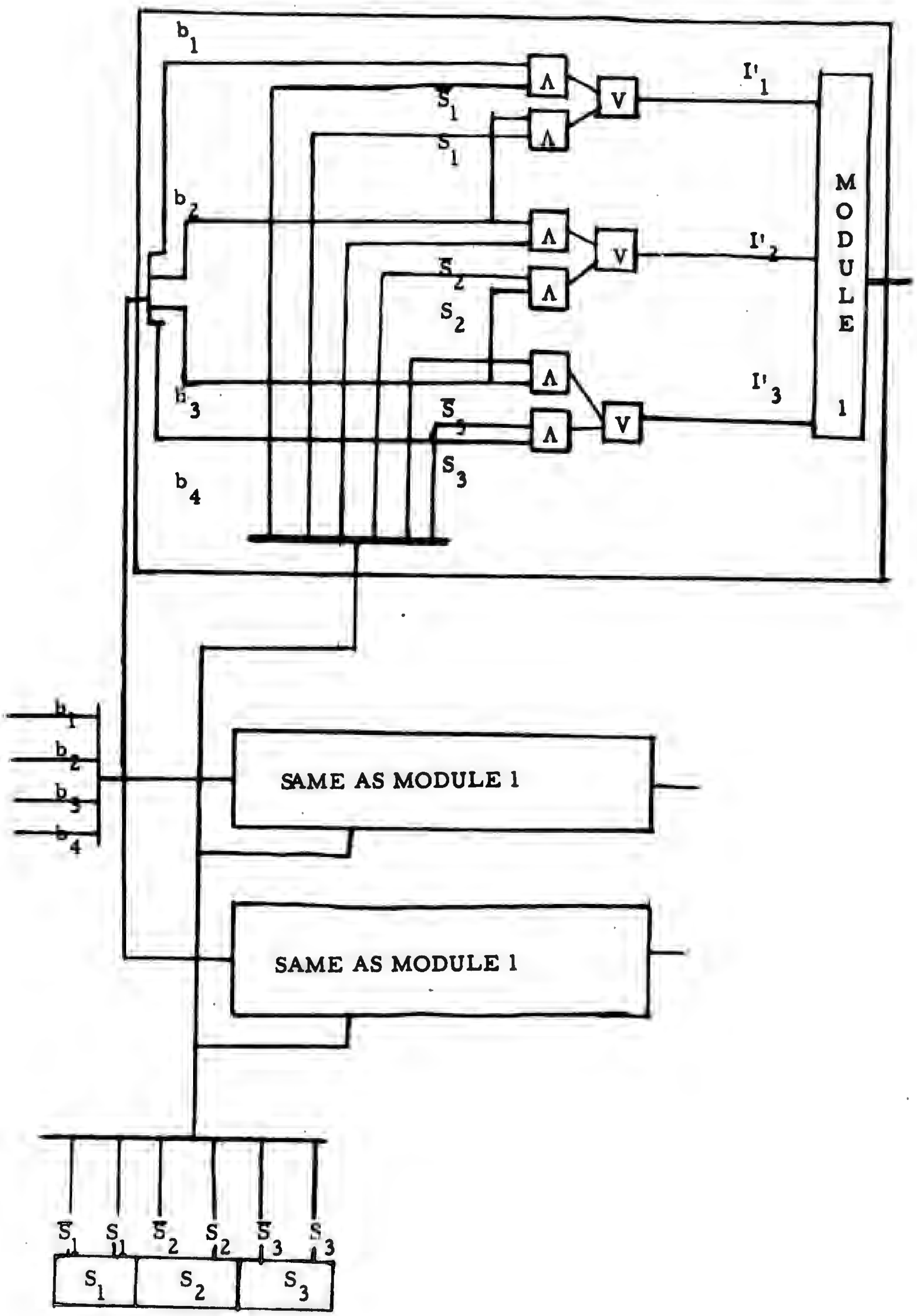


Figure 5.6 Buss-Module Switch

5.4.2.3.3 Mapping Information from Module to Buss

The basic method followed is different in this case. The basic rule is that each input to the buss switch must have no fan out.

A detailed logic structure is shown in the Figure 5.7 The buss lines are controlled as in the previous design. The device status register, SRD, is a three stage shift register with states 1, 0, 0; 0, 1, 0; 0, 0, 1. The output signals from switch 1 (SW1) are defined by

$$1_{j1} = D_j 1_{j1} \qquad J = 1, 2 \text{ or } 3 .$$

The signals 1_{ji} are duplicated to avoid fan out from a single logic gate, since, for example, 1_{11} is connected to both b_1 and b_2. A stuck at one error in SWB will incapacitate a buss line. A stuck at zero error in an AND gate can be compensated for by using another buss line or a different input unit. A stuck at zero error in an OR gate necessitates using another buss line.

A stuck at one error in a SW1 AND gate necessitates using another buss line. A stuck at zero error in a SW1 AND gate can be compensated for another buss line or a different input unit. An error on the output of a unit necessitates using another unit.

The following Figure 5.8.1 shows how the two input AND gates determining the flow of the same information (eg 1_{11}) can be elided to reduce the amount of circuitry.

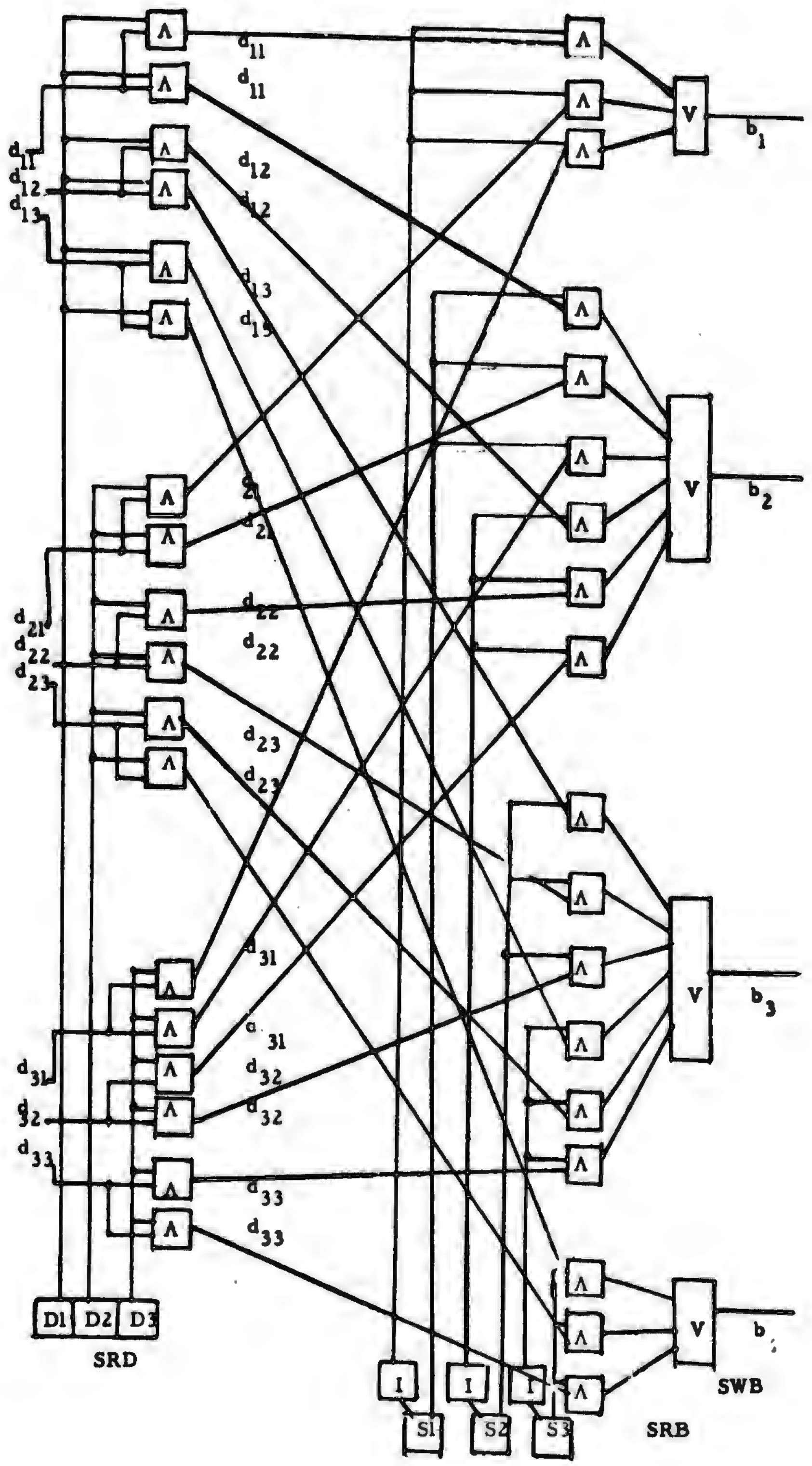
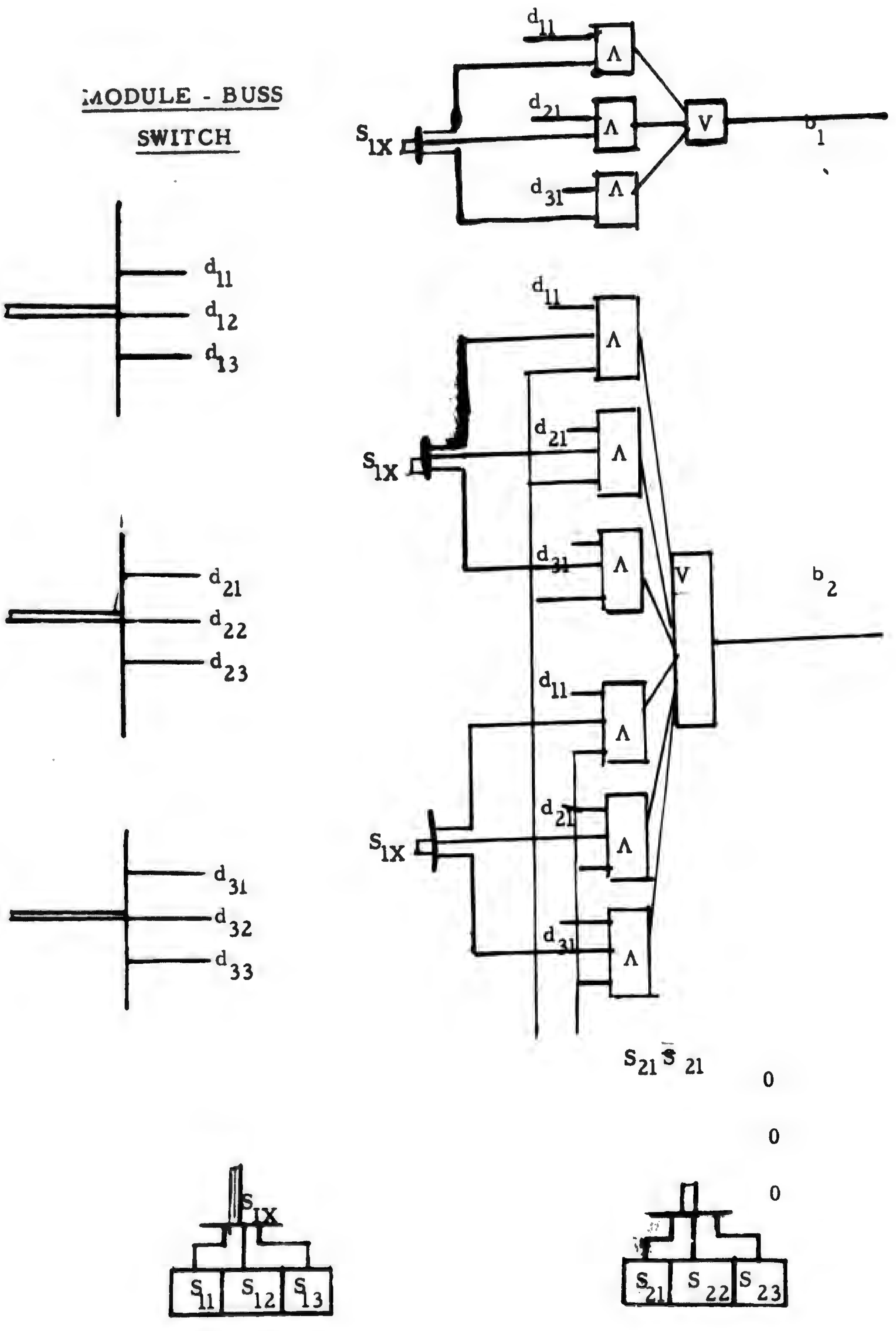


Figure 5.7 Module to Buss Switch.



MODULE - BUSS
SWITCH
d11
d12
d13
d21
d22
d23
d31
d32
d33
S1X
d11
d21
d31
Λ
V
b1
b2
S21 S̄21
0
0
0
S11
S12
S13
S21
S22
S23

Figure 5.8.1

A second configuration, with three spares, will illustrate the problems well.

For the purposes of the initial discussion of the reconfiguration it will be assumed that the width of the data buss is q and that $m = 3$ spare bit planes are included in the buss. This latter assumption is <u>not</u> a restriction to the scheme and modifications to cover other than 3 spare lines will be discussed after the initial presentation. These bits are used to record which line in the buss this particular data line is connected to; either its primary line or one of its three possible alternates.

Figure 5.8.2 illustrates the encoding scheme used in the status register (SR). The bits in the status register associated with data line a_i, denoted SR_{i1} and SR_{i2}, are shown inside the table at row i. During initial operation, when no solid failures have occurred, all entries in the SR are zero. This means data line a_i is connected to buss line b_i $i = 1, \ldots, q$. Now suppose buss line b_j has been found to have a solid failure. Then all status register positions $i > j$ have their entries changed from 00 to 01. This leaves $a_1, \ldots, a_{j-1}$ connected to $b_1, \ldots, b_{j-1}$ but connects $a_j, \ldots, a_q$ to $b_{j+1}, \ldots, b_{q+1}$, thus by-passing the failed line b_j.

Assume the second solid failure occurs in line b_k and, since the order of occurrence is not important as far as the resulting state of the ISR is concerned, assume $j < k$. Then the connections are:

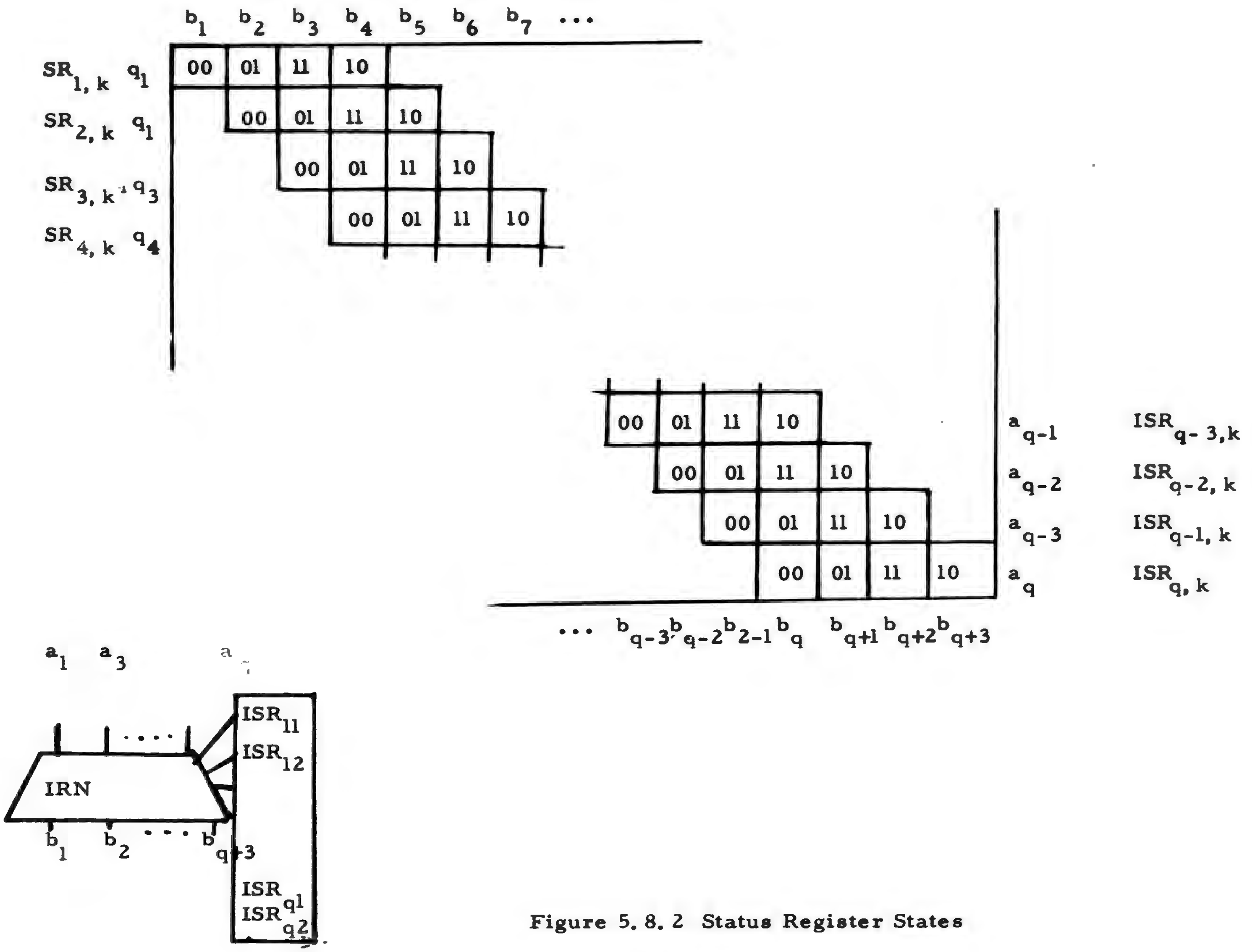


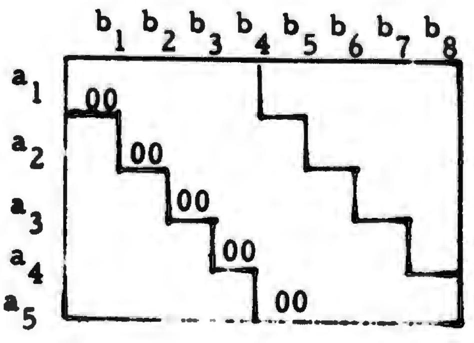
Figure 5. 8. 2 Status Register States

$$\begin{array}{lcl} a_1 \ ,\ldots, a_{j-1} & \leftrightarrow & b_1 \ ,\ldots, b_{j-1} \\ a_j \ ,\ldots, a_{k-2} & \leftrightarrow & b_{j+1},\ldots, b_{k-1} \\ a_{k-1},\ldots, a_q & \leftrightarrow & b_{k+1},\ldots, b_{q+2} \end{array}$$

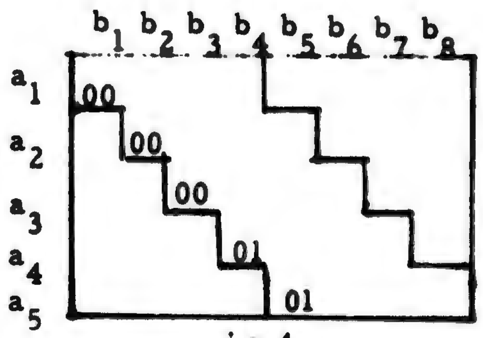
Finally assume b_ℓ fails with $j < k < \ell$. The new mappings become:

$$\begin{array}{lcl} a_1 \ ,\ldots, a_{j-1} & \leftrightarrow & b_1 \ ,\ldots, b_{j-1} \\ a_j \ ,\ldots, a_{k-2} & \leftrightarrow & b_{j+1},\ldots, b_{k-1} \\ a_{k-1},\ldots, a_{\ell-3} & \leftrightarrow & b_{k+1},\ldots, b_{\ell-1} \\ a_{\ell-2},\ldots, a_q & \leftrightarrow & b_{\ell+1},\ldots, b_{q+3} \end{array}$$

The sequence of status register contents and mappings for a buss with $q = 5$ and $m = 3$, subject to the failure sequence b_5, b_2, and b_5 is shown below.

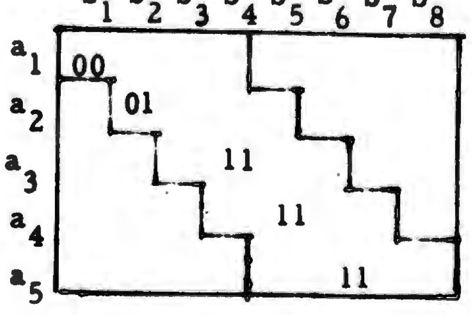


Initial



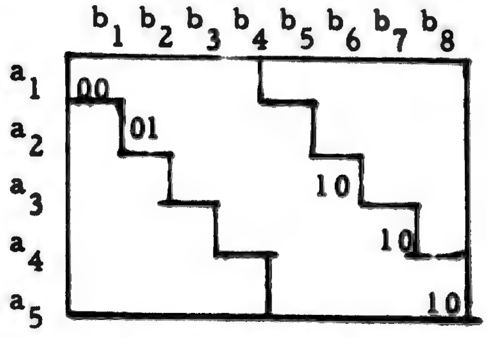
$j = 4$

After b_4 Fails



$j = 2$ $k = 4$

After b_2 and b_4 Fails



$j = 2$ $k = 4$ $= 5$

After b_2, b_4 and b_5 Fails

Although the preceding discussion was slanted towards the input mapping, an identical discussion is valid for the output since they are duals.

The switching in the network necessary to perform the mappings implied by the ISR is shown in Figures 5.8.3. In Figure 5.8.3 only the end cells of the network are shown; the cells for $b_5, \ldots, b_{q-1}$ are similar to the ones for b_4 and b_q. The logic was shown in terms of AND and OR gates in order to keep the networks as general as possible. However, conversion to established technologies provides no problem, e.g. in the usual transistor technologies both the AND and OR gates would be replaced by NAND gates with no resulting change in transfer function.

The sequence 00, 01, 11, 10 which was used in the status registers was selected only by convention. In any particular implementation the sequence could be changed with no loss in the overall system. Although using more bits in the status register than would theoretically be required to record all possible mappings, the reconfiguration scheme described here has the advantage that it reduces the total equipment used in <u>both</u> the status register and the reconfiguration switch. Thus while more efficient encodings exist, the resulting increase in hardware required to decode these states more than offsets any apparent gain. Further, only two levels of delay are inserted into the data paths. Finally, using a shift of connections to by-pass a failed line uses far less circuit fan-out and associated powering than is used by having three general purpose spares each of which can replace any of the planes $b_1, \ldots, b_q$.

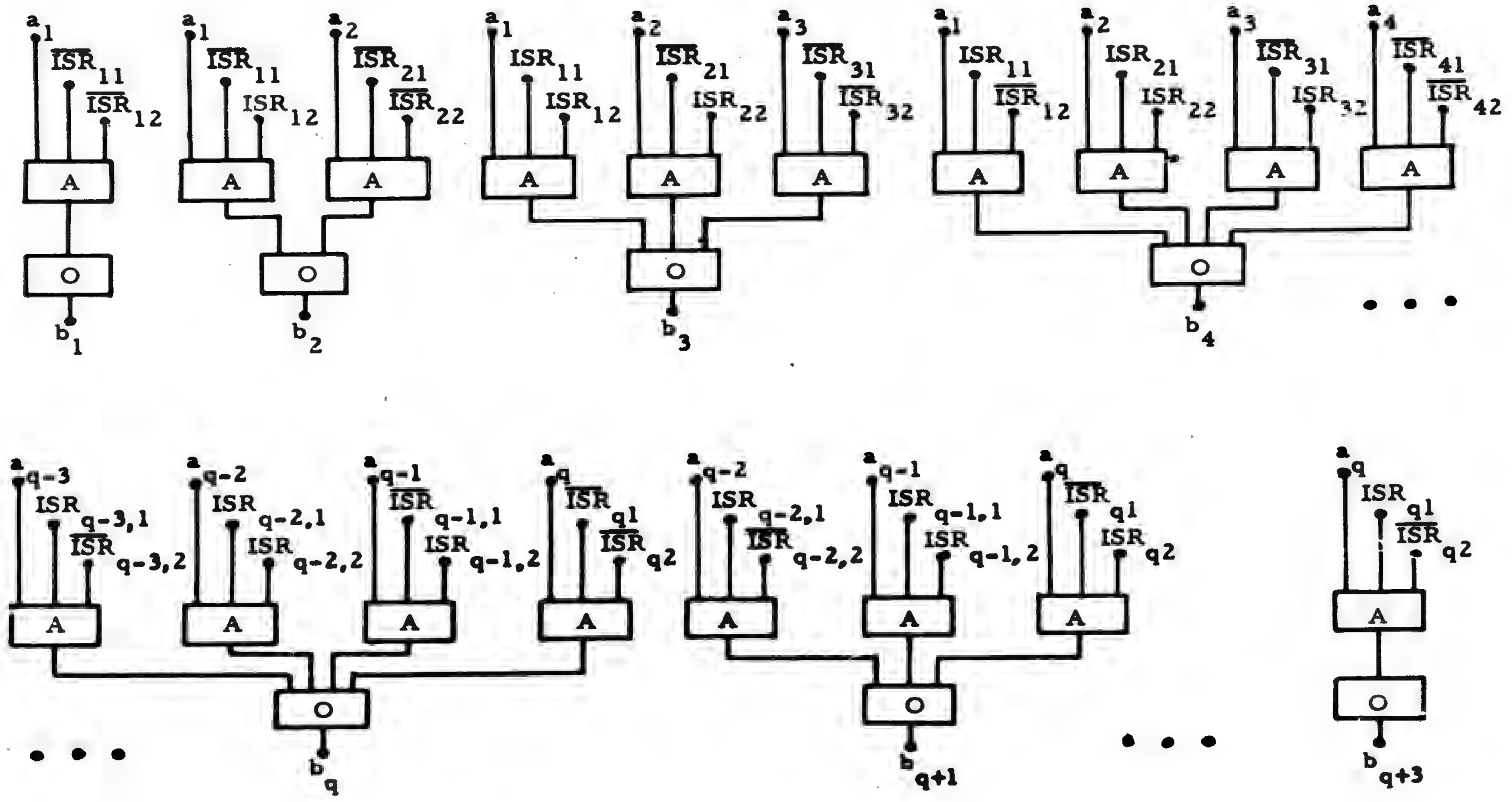


Fig. 5.8.3 Input Reconfiguration Network (IRN).

5.4.2.3.4 Generalizations to M Spares

For the general case with q data lines and m spares it is necessary to have $\beta = [\![\log_2(m+1)]\!]$ bits per data line in the status registers. As before, the sequence of states used to define the mapping of each data line to its initial bit plane and m possible alternate positions is determined by the particular hardware implementation, i.e. using an extension of the sequencing employed in section 2 one might select 000, 001, ..., 110, 111 for the sequence to be used with $m = 7$, $\beta = 3$.

5.4.2.4.2 Status Register for Module to Module Connection

The shift register will consist of two parts. The first is a core logic shift register using a "ladder" core technique. The shifting will be done on this core register, using a voting technique (each input line will supply 1/3 of the energy necessary to shift the register). Ladder Core technique has the property that as a 1 shifts from one core to the next, the first core is set to a 0 by the physical circuit (multi aperture cores are used). The cores will store the state of the status register. The second part consists of electronic flip flops which can read the information from the cores, store it, and provide this information for use in the electronic switch.

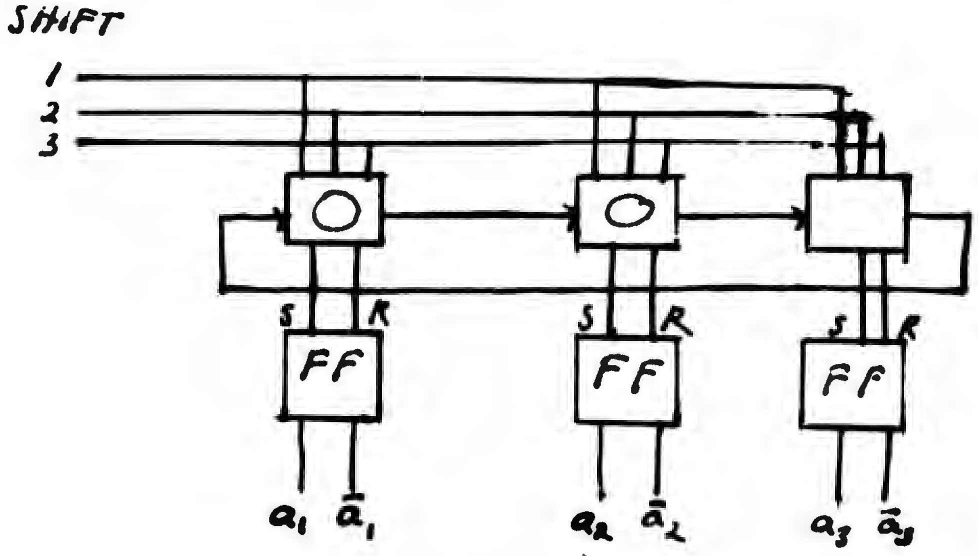
5.4.2.4 Error Correcting Design

5.4.2.4.1 General Method

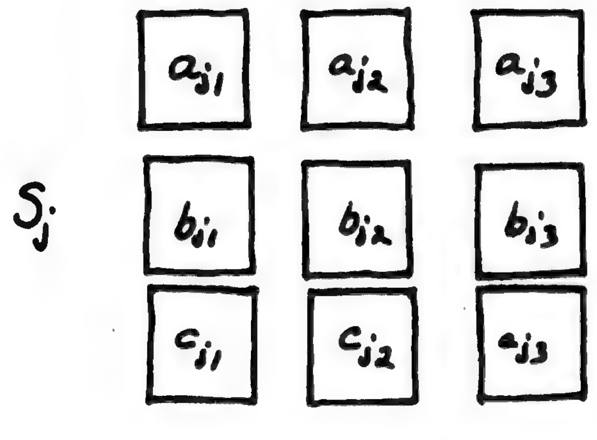
The switching units will be implemented using the redundant signals directly, so as to avoid fan out and the resulting hard core.

The implementation of status registers will be examined more closely. If they are TMR'd the reliability is satisfactory, but either much storage or much decoding is required. If error correcting codes are used, again storage may be traded off against decoding, or vice versa.

5.4.2.4.2.1 Initial Implementation An initial implementation uses TMR'd shift registers with the states 1, 0, 0; 0, 1, 0; 0, 0, 1. The initial setting line will set the state 1, 0, 0. The following drawings shows one of the TMR'd registers.



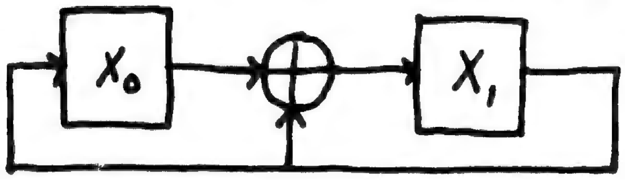
The 3 TMR'd registers will be written as



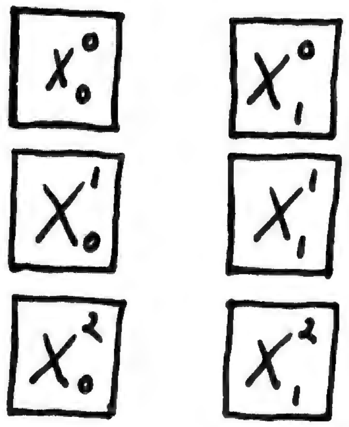
where j = 1, 2, 3 refer to the appropriate section. There are 9 output lines.

5.4.2.4.2.2 Second Possible Tradeoff

The two stage shift register with states 1, 0; 0, 1; 1, 1 may also be TMR'd. If X_0, X_1 are the stages and $\oplus$ is EOR, the implementation is as follows:



The three TMR'd registers may be written as:



There are six output lines, with the three states defined by TMRing the logical functions $X_0\overline{X}_1, \overline{X}_0 X_1$, $X_0 X_1$. Only six storage elements are used, and three EOR's.

5.4.2.4.2.3 Third Possible Tradeoff

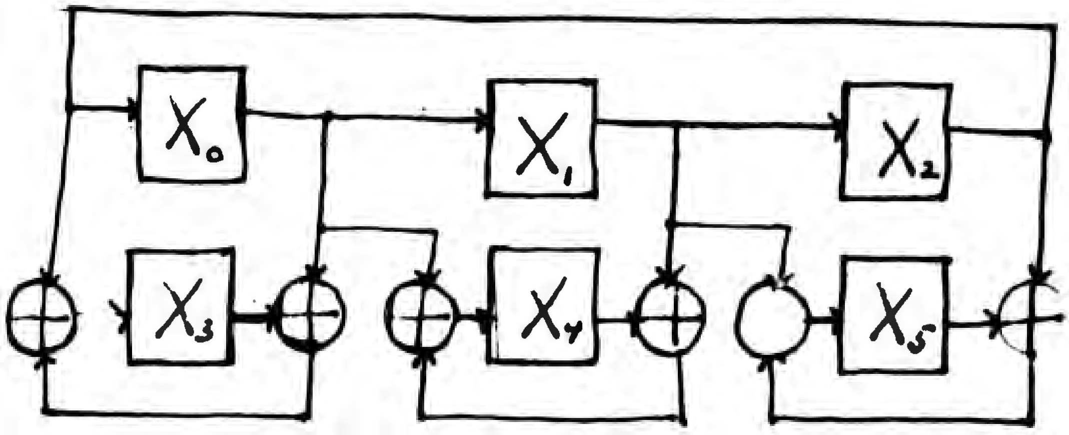
If the states 1,0,0; 0,1,0; 0,0,1 on digits X_0, X_1 and X_2 are taken as basic an error correcting shift register may be implemented by adding three parity check digits X_3 for X_0, X_4 for X_1 and X_5 for X_2. The valid code vectors using odd parity are:

X0	X1	X2	X3	X2	X5
1	0	0	0	1	1
0	1	0	1	0	1
0	0	1	1	1	0

and the parity check matrix (following Peterson) is

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

If $\oplus$ stands for exclusive OR (EOR) the shift register including correction digits is given by



The nine bits of TMR'd storage are replaced by six bits and six EOR's .

The parity checks are given by $pc_1 = X_0$ EOR X_3, $pc_2 = X_1$ EOR X_4 and $pc_2 = X_2$ EOR X_5

Let X_{ic} stand for the corrected signals X_i. The equations to produce these signals can be found from an examination of all possible single errors which may occur in the valid code vectors and the corresponding parity check signals. (The erroneous signals are underlined).

V	PC1	PC	PC
1 0 0 0 1 1	1	1	1
$\underline{0}$ 0 0 0 1 1	$\underline{0}$	1	1
1 $\underline{1}$ 0 0 1 1	1	$\underline{0}$	1
1 0 $\underline{1}$ 0 1 1	1	1	$\underline{0}$
1 0 0 $\underline{1}$ 1 1	$\underline{0}$	1	1
1 0 0 0 $\underline{0}$ 1	1	$\underline{0}$	1
1 0 0 0 1 $\underline{0}$	1	1	$\underline{0}$

V	PC1	PC	PC
0 1 0 1 0 1	1	1	1
$\underline{1}$ 1 0 1 0 1	$\underline{0}$	1	1
0 $\underline{0}$ 0 1 0 1	1	$\underline{0}$	1
0 1 $\underline{1}$ 1 0 1	1	1	$\underline{0}$
0 1 0 $\underline{0}$ 0 1	$\underline{0}$	1	1
0 1 0 1 $\underline{1}$ 1	1	$\underline{0}$	1
0 1 0 1 0 $\underline{0}$	1	1	$\underline{0}$

V	PC1	PC2	PC3
0 0 1 1 1 0	1	1	1
$\underline{1}$ 0 1 1 1 0	$\underline{0}$	1	1
0 $\underline{1}$ 1 1 1 0	1	$\underline{0}$	1
0 0 $\underline{0}$ 1 1 0	1	1	$\underline{0}$
0 0 1 $\underline{0}$ 1 0	$\underline{0}$	1	1
0 0 1 1 $\underline{0}$ 0	1	$\underline{0}$	1
0 0 1 1 1 $\underline{1}$	1	1	$\underline{0}$

$$X_0c = (X_0 \wedge pc\,1) \vee \overline{pc\,1}\ (\overline{X}_1 \wedge \overline{X}_2)$$
$$X_1c = (X_1 \wedge pc\,2) \vee \overline{pc\,2}\ (\overline{X}_0 \wedge \overline{X}_2)$$
$$X_2c = (X_2 \wedge pc\,3) \vee \overline{pc\,3}\ (\overline{X}_0 \wedge \overline{X}_1)$$
$$X_3c = (X_3 \wedge pc\,1) \vee \overline{pc\,1}\ (\overline{X}_4 \vee \overline{X}_5)$$
$$X_4c = (X_4 \wedge pc\,2) \vee \overline{pc\,2}\ (\overline{X}_3 \vee \overline{X}_5)$$
$$X_5c = (X_5 \wedge pc\,3) \vee \overline{pc\,3}\ (\overline{X}_3 \vee \overline{X}_4)$$

The signals to be used in the switch are:

$$X_{0c} = X_0 \overline{X}_3 \vee X_0 \overline{X}_1 \overline{X}_2 \vee \overline{X}_1 \overline{X}_2 \overline{X}_3$$
$$X_{1c} = X_1 \overline{X}_4 \vee \overline{X}_0 X_1 \overline{X}_2 \vee \overline{X}_0 \overline{X}_2 \overline{X}_4$$
$$X_{2c} = X_2 \overline{X}_5 \vee \overline{X}_0 \overline{X}_1 X_2 \vee \overline{X}_0 \overline{X}_1 \overline{X}_5$$

Then there are nine signals (three for each corrected signal) the same as the number from the TMR'd switch. A double error may be detected if desired.

5.4.2.4.2.4 Fourth Possible Tradeoff

If a single two stage shift register with states 1, 0; 0, 1; 1, 1 is used, then three parity check digits may be added. The correction equations become much more complicated. For example with check digits X_2, X_3,

$$X_{0c} = X_0\bar{X}_2 \vee X_0\bar{X}_1\bar{X}_4 \vee X_0X_1X_4 \vee \bar{X}_1\bar{X}_2\bar{X}_4 \vee X_1X_2X_4$$

The small gain in storage (1 bit) is submerged in the extra switching.

5.4.2.4.3 Switch for Module to Module Connection

For a TMR'd status register call the input lines a_{11} through c_{13} as defined previously. Figure 5.9shows the details of the logic structure. Clearly a single error in each stage of the TMR'd status register can be tolerated if there is no error in the switch. Any error in the OR gates or a stuck at 1 in the AND gates will cause the unit to have to be switched. A stuck at 0 will cause no problems until there is an error in the status register, where 2/3 of the time it will cause the units to be switched. The terms a_{11}, b_{11}, c_{11}, etc. may be replaced by any other terms defining state 1 in an error correcting way. For example a_{11} by $X_0^0\bar{X}_1^0$, b_{11} by $X_0^1\bar{X}_1^1$, c_{11} by $X_0^2\bar{X}_1^2$ in which case five input AND gates would be needed or a good voter circuit.

In another implementation a_{11} could be replaced by $X_0\bar{X}_3$, b_{11} by $X_0\bar{X}_1\bar{X}_2$ and c_{11} by $\bar{X}_1\bar{X}_2\bar{X}_3$ etc. This requires four input AND gates. There are many choices,all implementation dependent.

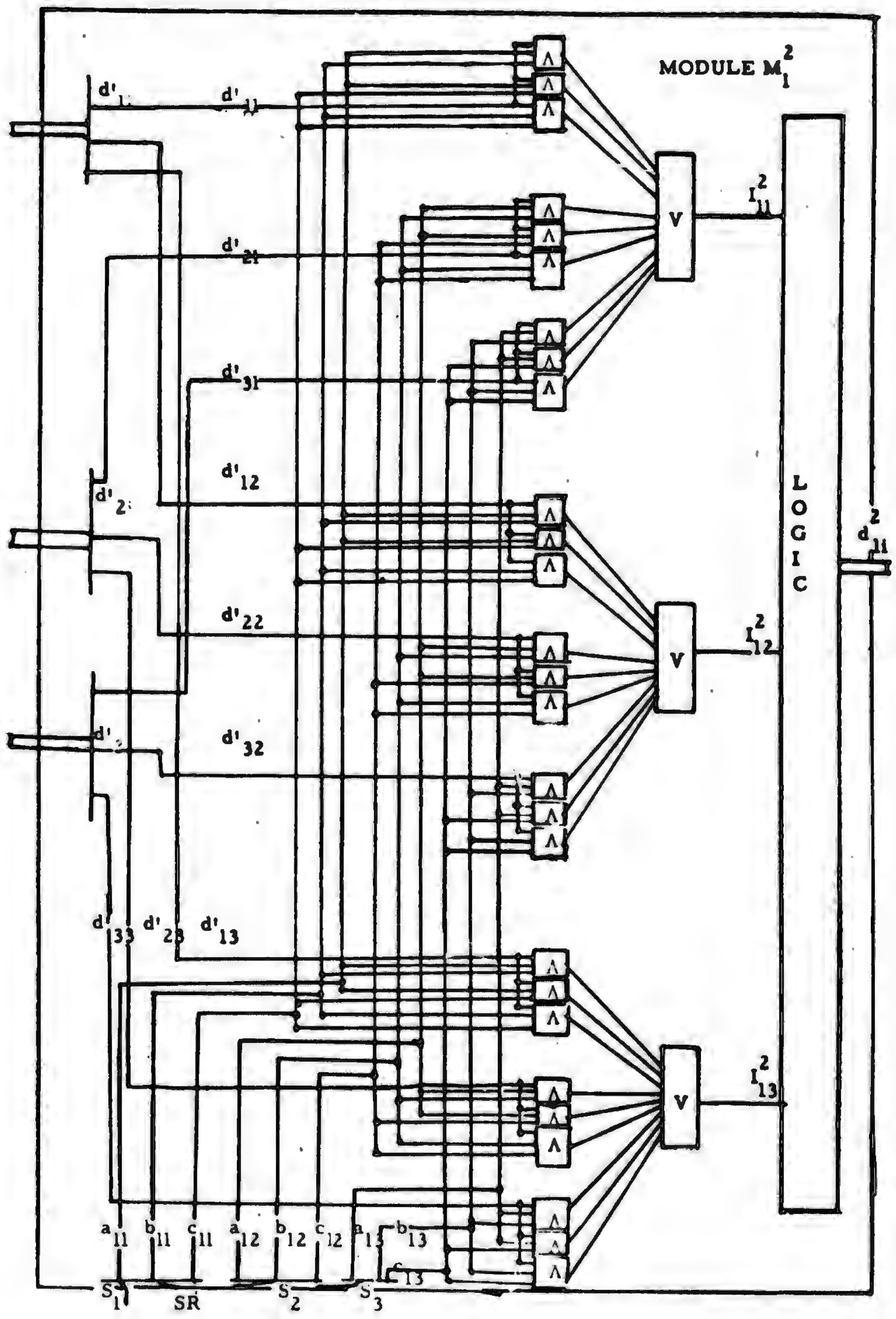


Figure 5.9 Module-Module Switch with TMR Status Register.

5.4.2.4.4 Status Register Switching Buss Lines with Three Spares to a Module

Let there be w information lines. Each information line i_j may be connected to one of four buss lines b_j, b_{j+1}, b_{j+2}, b_{j+3}. If the first error is in buss line b_j then the information lines i_{j+k}, $k=0,\ldots,w-j$ must be connected to buss lines b_{j+k+1}. Let the second error be in buss line b_ν if $\nu < j-1$, then lines $i_{\nu+k}$, $k=0,\ldots,j-\nu-2$ must be connected to lines $b_{\nu+k+1}$ while lines i_{j+k}, $k=-1,0,1,\ldots,w-j$ must be connected to lines b_{j+k+2}. If $\nu=j-1$, then lines i_{j+k}, $k=-1,0,1,\ldots,w-j$ must be connected to lines b_{j+k+2}. If $\nu \geq j$, then all lines $i_{\nu+k}$, $k=0,\ldots,w-j$ must be connected to $b_{\nu+k+2}$. The cases may be summarized in the following table:

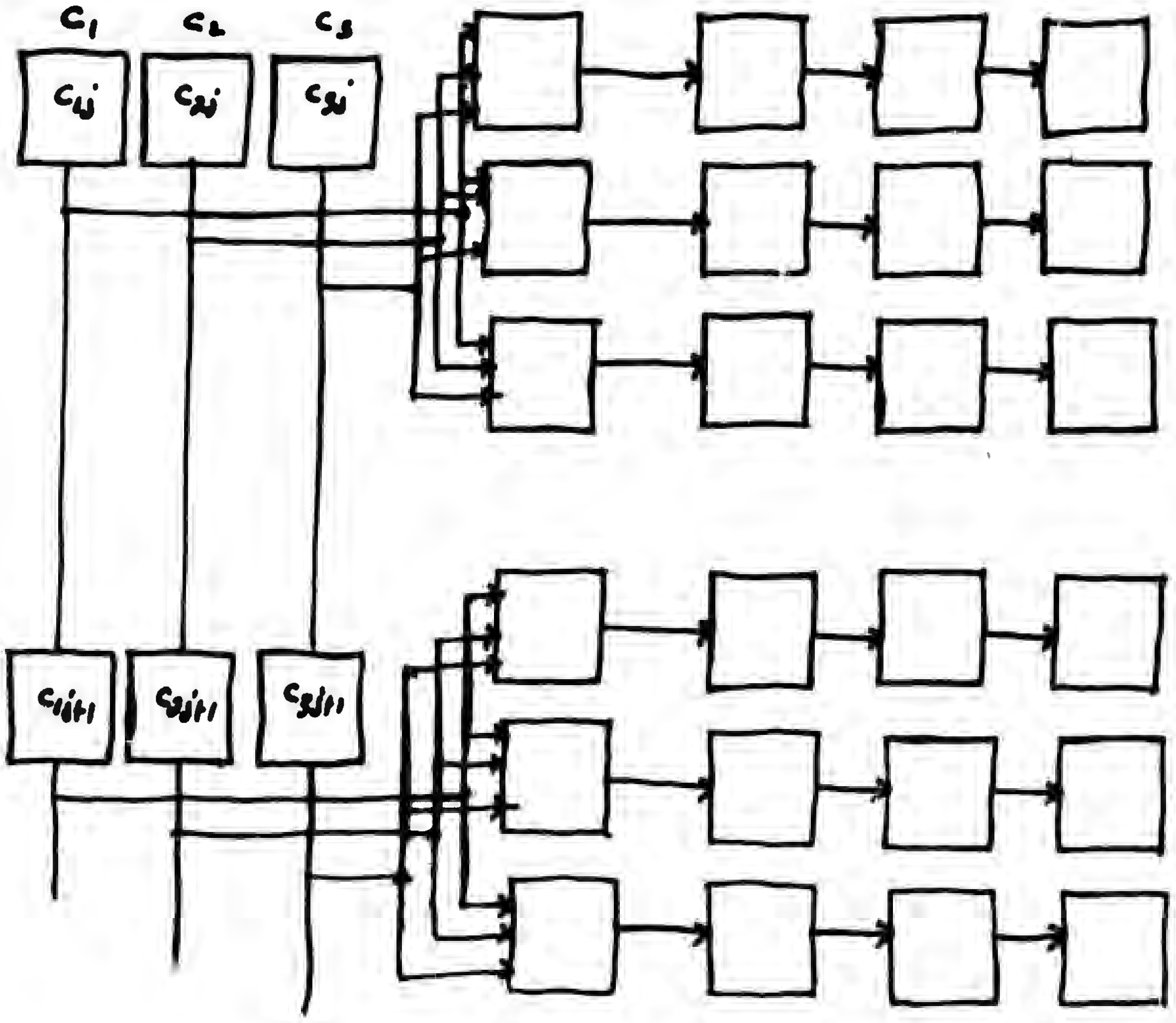
Information Lines i_j		Connected Buss Lines b_j	
No Error	Error b_j	Errors b_j, b_ν	
		$\nu \leq j-1$	$\nu \geq j$
i_α ; b_α	$\alpha < j$; b_α	$\alpha < \nu$; b_α	$\alpha < j$; b_α
	$\alpha \geq j$; $b_{\alpha+1}$	$\nu \leq \alpha < j-1$; $b_{\alpha+1}$	$j \leq \alpha < \nu-1$; $b_{\alpha+1}$
		$j-1 \leq \alpha$; $b_{\alpha+2}$	$\nu-1 \leq \alpha$; $b_{\alpha+2}$

Errors

b_j, b_ν, b_μ (assume $j < \nu$ without loss of generality)

$\mu \leq j-1$	$j \leq \mu \leq \nu-1$	$j < \nu < \mu$
$\alpha < \mu$; b_α	$\alpha \leq j-1$; b_α	$\alpha \leq j-1$; b_α
$\mu \leq \alpha < j-1$; $b_{\alpha+1}$	$j \leq \alpha < \mu-1$; $b_{\alpha+1}$	$j \leq \alpha < \nu-1$; $b_{\alpha+1}$
$j-1 \leq \alpha < \nu-2$; $b_{\alpha+2}$	$\mu-1 \leq \alpha < \nu-2$; $b_{\alpha+2}$	$\nu-1 \leq \alpha < \mu-2$; $b_{\alpha+2}$
$\nu-2 \leq \alpha$; $b_{\alpha+3}$	$\nu-2 \leq \alpha$; $b_{\alpha+3}$	$\mu-2 \leq \alpha$; $b_{\alpha+3}$

The status registers will be made of multi-aperture cores as before, and there must be four possible correct states of each register. If there are w information lines in the buss, then there will be w such status registers. The shift signals for the w registers will be derived from three registers c_1, c_2 and c_3 each of length w (these registers will have other uses). A one in the j^{th} position in c_1, c_2, c_3 will send shift signals to the 3 j status registers. See the drawing below (for the status register with states 1, 0, 0, 0; 0, 1, 0, 0; 0, 0, 1, 0; 0, 0, 0, 1).



The status registers will be set to their initial state. The notation SR_i means the ith in this set of status registers. The shifting of the status registers will be controlled by a microprogram in the ROS, and by the contents of those (triplexed) registers l_0, l_1, and l_2. When information line i_j is determined to be in error because a buss line is down, the number of the information line, j, is put into the register l_0 and control is transferred to a micro-program in the ROS. The work to be done is the following:

First error	Error is at location j, shift all registers SR_y where $y \geq j$. Store j.
Second error	Error is at location ν, shift all registers SR_y where $y \geq \nu$. Let $z = \max(\nu, j) - 1$, shift SR_z. Store ν, j in order of magnitude as b_1, b_2.
Third error	Error is at location μ. Shift all registers SR_y where $y \geq \mu$. Let $z = \max(b_1, \mu) - 1$. Shift SR_z. Let $t = \max(b_2 \mu) - 2$. Shift SR_t.

A possible microprogram to perform the switching of buss status registers will now be given. The contents of l_0, l_1 and l_2 at the end of each error correction step will be given below.

	Step 1	Step 2	Step 3
l_0	j	ν	μ
l_1	j	b_1	$\max(b_1, \mu)$
l_2	j	b_2	$\max(b_2, \mu)$

C(X) means the contents of memory location X. The following information will be stored in w bit words in the ROS.

S01	1 0 0 . . . 0
S02	0 1 0 . . . 0
S0w	0 0 0 . . . 1
SD1	1 1 1 . . . 1
SD2	0 1 1 . . . 1
SD3	0 0 1 . . . 1
SDw	0 0 0 . . . 0 1

```
T1    Transfer C(SD1_0) into c_1, c_2 and c_3, shifting
      the appropriate set of shift registers
         C(SRW) = 0010 → go to T2
         C(SRW) = 0010 → go to T3
         C(SRW) = 0001 → go to T6
T2    $1_1 \leftarrow c(1_0)$, $1_2 \leftarrow c(1_0)$, go to END
T3    $c(1_0) \leq c(1_1)$, go to T4
      $c(1_0) > c(1_1)$, go to T5
T4    $1_1 \leftarrow c(1_0)$, go to T9
T5    $1_2 \leftarrow c(1_0)$, to to T9
T6    $c(1_0) \leq c(1_1)$, to to T10
      $c(1_1) < c(1_0) < c(1_2)$, go to T7
      $c(1_2) < c(1_0)$, go to T8
T7    $1_1 \leftarrow c(1_0)$, go to T10
T8    $1_1 \leftarrow c(1_0)$, $1_2 \leftarrow c(1_0)$, go to T10
T9    $x \leftarrow c(SD1_2) - 1$
         Transfer C(SD0_x) into c_1, c_2, c_3; shift the
      appropriate shift register, go to END.
```

```
T10    $x \leftarrow c(SD1_1) - 1$
       Transfer $c(SD0_x)$ into $c_1, c_2, c_3$ and effect shift,
           go to T11
T11    $x \leftarrow c(SD1_2) - 1$, go to T12
T12    $x \leftarrow x - 1$
       Transfer $c(SD0_x)$ into $c_1, c_2, c_3$ and effect shift,
           go to END.
```

5.4.2.4.5 Status Register Implementation for a Buss

Since each information line may be connected to one of four buss lines, if there are w information lines 4w states must be identified.

A first implementation is with a four stage shift register, each of whose states is TMR'd. The states are defined by 1, 0, 0, 0; 0, 1, 0, 0; 0, 0, 1, 0; and 0, 0, 0, 1. This design simplifies the switch for the buss to device or device to transfer, uses 12w stages of shift register storage for w information lines.

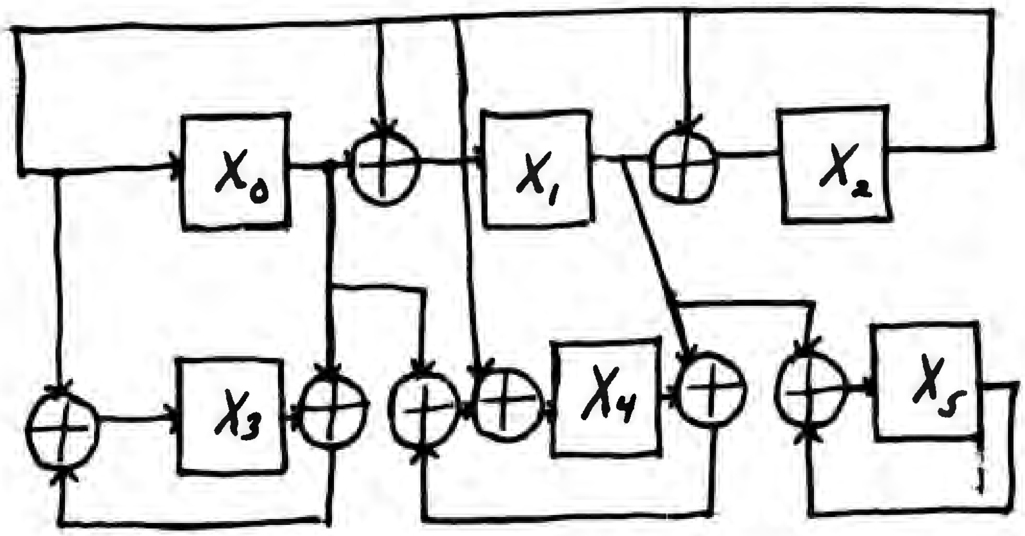
A second implementation notation is to consider the three stage shift register (x_0, x_1, x_2) with the following four states: 1, 0, 0; 0, 1, 0; 0, 0, 1; 1, 1, 1 . Add check digits x_3, x_4 and x_5 and the following parity checks:

$$pc_1 = x_0 \text{ EOR } x_3$$
$$pc_2 = x_1 \text{ EOR } x_4$$
$$pc_3 = x_2 \text{ EOR } x_5$$

If odd parity is used the following code vectors are valid:

1	0	0	0	1	1
0	1	0	1	0	1
0	0	1	1	1	0
1	1	1	0	0	0

A shift register to give these vectors is the following, using $\oplus$ for EOR.



The initial state must be 1, 0, 0, 0, 1, 1 . Six storage elements and eight EOR logic devices must be used.

As before the corrected signals may be written as x_{ic} and their equations derived from a study of the possible single error vectors.

$$x_{0c} = (x_0 \; pc_1) \vee \overline{pc}_1 (x_1 x_2 \vee \bar{x}_1 \bar{x}_2)$$
$$x_{1c} = (x_1 \; pc_2) \vee \overline{pc}_2 (x_0 x_2 \vee \bar{x}_0 \bar{x}_2)$$
$$x_{2c} = (x_2 \; pc_3) \vee \overline{pc}_3 (x_0 x_1 \vee \bar{x}_0 \bar{x}_1)$$
$$x_{0c} = x_0 \bar{x}_3 \vee x_0 x_1 x_2 \vee x_0 \bar{x}_1 \bar{x}_2 \vee x_1 x_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3$$
$$x_{1c} = x_1 \bar{x}_4 \vee x_0 x_1 x_2 \vee \bar{x}_0 x_1 \bar{x}_2 \vee x_0 x_2 \bar{x}_4 \vee \bar{x}_0 \bar{x}_2 \bar{x}_4$$
$$x_{2c} = x_2 \bar{x}_5 \vee x_0 x_1 x_2 \vee \bar{x}_0 \bar{x}_1 x_2 \vee x_0 x_1 \bar{x}_5 \vee \bar{x}_0 \bar{x}_1 \bar{x}_5$$

In this case, the correction signals are more complicated than the TMR'd case, but the storage requirements are smaller.

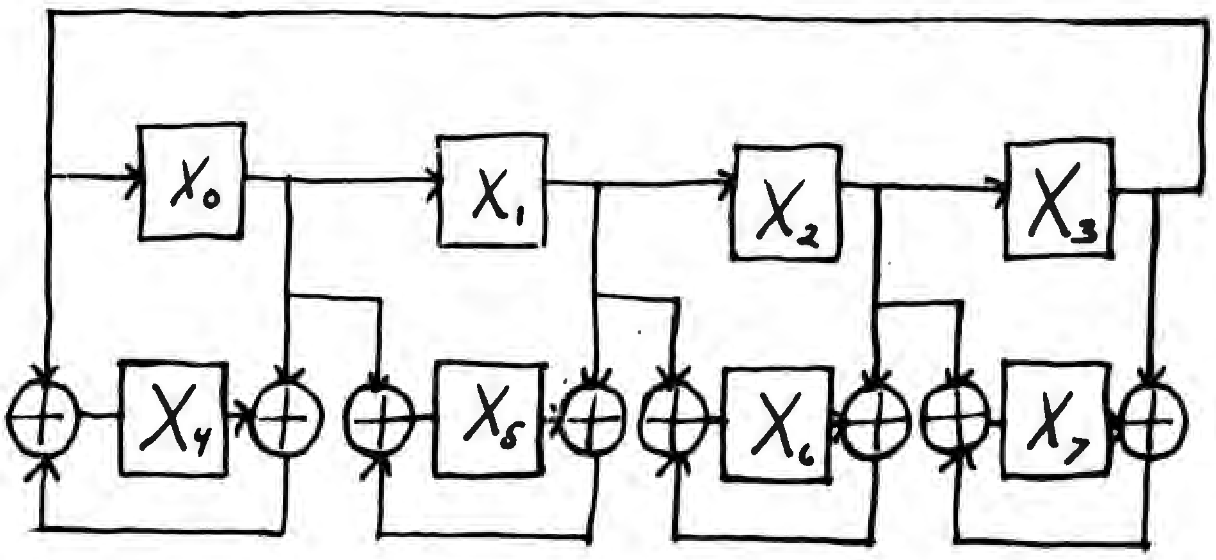
A third implementation is to consider the four stage shift register (x_0, x_1, x_2, x_3) with the following four stages 1, 0, 0, 0; 0, 1, 0, 0; 0, 0, 1, 0; 0, 0, 0, 1. Add check digits x_4, x_5, x_6, x_7 and the following parity checks

$$pc_1 = x_0 \text{ EOR } x_4$$
$$pc_2 = x_1 \text{ EOR } x_5$$
$$pc_3 = x_2 \text{ EOR } x_6$$
$$pc_4 = x_3 \text{ EOR } x_7$$

(no lesser number of stages gives an appreciable saving).

The valid code vectors are the following:

1	0	0	0	0	1	1	1
0	1	0	0	1	0	1	1
0	0	1	0	1	1	0	1
0	0	0	1	1	1	1	0



The initial storage status must be 1, 0, 0, 0, 0, 1, 1, 1.

The corrected signals are as follows:

$$x_{0c} = x_0 pc_1 \ \mathrm{V} \ \overline{pc}_1 (\bar{x}_1 \bar{x}_2 \bar{x}_3)$$
$$x_{1x} = x_1 pc_2 \ \mathrm{V} \ \overline{pc}_2 (\bar{x}_0 \bar{x}_2 \bar{x}_3)$$
$$x_{2c} = x_2 pc_3 \ \mathrm{V} \ \overline{pc}_3 (\bar{x}_0 \bar{x}_1 \bar{x}_3)$$
$$x_{3c} = x_3 pc_4 \ \mathrm{V} \ \overline{pc}_4 (\bar{x}_0 \bar{x}_1 \bar{x}_2)$$
$$x_{0c} = x_0 \bar{x}_4 \ \mathrm{V} \ x_0 \bar{x}_1 \bar{x}_2 \bar{x}_3 \ \mathrm{V} \ \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$$
$$x_{1c} = x_1 \bar{x}_5 \ \mathrm{V} \ \bar{x}_0 x_1 \bar{x}_2 \bar{x}_3 \ \mathrm{V} \ \bar{x}_0 \bar{x}_2 \bar{x}_3 \bar{x}_5$$
$$x_{2c} = x_2 \bar{x}_6 \ \mathrm{V} \ \bar{x}_0 \bar{x}_1 x_2 \bar{x}_3 \ \mathrm{V} \ \bar{x}_0 \bar{x}_1 \bar{x}_3 \bar{x}_6$$
$$x_{3c} = x_3 \bar{x}_7 \ \mathrm{V} \ \bar{x}_0 \bar{x}_1 \bar{x}_2 x_3 \ \mathrm{V} \ \bar{x}_0 \bar{x}_1 \bar{x}_2 \bar{x}7$$

The implementation using least storage is a TMR'd two stage ring counter. The states are defined by 0, 0; 1, 0; 0, 1; 1, 1. This uses 6 w stages of storage for w information bits.

5.4.2.4.6 Switch for Switching Buss Lines with Three Spares to a Module

Each element of the switch is straightforward (given the previous work). The only problem is the tradeoffs involved. Consider the j^{th} information line and the four buss lines b_j, b_{j+1}, b_{j+2}, b_{j+3}, which may be connected to it. Each buss line may be connected to three other information lines (b_j to i_{j-3}, i_{j-2} or i_{j-1} etc. for all positive indices). The buss fan outs will not be shown. In addition each buss line must go to all devices to which it is attached.

Let the j^{th} TMR'd status register be written as:

	a_j^1	a_j^2	a_j^3	a_j^4
SR_j	b_j^1	b_j^2	b_j^3	b_j^4
	c_j^1	c_j^2	c_j^3	c_j^4

The necessary logic detail for the switch is shown in Figure 5.10.

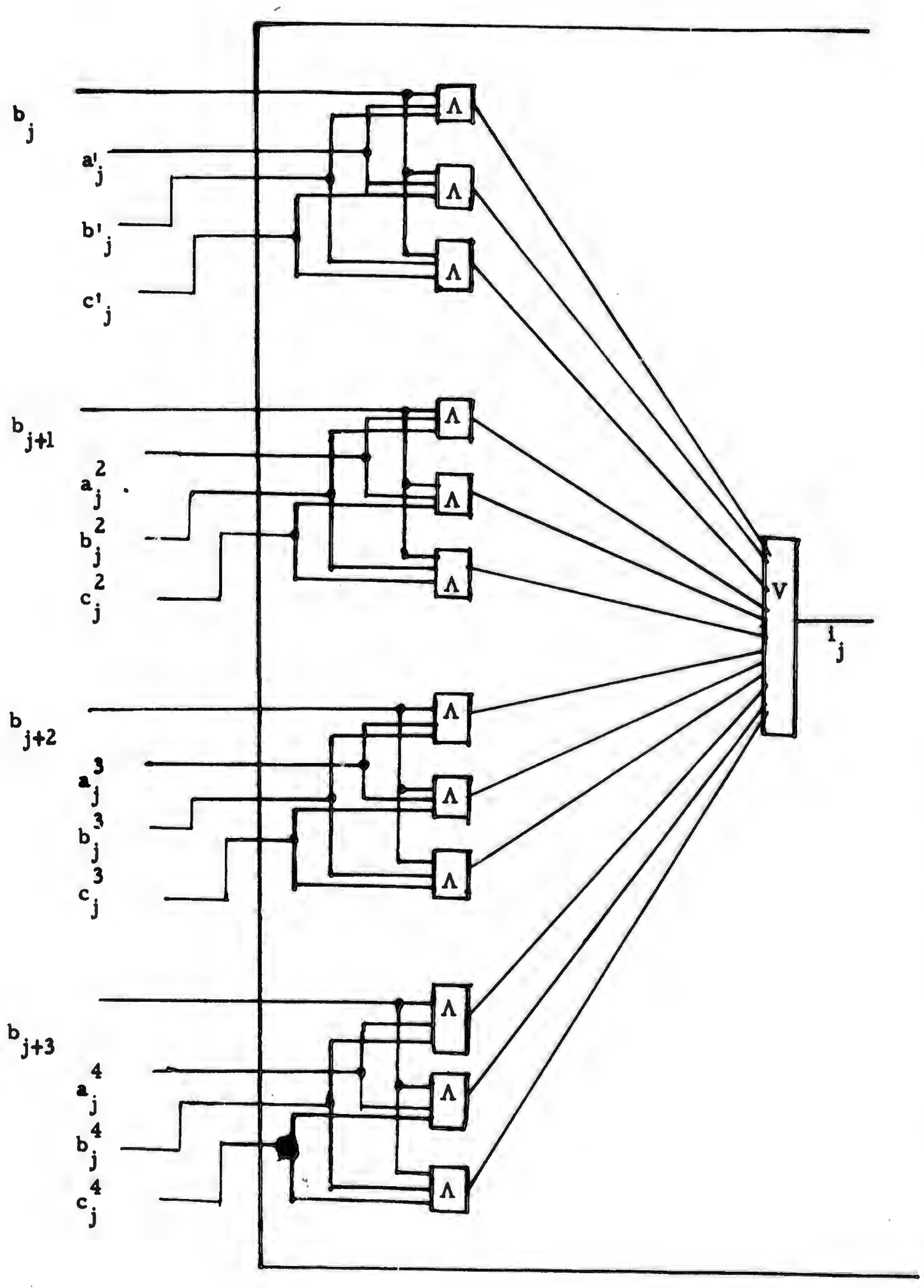


Figure 5.10

As before, a single error can be tolerated in each stage of each SR_j if there is no error in the switch. An error in the OR block or a stuck at 1 in an AND block will cause the unit to be down. A stuck at 0 in an AND block will cause no error if there is no error in the status register stage. If there is an error in the status register stage there is a 2/3 chance that this will cause an error in the device. This can be corrected by using an alternate buss line or by using another module.

If the three stage shift register is used for a buss status register, then the 3 three-wayAND blocks at each line b_j must be replaced by five four way AND blocks.

If the four stage shift register is used for a buss status register then the 3 three-way AND blocks at each line b_j must be replaced by 3 five-way and blocks (or however they are implemented).

If the TMR'd two stage ring counter is used then a_j^1 is replaced $x_0^0 \bar{x}_1^0$, b_j^1 by $x_0^1 \bar{x}_1^1$ and c_j^1 by $x_0^2 \bar{x}_1^2$. This direct implementation needs 3 five-way AND circuits - however better TMRing can probably be done. The savings in storage (50%) is impressive. However, only one error per shift register, not one error per stage, can be tolerated.

Tradeoffs will depend upon technology, circuit implementation, and possible reliability requirements. Seven extra buss lines could be handled with six bits of storage per information line, but plenty of switching.

5.4.2.4.7 Switch for Switching a Module to a Buss Line with Three Spares

In order to avoid a hard core this takes lots of c circuits. Figure 7 will show one information line from one of these devices and this line will connect with one of four buss lines as labeled. Both the device and buss status registers will be assumed to be TMR'd single shift registers.

Each output line 1_{ij} from the device now drives four identical sets of 3 three-way AND gates (12 in all). The input to the buss line consists of ORing together the output from four sets of lines with the activated set of controls from TMR'd Buss Status Input defining one of four states. (See Figure 5.11)

The inputs to b_4 consist of TMR'd lines 1_{14} from line 4, unit 1, TMR'd lines 1_{24} from line 4 unit 2; and TMR'd lines 1_{34} from line 4 unit 3. In state 1, any inputs from these lines go out onto the buss. The inputs which are used depend upon the state of the module status register. In state 2, inputs come from line 3 from the appropriate unit; in state 3 from line 2 and in state 4 from line 1.

The possibilities for tradeoffs are many, and depend upon the technology. The problem is to avoid creating hard core by fanning out to several modules. In the diagram shown, an error in 1_{14} output line from module 1 or 1_{14} input line to the buss would result in 1_{24} being used, and so unit 2 will be activated. An error on b_4 would result in b_4 being skipped and 1_{14}, 1_{24} and 1_{34} being attached to buss b_5 (whose status register would then be in state 2).

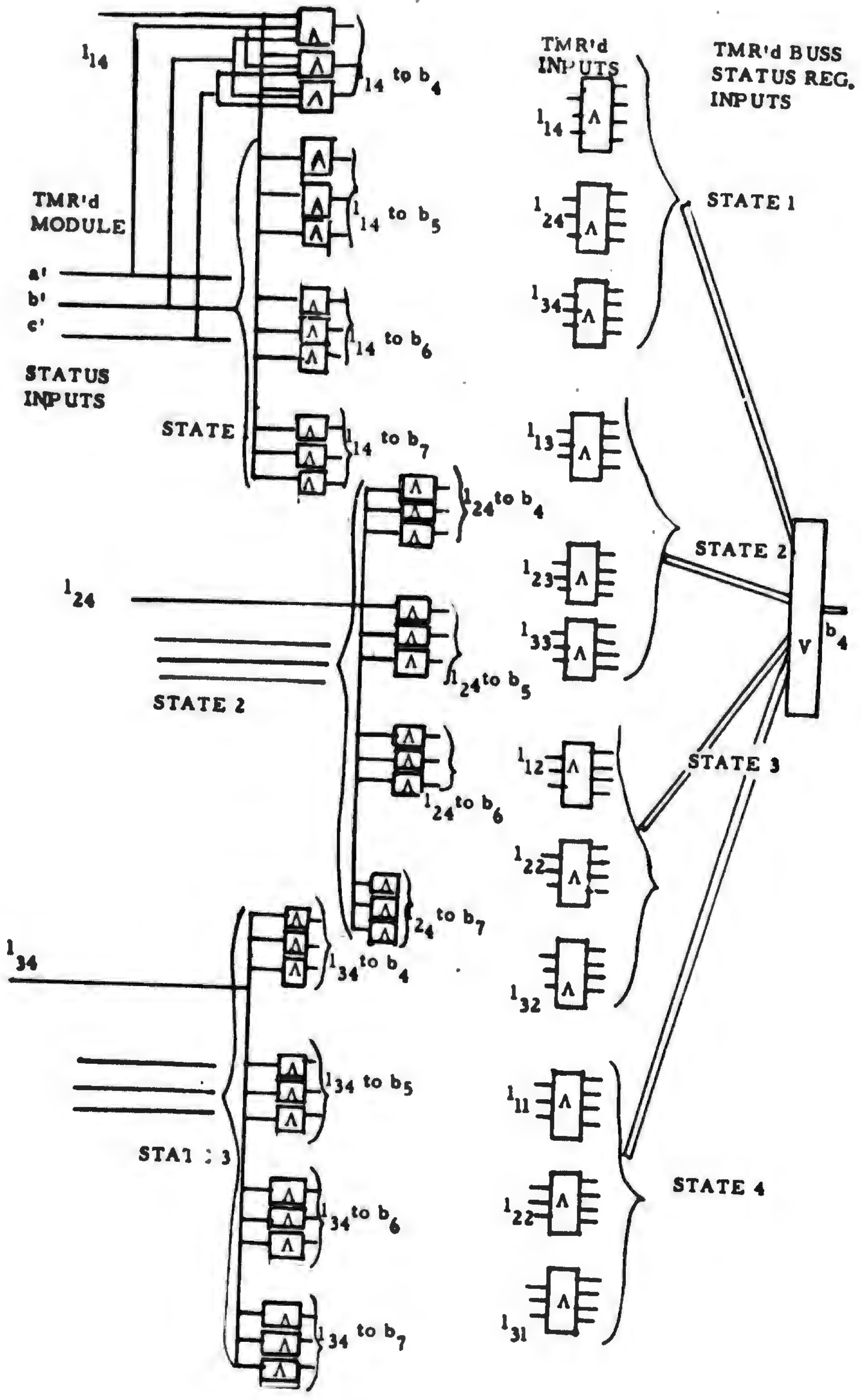
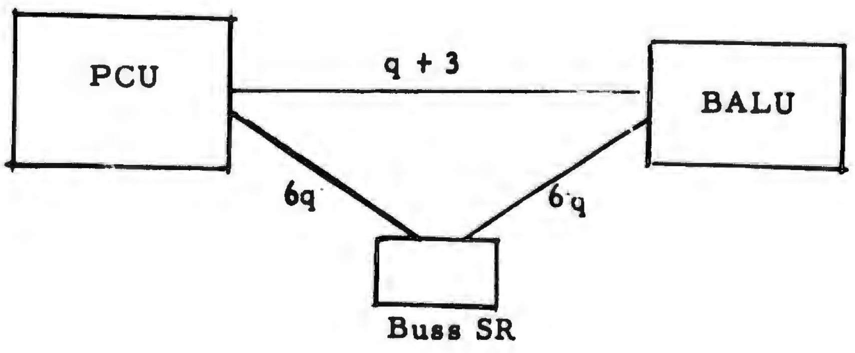


Figure 5.11 Switching Module to Buss with 3 Spares (1 Information Line).

5.4.2.4.8 The Buss Line Number Problem

This design, if implemented directly would need the number of buss lines shown in the diagram below.



Clearly this can be partially solved by replicating the Buss SR with each unit. More work will be performed to evaluate this problem.

5.4.3 Error Tolerant BOM's

5.4.3.1 Introduction

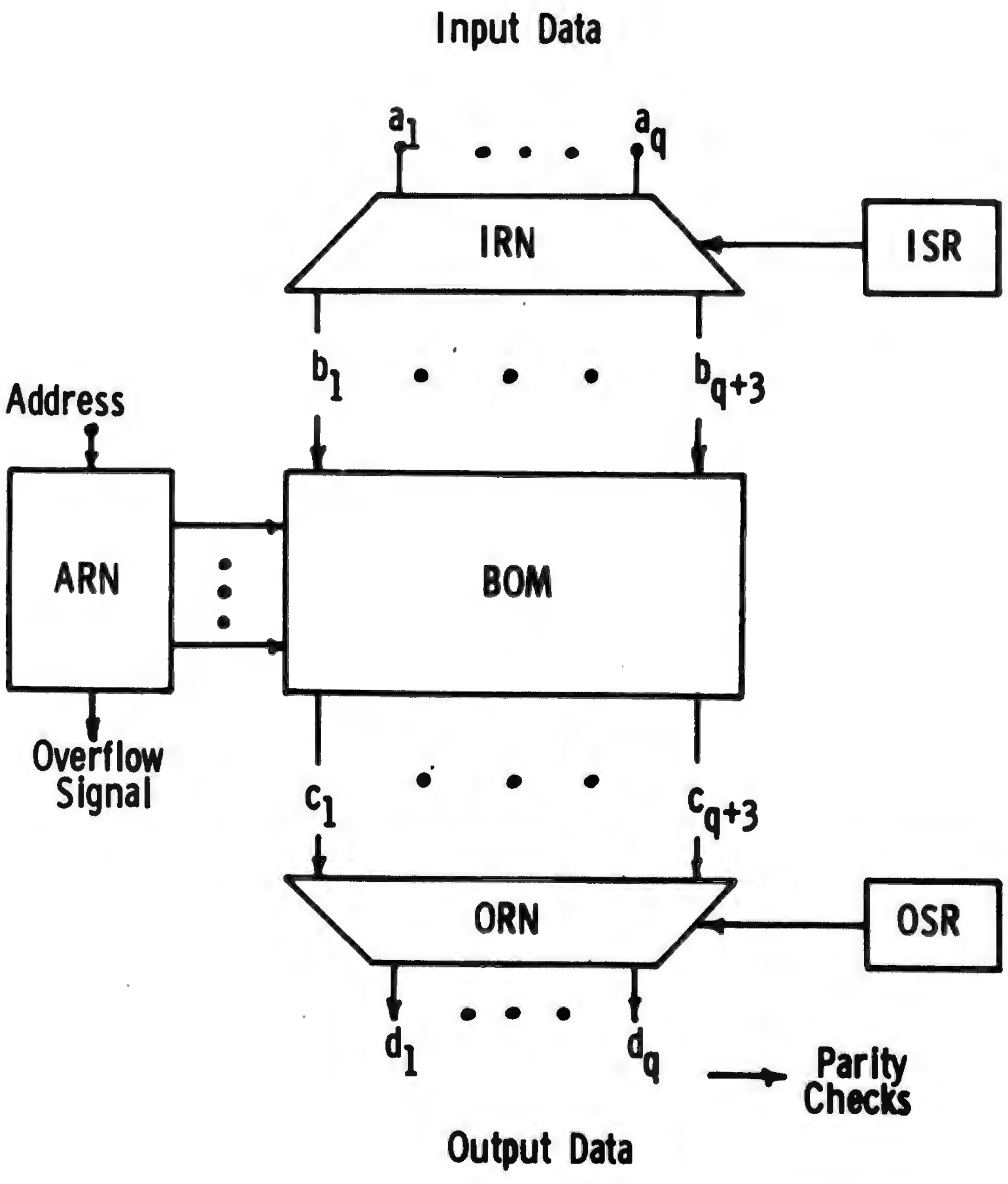
Because of the BOM module's low intrinsic reliability, the modules were changed so that malfunctions in either core storage or addressing circuitry could be bypassed.

A majority of the failures in a digital computer storage module manifest themselves as an error in a single bit position of all words read out of the module. Error correcting codes (Hamming, block codes, etc. [Peterson, 61]) have proven useful in combating this problem, especially in the area of transient failures. However, such an encoding scheme becomes very inefficient once a solid error has occurred: even though the site of failure is well known, the data must always be run through the corrector before use, thereby wasting time, and subsequent new failures will cause the coding to miss errors. The probability of incorrect correction of malfunctions must also be dealt with.

5.4.3.2 Basic Configuration

5.4.3.2.1 Core Storage Reconfiguration The first part of this specification describes a scheme employing spare storage bit planes, reconfiguration networks, and status registers which can be used with error correcting codes to extend their effectiveness or be used with diagnostic tests to extend the useful life of a storage module.

Figure 5.12 contains a block diagram of the type of storage system to be discussed. The Basic Operating Module (BOM) may contain any writable storage medium, e.g. cores, plated wire, thin film, flip-flop, etc. At the input to the BOM there is an Input



Basis Configuration for an Ultra-Reliable Storage Module

Figure 5.12

Reconfiguration Network (IRN) and its associated Input Status Register (ISR). Similarly, the output has an Output Reconfiguration Network (ORN) and Output Status Register (OSR). Attached to the output buss there may or may not be an error corrector, depending on whether or not the data is coded. The structure of the addressing circuitry will be ignored until section 5. 4. 3. 3 .

The status registers and reconfiguration network are good examples of switchable interfaces between modules and a buss of width ℓ+m. These switches and registers were discussed previously in section 5.4.2.3.2 and 5.4.2.3.3..

5. 4. 3. 2. 2 Strategy for Reconfiguration As a first case assume error correcting codes are being used. Then as soon as a solid failure is located, the ISR is changed to its new state, the OSR remaining unchanged. Next, each word in memory is read out, corrected by passage through the corrector, and then returned back into the same word location. Finally the contents of the ISR is transferred into the OSR. The result of this operation is to correct and shift each word in the reconfigured BOM, thereby removing the faulty bit plane and data from subsequent use by the computer.

If error correcting codes are not being used, the solid failures are located by diagnostic tests. A software recovery procedure should be initiated to reestablish the data in the failed BOM. Reconfiguration now becomes a one step process; the ISR is changed to its new state. In this mode of operation the ISR and OSR are always identical so only one such register is required.

In most practical computer applications the frequency of storage failures is low enough so that reconfiguration via changing of the status registers only occurs after relatively long time periods (on the order of several days). For this reason high speed access into and out of the status registers and special circuits to compute their new states is not required; i. e. the computer itself can perform the computation and feed the new state into the register serially.

5. 4. 3. 2. 3 High Reliability The reconfiguration scheme described in this disclosure permits correct operation of a storage device even after multiple bit failures have occurred in the BOM. Since simultaneous multiple failures have an extremely low probability of occurrence, this means a single error correcting code can be made to look like a multiple error correcting code with no increase in complexity of the encoder-decoder and a minor increase in the number of bit planes in the BOM. Further, after a solid bit failure occurs the immediate reconfiguration permits continued operation with no time degradation since the data no longer has to pass through the corrector before use. When error correcting codes are not used, one still has the advantage of being able to use the BOM after it has several bit failures.

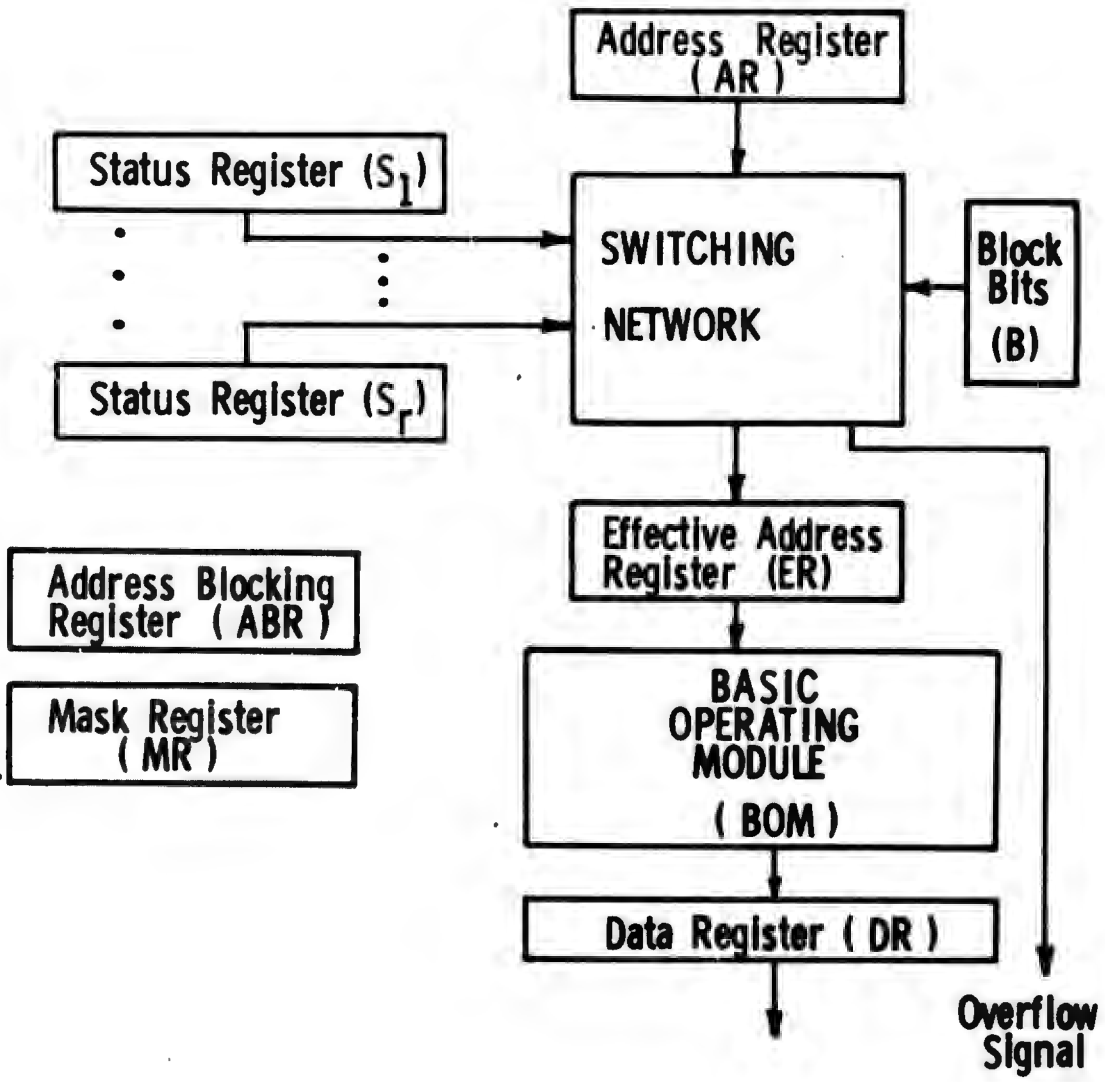
The ability to tolerate failures and still function provides extremely large gains in reliability for the BOM. If P is the probability of a single bit failure and m spare planes exist, the probability of the BOM becoming defunct because of bit failures is P^{m+1} . If, for example, a BOM with three spare bit planes has $P \cong .01$, this means the probability of the reconfigurable system failing because not enough good planes exist is .00000001. Essentially, bit failures in the BOM, which were the most probable, can now be ignored in the reliability calculations.

The reliability of the storage device becomes the reliability of the addressing and reconfiguration circuitry.

5.4.3.3 Dynamic Storage Address Blocking to Achieve Error Toleration in the Addressing Circuitry.

5.4.3.3.1 Basic Configuration. This section will illustrate a method for automatically continuing use of many of the words in a main store after part of its addressing circuitry has failed so as to preclude correct accessing of a subset of the words. To do this, extra circuitry will be added at the address input section of the store. Well known methods can be applied to improve the reliability of such circuitry, whereas none exists for the actual storage devices.

The addressing structure of the high availability store is organized around an existing Basic Operating Module (BOM) as shown in Figure 5.13. The Address Register (AR) receives the input address from the central processing unit. The switching network generates an effective address which does not access any of the BOM addresses lying within a previously determined failed block. This effective address is placed into the Effective address Register (ER) for actual BOM addressing. The switching network is indirectly controlled by information derived from the Address Blocking Register (ABR) and the Mask Register (MR). The contents of these two registers will be described subsequently. The switching logic is directly controlled by register B, which contains the Block definition bits (i.e., the bits in ABR which correspond to 1's in MR) and by the contents of the Status registers (S). There is one S for each entry in B. The AR, ER, ABR, MR, and S are all q bits wide. The function of the switching network and control registers will



BOM Address Reconfiguration

Figure 5.13

become more clear after an example.

The ABR and the MR are used to define the location of each block of addresses which has previously been determined (by diagnostic tests or checking with diagnostic circuitry) to produce erroneous outputs due to the existance of failures in the accessing. These blocks are not to be addressed, and are defined by the contents of the MR and ABR. The MR contains 1's in the bit positions which define the block, eg those bit positions whose values have been constant through all previous malfunctions. In those positions where the bits in the MR are 1, the ABR contains the corresponding bit values; where the MR bits are zero, ABR can contain only value.

<u>5.4.3.3.2 Example, $q = 5$.</u> For example, in a 5 bit address BOM assume tests have found malfunctions in the addresses 8 (01000) and 28 (11100).

The MR will contain 01011 and the ABR will contain 01100. This register pair defines the blocked locations to be those addresses which are specified by:

$$\text{x1x00}$$

$$\text{i.e.}\quad \left\{\begin{array}{l} 01000 \\ 01100 \\ 11000 \\ 11100 \end{array}\right.$$

or address locations 8, 12, 24, 28 respectively. All other addresses can be used.

The test which can be used to examine an address for falling outside a block is, in Iverson notation,

$$\vee / \text{MR} \wedge (\text{AR} \neq \text{ADDRESS}).$$

It is important to realize that when blocks are defined using this scheme, the address locations are not necessarily physically contiguous nor do they correspond to sequential addresses. They do, however, correspond to locations which would be logically grouped together by a failure in the addressing circuitry of a BOM The number of such register pairs depends on the span of the address space and the desired reliability; as each becomes larger more pairs would become necessary.

The algorithm for determining the contents of the ABR and MR, after a failure, is defined recursively. When the first failed address is diagnosed it is placed in ABR and MR is set to all 1's. After the kth failure, MR is respecified to be

$$MR \wedge (ABR \equiv \text{FAILED ADDRESS})$$

while ABR becomes

$$ABR \wedge \text{FAILED ADDRESS}.$$

For a BOM, the addresses used by the program (in the AR) must be sequential, and the addresses used by the ER must be outside the blocked locations. The contents of the S's and B control the switching network to perform this transformation.

The contents of the ith status register, S_i, is determined by the position of the ith 1 in MR (counting from MR_1 to MR_q). If this 1 occurs at MR_k then, denoting the jth entry in S_i by S_{ij},

$$\begin{array}{ll} S_{ij} = 1 & j = 1, \ldots, k \\ S_{ij} = 0 & j = k+1, \ldots, q. \end{array}$$

If MR does not contain at least i ones, S_i is set to all zeros. Similarly bit B_i in B, which corresponds to S_i, is set equal to ABR_k where k is as above. The next two sections show that using this particular method of defining S_i and B_i simp'ifies the address mapping algorithm and its associated circuitry.

Continuing the previous example, the three status registers will have the following contents.

S_1	11000
S_2	11110
S_3	11111

and B will contain 100. . The S_i and B will be used as follows to define the mapping from AR to ER.

Case 1. Let the program addresses range from 0 to 15. To get the ER addresses put the negation of the contents of the first bit position in B (in this case 0) into the bit position corresponding to the last 1 in S_1 (in this case position 2) and map as shown.

$$\begin{tikzcd}
AR & 0 & a_1 \arrow[dl] & a_2 \arrow[d] & a_3 \arrow[d] & a_4 \arrow[d] \\
ER & a_1 & \bar{b}_1=0 & a_2 & a_3 & a_4
\end{tikzcd}$$

In numeric values,

AR	→ ER
0-7	0-7
8-15	16-23

Case 2. Let the program addresses range from 16 t0 23. To get the ER address use registers S_1, S_2 and the values in the first two bit positions in B to define the following mapping.

$$\begin{tikzcd}
\text{AR} & 1 & 0 & a_2 \arrow[dll, dashed] & a_3 \arrow[dl, dashed] & a_4 \arrow[d] \\
\text{ER} & a_2 & b_1=1 & a_3 & \bar{b}_2=1 & a_4
\end{tikzcd}$$

In numeric values

AR	→ ER
16, 17	10, 11
18, 19	14, 15
20, 21	26, 27
22, 23	30, 31

Case 3. Let the program addresses range from 24 to 27. To get the ER address use registers S_1, S_2, S_3 and the values in the first three bit positions in B to define the following mapping.

$$\begin{tikzcd}
\text{AR} & 1 & 1 & 0 & a_3 \arrow[dlll, dashed] & a_4 \arrow[dll] \\
\text{ER} & a_3 & b_1=1 & a_4 & b_2=0 & \bar{b}_3=1
\end{tikzcd}$$

In numeric values

AR	→ ER
24	9
25	13
26	25
27	29

Case 4. If the program addresses equals or exceeds 28, overcapacity is signalled.

AR 1 1 1 a_3 a_4

OVERCAPACITY

The example in Figure 5. 14 illustrates the register definition and address mapping described by this algorithm for the case where only two status registers are present. The remainder of Figure 5. 14 illustrates the contents of the registers, the blocked locations defined by ABR & MR, the mapping of the 24 permitted input addresses, and the four good addresses unused by the algorithm. (The reason the latter exist is because only two instead of three status registers were used in this.) Now assume another failure occurs at address location 14 in the BOM. Then ACR and MR are modified so as to generate the new set of registers and mappings shown in Figure 5. 14b. Now there are no unused good addresses since the number of 1's in MR equals the number of status registers. Again the address mappings performed by the switching network are given.

<u>5. 4. 3. 3. 3 Algorithm for Address Mapping</u> Figure 5. 15 shows a sample ABR and MR for a 12 bit address together with the S_1, S_2, S_3, and B generated from them. If more than 3 status regist rs existed, all S_i for $i > 3$ would be set to zero.

Figure 5. 16 shows how the S_i of the previous example are used to perform the mapping from AR to ER.

1	2	3	4	5	
0	1	1	0	0	ABR
0	1	0	1	1	MR
1	1	1	1	0	S_2
1	1	0	0	0	S_1
1	0				B

Block Addresses: X 1 X 0 0 or 8, 12, 24, 28

Address Mapping:

A R	→ ER
0- 7	0- 7
8-15	16-23
16, 17	10, 11
18, 19	14, 15
20, 21	26, 27
22, 23	30, 31

Unused Addresses: 9, 13, 25, 29

(a)

1	2	3	4	5	
0	1	1	0	0	NBR
0	1	0	0	1	MR
1	1	1	1	1	S_2
1	1	0	0	0	S_1
1	0				B

Blocked: X 1 X X 0 or 8, 10, 12, 14, 24, 26, 28, 30

Mapping:

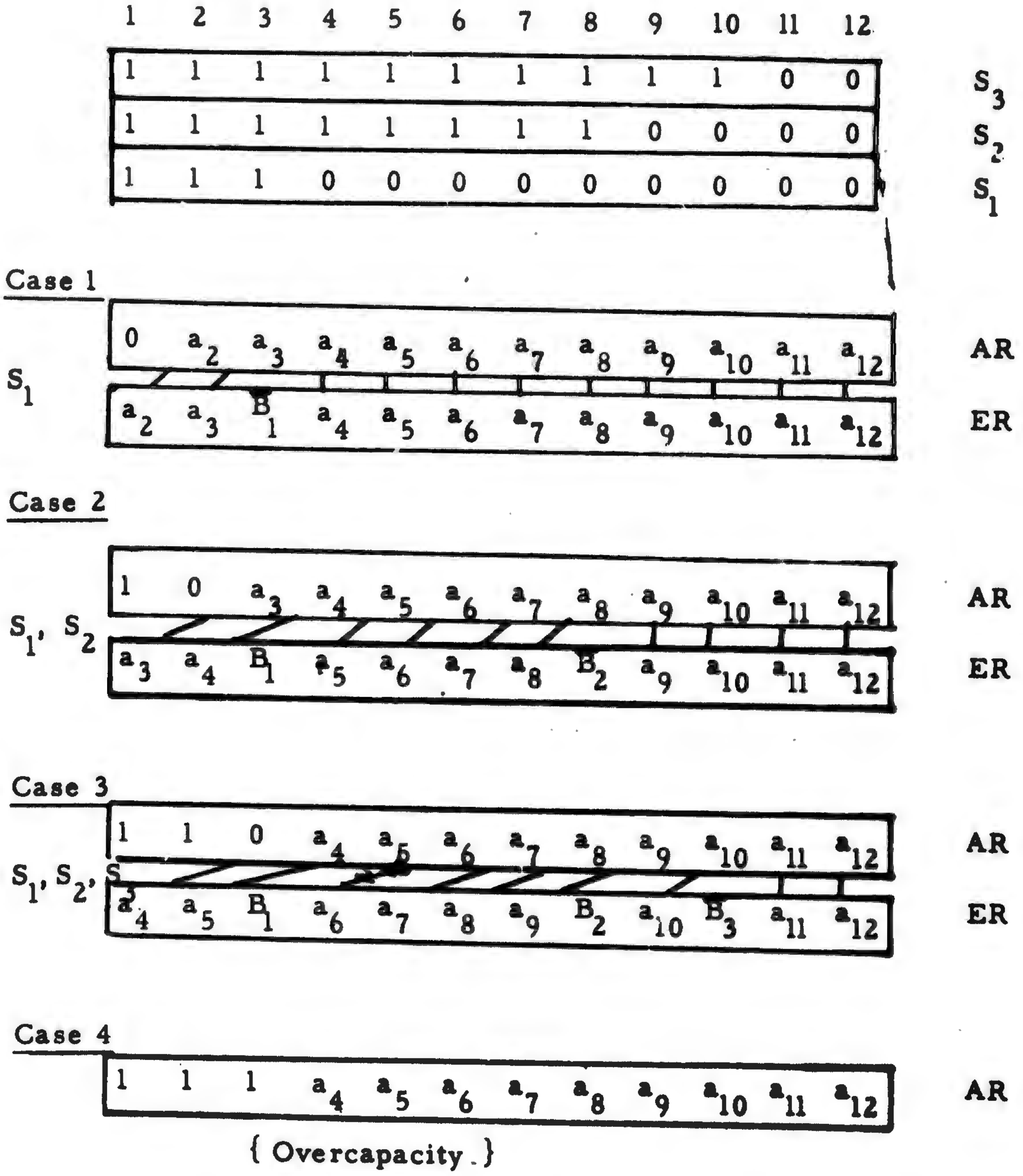
AR	→ ER
0 - 7	0-7
8-15	16-23
16, 17, 18, 19	9, 11, 13, 15
20, 21, 22, 23	25, 27, 29, 31

(b)

Figure 5.14 Example Illustrating Address Mapping Using S_i for

1	2	3	4	5	6	7	8	9	10	11	12	
-	-	0	-	-	-	-	1	-	1	-	-	ABR
0	0	1	0	0	0	0	1	0	1	0	0	MR
1	1	1	1	1	1	1	1	1	1	0	0	S_3
1	1	1	1	1	1	1	1	0	0	0	0	S_2
1	1	1	0	0	0	0	0	0	0	0	0	S_1
0	1	1										B

Figure 5.15 Contents of Control Registers



1 2 3 4 5 6 7 8 9 10 11 12
1 1 1 1 1 1 1 1 1 1 0 0 S3
1 1 1 1 1 1 1 1 0 0 0 0 S2
1 1 1 0 0 0 0 0 0 0 0 0 S1
Case 1
S1
0 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 AR
a2 a3 B1 a4 a5 a6 a7 a8 a9 a10 a11 a12 ER
Case 2
S1, S2
1 0 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 AR
a3 a4 B1 a5 a6 a7 a8 B2 a9 a10 a11 a12 ER
Case 3
S1, S2, S3
1 1 0 a4 a5 a6 a7 a8 a9 a10 a11 a12 AR
a4 a5 B1 a6 a7 a8 a9 B2 a10 B3 a11 a12 ER
Case 4
1 1 1 a4 a5 a6 a7 a8 a9 a10 a11 a12 AR
{ Overcapacity }

Figure 5.16 Example of Address Mappings

Case 1 $a_1 = 0$. The contents of S_1 is used to map input addresses from 0 to $2^{11} - 1$ to the unblocked locations specified by $xx\overline{B}_1xxxxxxxxx$. A 1 bit at position S_{1k} means a a_k is to be shifted to e_{k-1}; a 0 means a_k goes directly to e_k. High order bits are shifted out and dropped. At the point where the transition from 1 to 0 occurs in $\overline{S}_1$, i.e. where $S_{1k}\overline{S}_{1k+1} = 1$ (here at $k=3$) the $\overline{B}_1$ value is inserted into e_k in order to insure that ER is disjoint from the blocked locations.

Case 2 $a_1 a_2 = 10$. Addresses from 2^{11} to $2^{11} + 2^{10} - 1$ are mapped into locations $xxB_1xxxx\overline{B}_2xxx$ using S_1 and S_2. The amount each a_k is shifted is now determined by the change of value points in S_1 and S_2 respectively.

Case 3 $a_1a_2a_3 = 110$. These leading digits characterize the addresses ranging from $2^{11} + 2^{10}$ to $2^{11} + 2^{10} + 2^9 - 1$ which S_1, S_2, and S_3 map to the unblocked addresses specified by $xxB_1xxxxB_2x\overline{B}_3xx$. The shifting and bit insertion follows an extension of the previous algorithm; bit shifting is determined by the number of ones in S_1, S_2, S_3 and bit insertion by the changes in value.

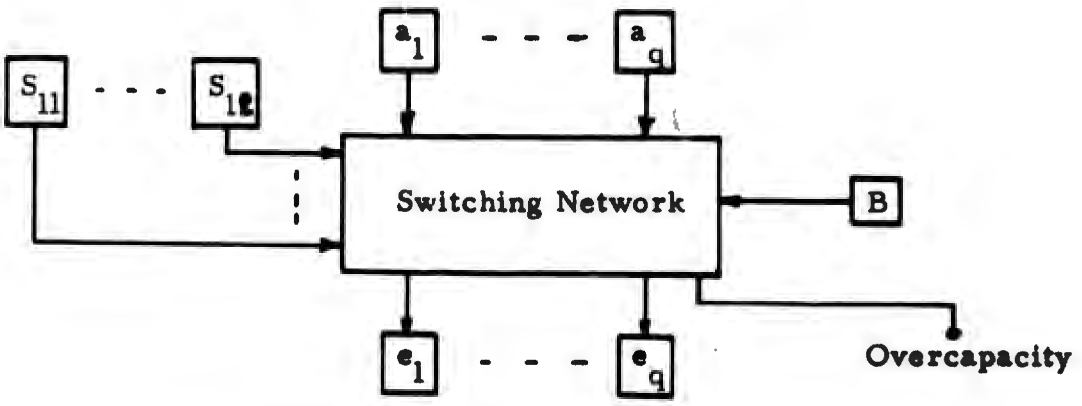
Case 4 $a_1a_2a_3 = 111$. The address in AR is beyond the allowed range for the failed BOM so an overcapacity is indicated.

In the general case where r status registers are available, the following algorithm prescribes their use. When no failures exist, all S_i $i = 1, \ldots, r$ are set to zero. Now assume a failure has occurred, MR has been defined, and that MR contains n ones. Let m be the minimum of n and r. Then all S_i $i > m$ are set to zero while the S_i for $i < m$ are defined as prescribed in the previous section. The values of $B_1, \ldots, B_m$ are also specified as in the previous section; if $m < r$ then B_i $i > m$ are arbitrary. When an address is entered into AR and the first m bits are 1, an overcapacity is signalled. Otherwise, if j is the length of the sequence of leading 1's, i.e. $a_1 \wedge \ldots \wedge a_j \wedge \bar{a}_{j+1} = 1$, status registers $S_1, S_2, \ldots S_{j+1}$ and bits $B_1, B_2, \ldots, B_{j+1}$ are used to perform the shifting and bit insertion. Again the length of the shift of a_k is equal to the number of 1's in $S_{1i}, \ldots, S_{j+1i}$ and the point where bit B_ℓ is inserted is at the location k where $S_{\ell k} \bar{S}_{\ell k+1} = 1$. Only the bit B_m is inserted complemented. This algorithm will allow input addresses from 0 to $(\sum_{i=1}^{m} 2^{m-i})-1$ to be used in the BOM.

For a q bit address 2^q word BOM, use of S_1 alone permits 2^{q-1} words to be available up until such time as enough failures occur so that MR contains no ones. If two registers S_1 and S_2 are used, it is possible to access $2^{q-1} + 2^{q-2}$ locations until MR contains exactly one 1, at which time S_2 is not used and only 2^{q-1} locations can be addressed. In general, if there are r registers it is possible to access $\sum_{i=1}^{r} 2^{q-i}$ locations as long as there are r ones in MR. It is important to realize that as the number of the registers increases the incremental investment in adding an extra register produces an exponentially decreasing incremental gain in addressable storage. For example, in an 8K BOM, S_1 allows 4K to be used after the first failure, adding S_2 allows 6K to be used, adding S_3 allows 7K to be used, etc. Thus, it is felt only a few such registers will be employed in practice.

5.4.3.3.4 Switching Network Implementation. The equations describing the switching network when one status register is used are shown in Figure 5.17. A general circuit implementation for the ith cell of the network is also given.

Figures 5.18 through 5.20 illustrate the structure of the switching network when two status registers are employed: this structure is easily generalized for $r > 2$. The variable B_1V is the value of B_1, either true or complemented, which is to be used in forming ER, i.e. when only S_1 is involved in forming ER, B_1 is used; when both S_1 and S_2 are involved, $\bar{B}_1$ is used. Only the first three don't care conditions shown in Figure 5.18 were used in setting up the network dscription of Figure 5.19. In this graphical description of the network an OR operation is performed at the nodes in the middle and bottom levels. The equations written along side each line are the conditions for gating that line value into the OR gate; i.e., each condition is ANDed with its line value. Subject to the don't care conditions, it is seen that these gating conditions at each node are disjoint. At the middle level $\bar{B}_2$ is inserted at its correct position and all lower order positions are shifted one left, as determined by S_2. The lower level uses S_1 to insert B_1 and left-shift all positions to the left of this point one bit. When S_2 is all 0 the identity transformation occurs at the middle level. Similarly S_1 being all 0 causes the lower level to generate the identity transformation. Figure 5.20 shows a circuit implementation for the ith cell of the network; the two end cells do not use all the indicated AND gates.



OVERCAPACITY = $S_{11}a_1$

$$e_q = a_q \bar{S}_{1q} \vee \bar{B}S_{1q}$$

$$e_i = a_j \bar{S}_{ii} \vee a_{j+1} S_{i+1} \vee B \bar{S}_{i+1} S_i \qquad i = 1, \ldots, q\text{-}1$$

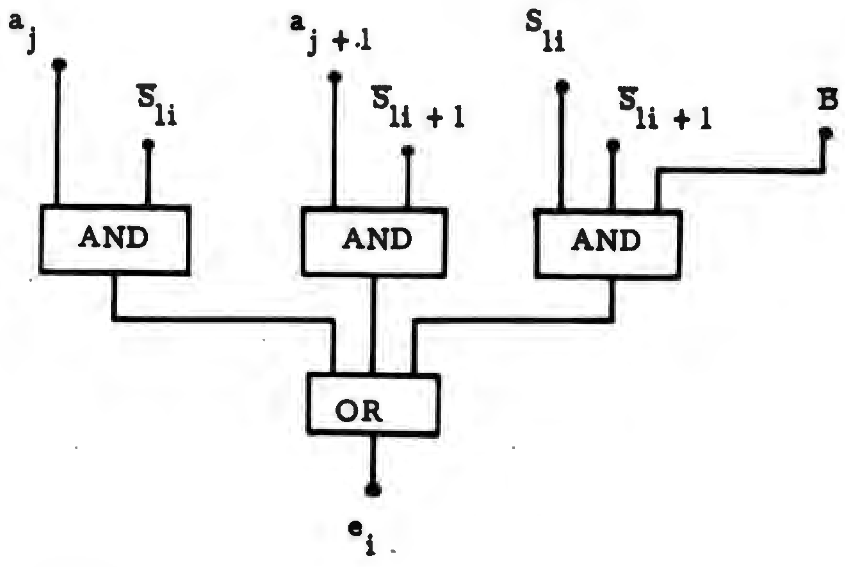
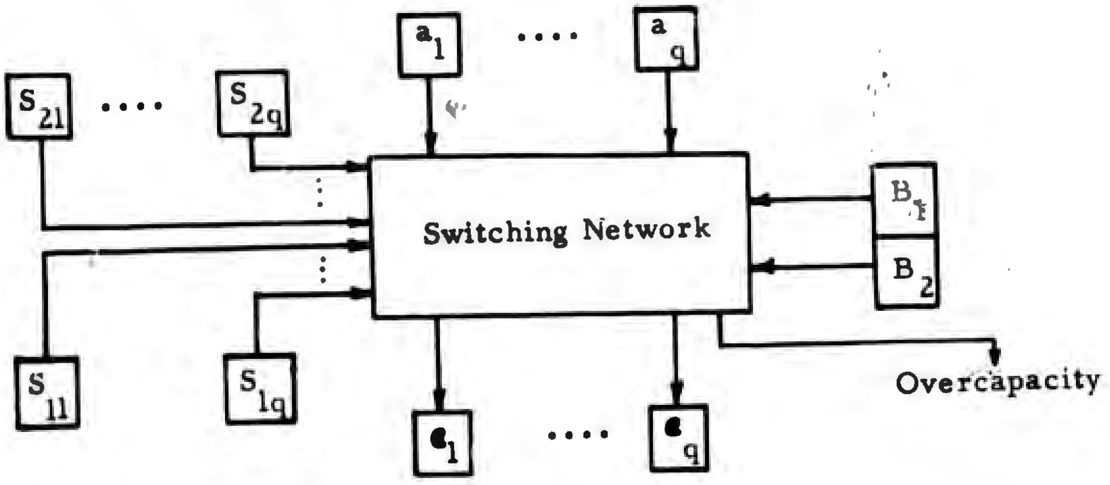


Figure 5.17 General Structure of Switching Network for a q Bit Address with one Register



OVERCAPACITY $= S_{21}\, a_1\, a_2 \vee \bar{S}_{21}\, S_{11}\, a_1 = (a_2 \vee \bar{S}_{21})\, S_{11}\, a_1$

$B_1 \vee = B_1 (\bar{S}_{21} \vee \bar{a}_1) \vee B_1\, S_{21}\, a_1$

The following don't cares exist:	$\bar{S}_{2k}\, S_{2\;k+1} = 0$
	$\bar{S}_{1k}\, S_{i\;k+1} = 0$
	$S_{21}\, \bar{S}_{11} = 0$
	$S_{21}\, \bar{S}_{22} = 0$

Figure 5.18 **Basic Two Status Register System**

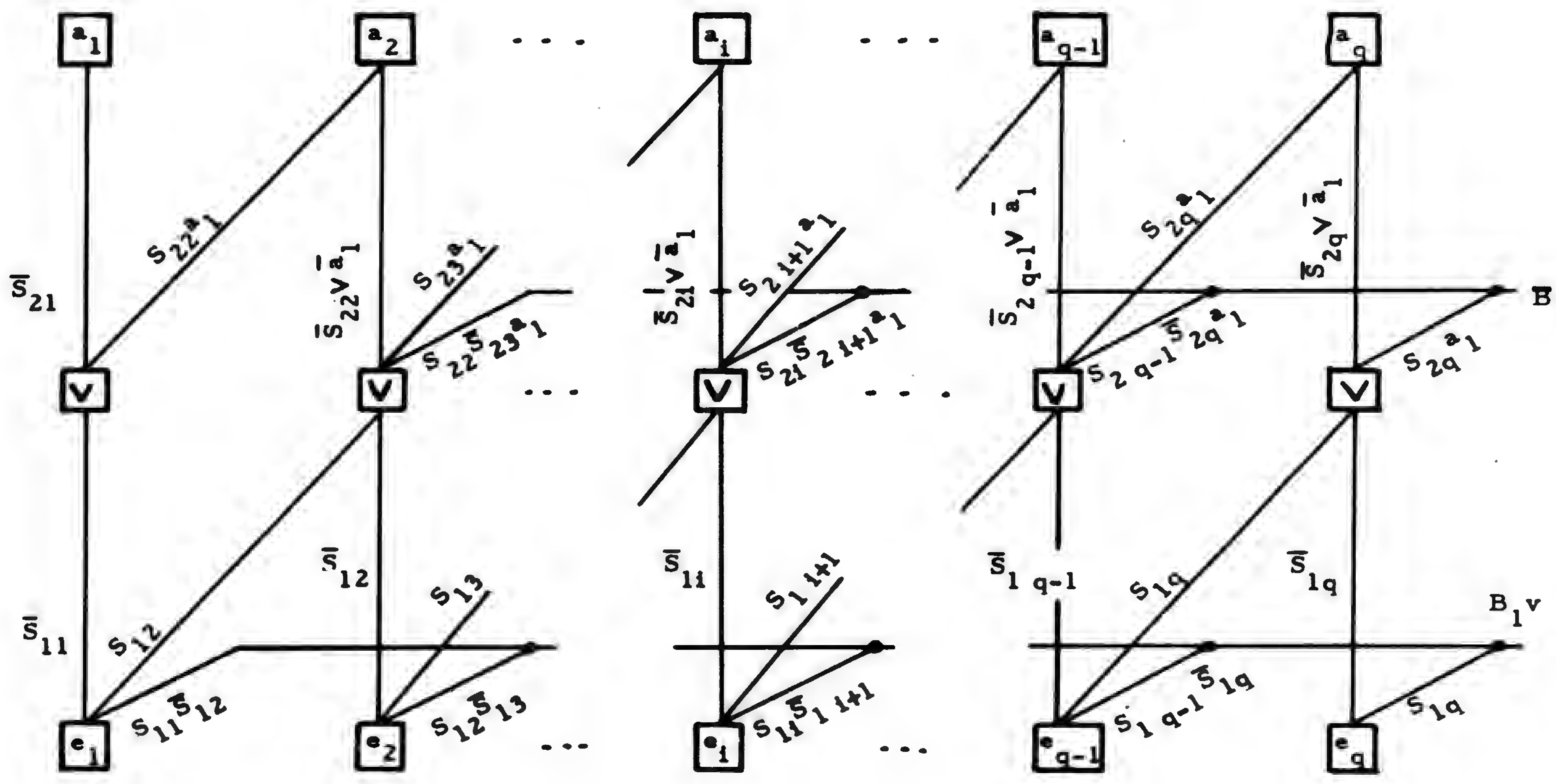


Fig. 5.19 Structure of switching network when two status registers are used.

When $r > 2$ the iterative network described by Figure 5.19 generalizes to one involving r gating levels with S_i controlling the shifting and gating of B_i V into the ith level up from the bottom.

If a large enough fan-in and fan-out exists, it would be possible to produce a two level circuit implementation for the cell of Figure 5.20. However, the delay will still be limited by the fact that a_1, must be used throughout the network to specify the gating conditions. Thus little gain in speed would be achieved.

5.4.3.4 Availability The dynamic address blocking and relocation scheme described here can be combined with the bit plane reconfiguration algorithm in order to produce BOM's which will continue to function in the presence of most failures. Such BOM's adhere to the philosophy of "graceful degradation" as the number of failures mount.

As mentioned earlier, the extra circuitry which was added in order to generate the effective address can be made more reliable using well known techniques, something that cannot be done to the failure prone BOM itself.

In most practical computer applications the frequency of storage failures is low enough so that reconfiguration via changing of the status registers only occurs after relatively long time periods (on the order of several days). For this reason high speed access into and out of the status registers and special circuits to compute their new states is not required; i.e. the computer itself can perform this computation (say by ROS microprogram) and feed the new state into the registers

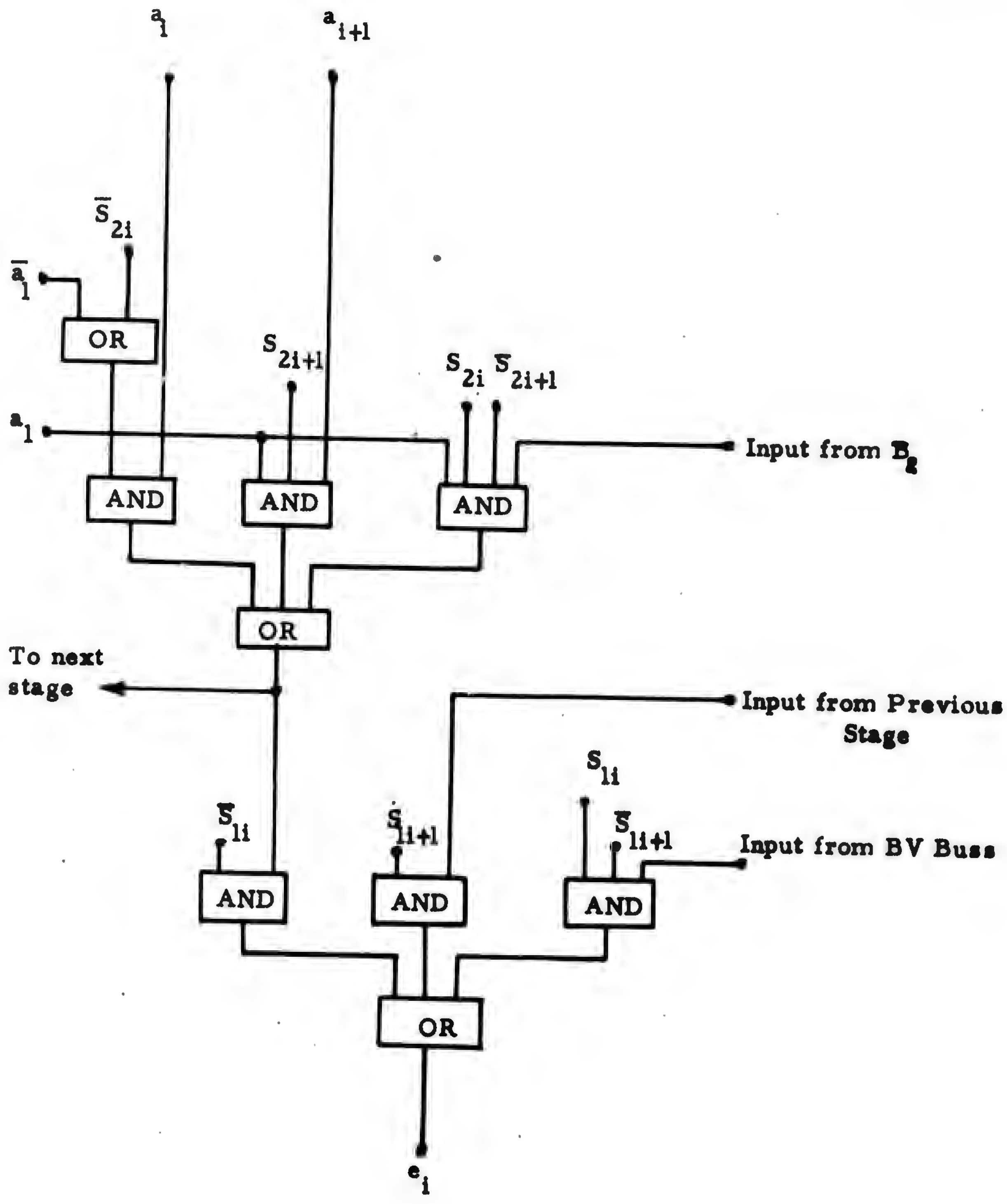


Figure 5.20 Implementation of ith cell for the Switching Network of Figure

serially. If all the registers (S_i and B) are viewed as one big register, a single buss line can shift the data into all register positions. This buss can then be made ultra-reliable using standard multiplexing or TMR techniques. It is also clear that MR and ABR do not have to be in high speed circuitry since they are only used indirectly. As each error is diagnosed in the addressing circuitry, their contents are updated and used as an aid in generating the correct contents of the registers S_i and B.

5.4.4 Ultra-reliable ROS Configuration (URROS)

5.4.4.1 Introduction. In an ultra-available computer employing a Read-Only-Store (ROS) it is necessary to have more than one such ROS present in the machine in order to guarantee correct bit failures, but the required decoding introduces delay in the output. Such codes are also of little help in correcting failures in the addressing circuitry.

Figure 5.21 illustrates the basic structure of the high availability ROS configuration discussed in this disclosure. Each of the n modules is composed of a ROS, its ROS Address Register (ROAR), and its ROS Data Register (RDR). The ROS Address Block (RAB) is used to select those ROS's for which the current address is legitimate, i.e. will not produce an erroneous output by accessing a failed portion of the ROS's addressing circuitry. The Output Reconfiguration Network (ORN) is used to selectively gate the correct bits from the n RDR's onto the output buss.

Use of the RAB and ORN will be shown to provide a correct ROS function even in the presence of multiple failures in the individual ROS modules. The reliability of the whole configuration will depend primarily on the RAB and ORN, devices which can be made reliable using well known techniques.

5.4.4.2 ROS Address Block (RAB). The RAB performs two basic functions. First, it stores, in registers, the location of each block of addresses which has previously been determined (by diagnostic tests) to produce erroneous outputs due to the existence of failures in the accessing. Second, as each address enters the ROS, the RAB tests it for inclusion in an inaccessible block and then signals the ORN of its decision.

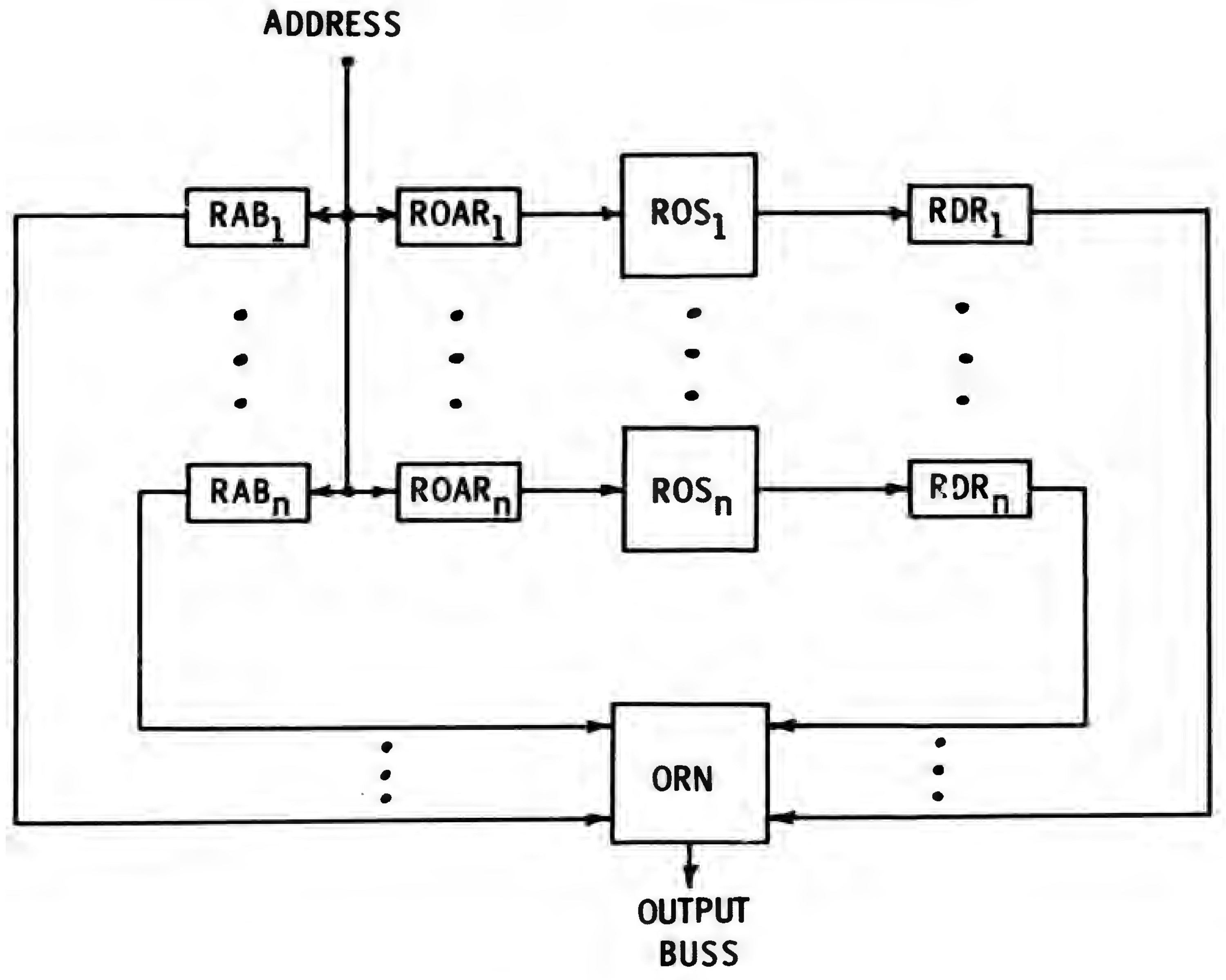


Figure 5.21 Ultra-available READ-ONLY STORE configuration.

If the address does fall into such a block, the ORN is told to ignore the output of that particular ROS.

Two registers, the Address Register (AR) and the Mask Register (MR), which are the same size as the ROAR, are used in the RAB to define each block of locations which is not to be addressed, in just the same way as for the BOM. As in the previous section, the test which can be used to examine an address for falling outside a block is, in Iverson notation,

$$\vee / MR \wedge (AR \neq ADDRESS) .$$

Since the ROS addresses don't change, a circuit implementation, for a general r bit address, is shown in Figure 5.22 . The first level XOR gates produce 1's when the address bits and AR disagree. The AND gates pass the 1's on only if these are positions to be tested. ORing the outpus of these ANDs produces a 1 if there is a disagreement at any tested bit position, i.e. if the address falls outside the block.

The algorithm for determining the contents of the AR and MR, after a failure, is exactly the same as that for the BOM. To repeat when the first failed address is encountered it is placed in AR, and MR is set to all 1's. After the kth failure, MR is respecified to be

$$MR \wedge (AR = ADDRESS)$$

while AR becomes

$$AR \wedge ADDRESS .$$

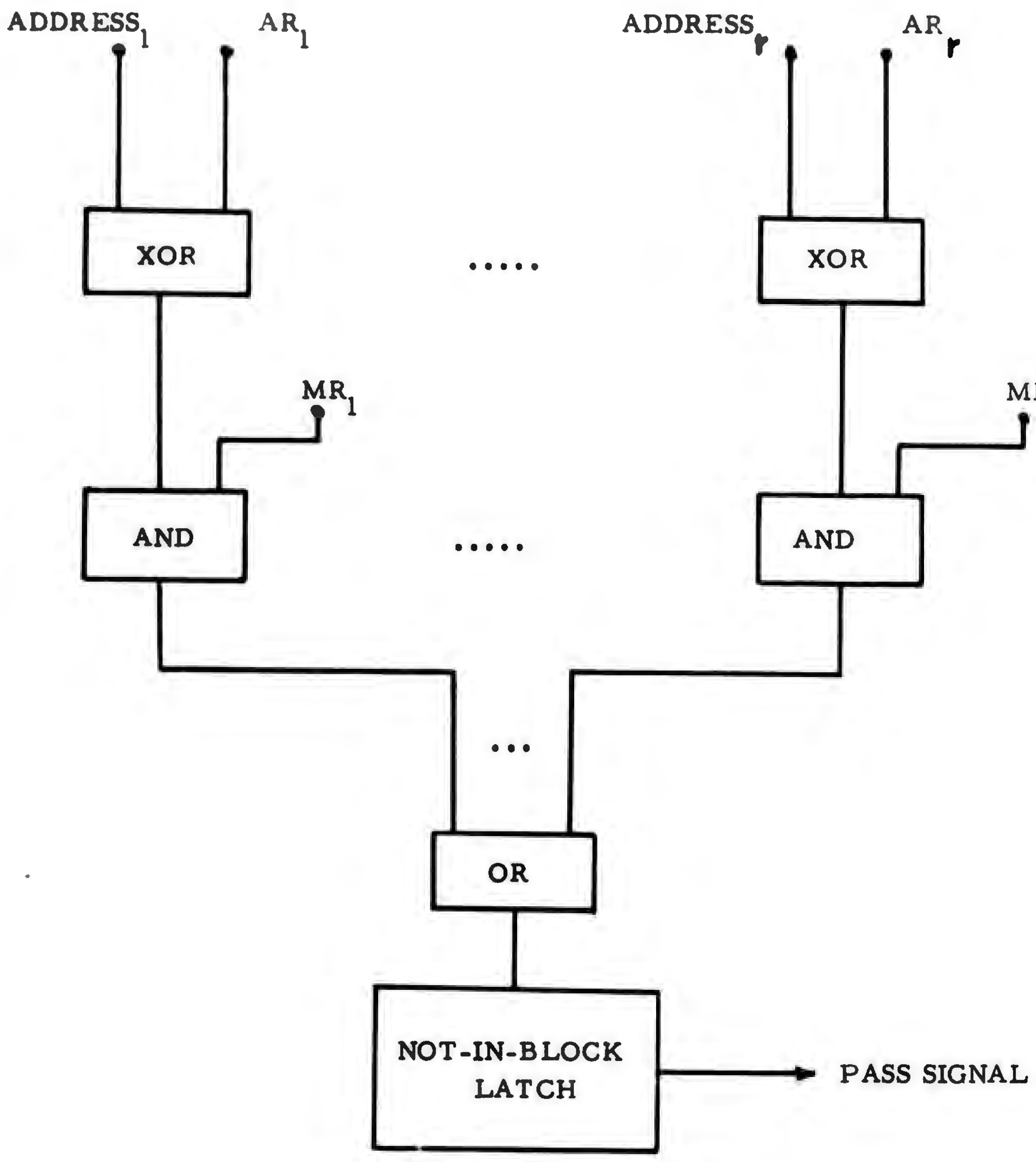


Figure 5.22 Address Test Inside a RAB

The worst case addressing failures occur when the first two failed addresses are complements of each other. Since the expression AR ≡ ADDRESS contains all zeros, MR becomes zero and all addresses are blocked. When it is desired to handle this as a special situation, the complementary address would be loaded into AR and MR. The modification to the test circuitry is as shown in Figure 5.23. Fixing T = 0 provides the normal mode of operation depicted in Figure 5.22. The special case is handled by setting T = 1 and sequentially applying AR and then MR to the inputs labelled AR; the inputs labelled MR can continue to receive the output of MR. Since these tests are being applied concurrently with the ROS access, there is more than enough time to perform sequentially this test for matching addresses.

It is important to realize that when blocks are defined using this scheme the address locations are <u>not</u> necessarily physically contiguous nor do they correspond to sequential addresses. They do, however, correspond to locations which would be logically grouped together by a failure in the addressing circuitry of a ROS.

The number of such register pairs in a typical RAB depends on the span of the address space and the desired reliability; as each becomes larger more pairs would become necessary.

In most practical computer applications the frequency of storage failures is low enough so that reconfiguration via changing of the status registers only occurs after relatively long time periods (on the order of several days). For this reason high speed access into and out of the status registers

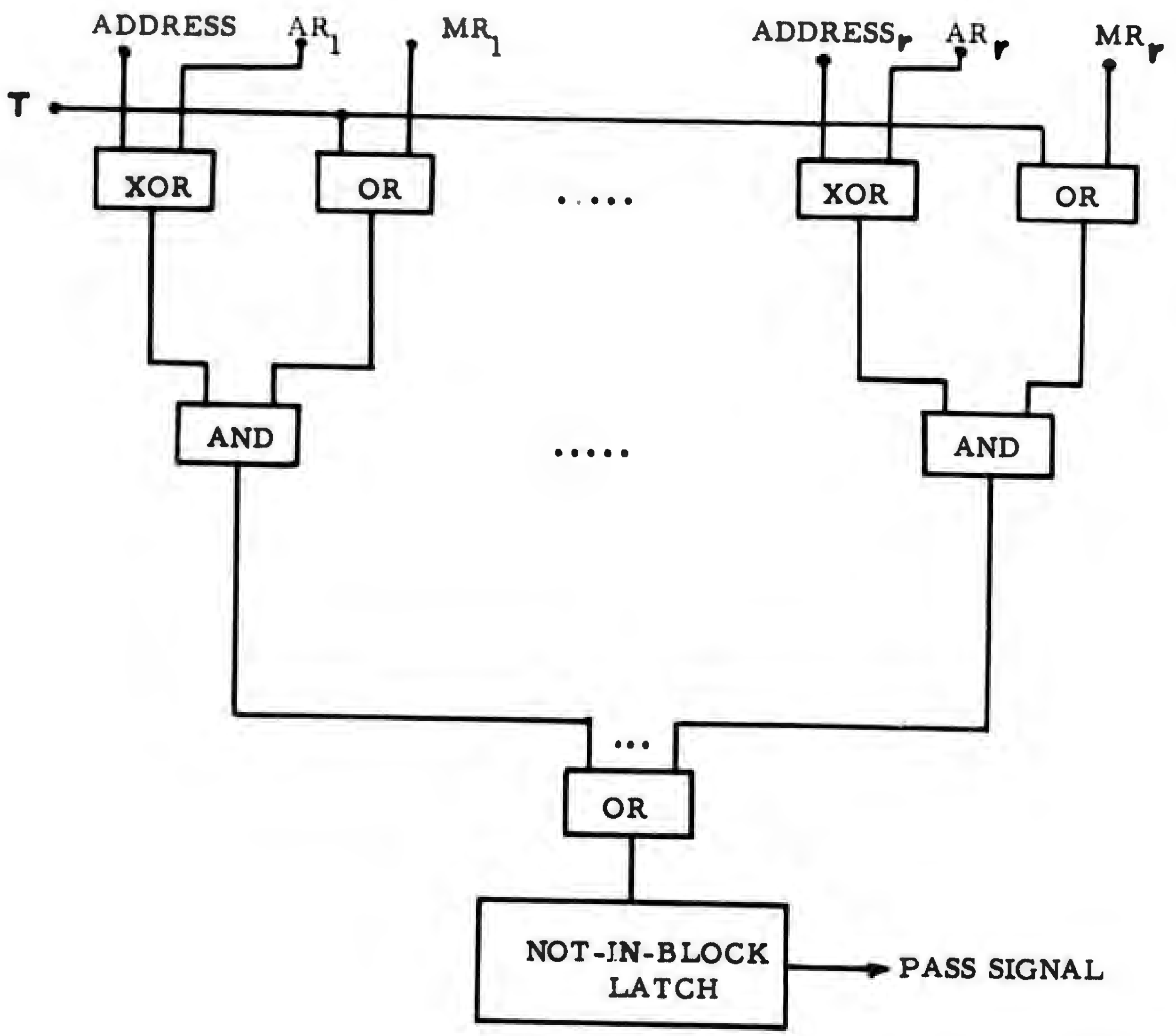


Figure 5.23 Special Case of Complement Addresses

and special circuits to compute their new states is not required; i. e. the computer itself can perform the computation and feed the new state into the register serially.

5. 4. 4. 3 Output Reconfiguration Network (ORN). The ORN is used to selectively gate to the buss the output of the ROS which both the RAB and a check of the RDR determine to be correct. Figure 5. 24 contains a typical circuit implementation. The outputs of the RAB's are labelled $P_1, \ldots, P_n$; the outputs of the checkers (denoting a passed check) are $c_1, \ldots, c_n$; the buss outputs $b_1, \ldots, b_s$; and the output of j^{th} ROS, $RDR_{j1}, \ldots, RDR_{js}$. The function implemented by the ORN is

$$b_k = (P_1 \wedge C_1 \wedge RDR_{1k}) \vee \ldots \vee (P_n \wedge C_n \wedge RDR_{nk}) .$$

Thus only those ROS's which are specified to be operating correctly by both the RAB and the RDR checker will have their outputs gated onto the buss. (Gating the OR of several correctly operating ROS's will not affect the buss output.)

5. 4. 4. 4 Operation and Reliability. When diagnosis discovers and error in a particular ROS it is classified as to whether it is,

1. a failure which will be detected by the RDR checking (e. g. simple parity failures) or,
2. an addressing failure which will cause either the wrong word to be accessed, a combination of words to be accessed, or some other such failure not detected by parity.

In the former case there is no change in the RAB since the checking will block the ROS output when this situation arises. Only in the latter case must the registers in the RAB be updated, using the previously described algorithm.

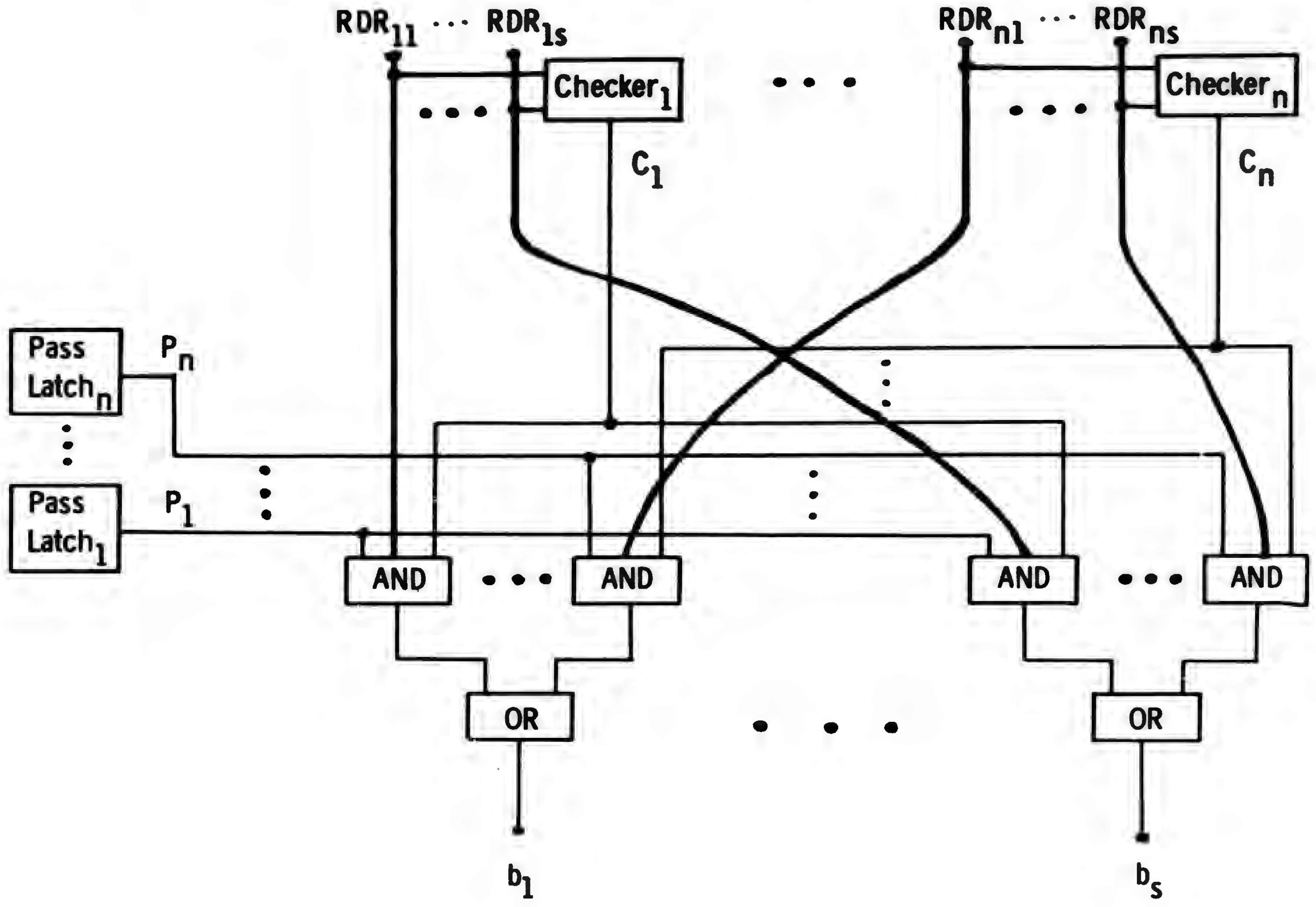


Figure 5.24 ROS Output Reconfiguration Network

Consider the usual n ROS sparing scheme where only one ROS is active and when it fails it is replaced by one of the spares. If P is the probability a ROS has a failure, then the probability of failure of such a structure is P^n. The ROS configuration described in this disclosure has a much lower probability of failure; each of the n ROS's can fail and the system will continue to function correctly. What is required for failure is that all n ROS's fail in a manner such that the same word be erroneous or be blocked for all n ROS's; a far less likely specific event than just having all n ROS's fail.

The probability of failure of the n ROS structure in Figure 5. 21 becomes so low that reliability is bounded by the reliability of the RAB and ORN, which is far higher than that of a core storage array. Further, well known techniques exist for improving the reliability of circuits such as the RAB and ORN.

The system structures diagrammed in Figures 5. 21 5. 24 assumed that the data output was the complete s bit wide ROS output. A higher reliability is achieved by segmenting the ROS by bit planes and using the previous techniques on each segment as shown in figure 5. 25. For example, suppose the ROS put out an 80 bit word. Then a segmentation into 10 bytes would involve having separate RABs for each byte and grouping, at least conceptually, the existing parts of the ORN by bytes. The reliability improves because the previously discussed combinations of failures which bring the ROS structure down must now occur in one segment, i. e. in a much smaller module, hence a less likely event.

ROS _ PAGING / SEGMENTING

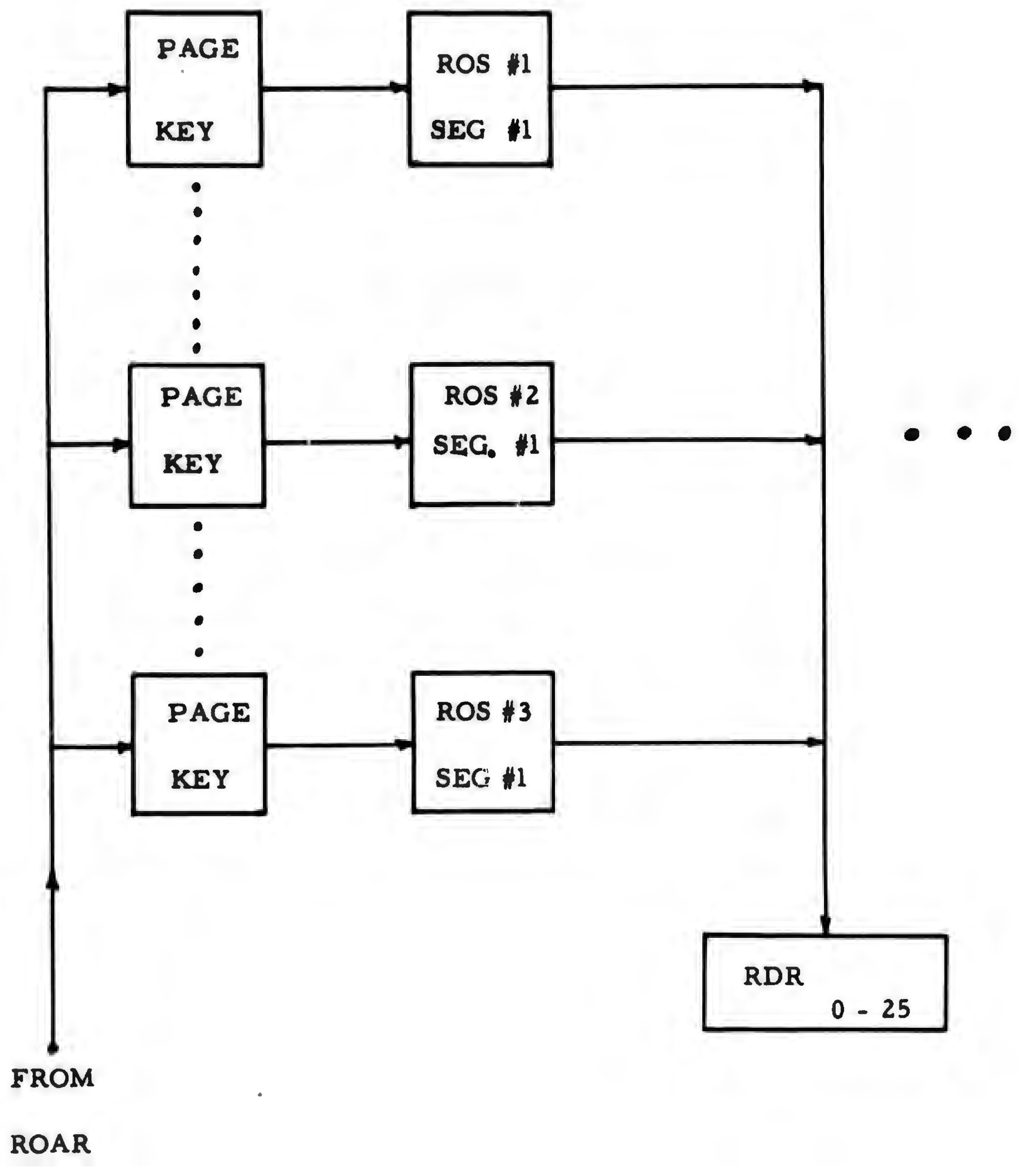


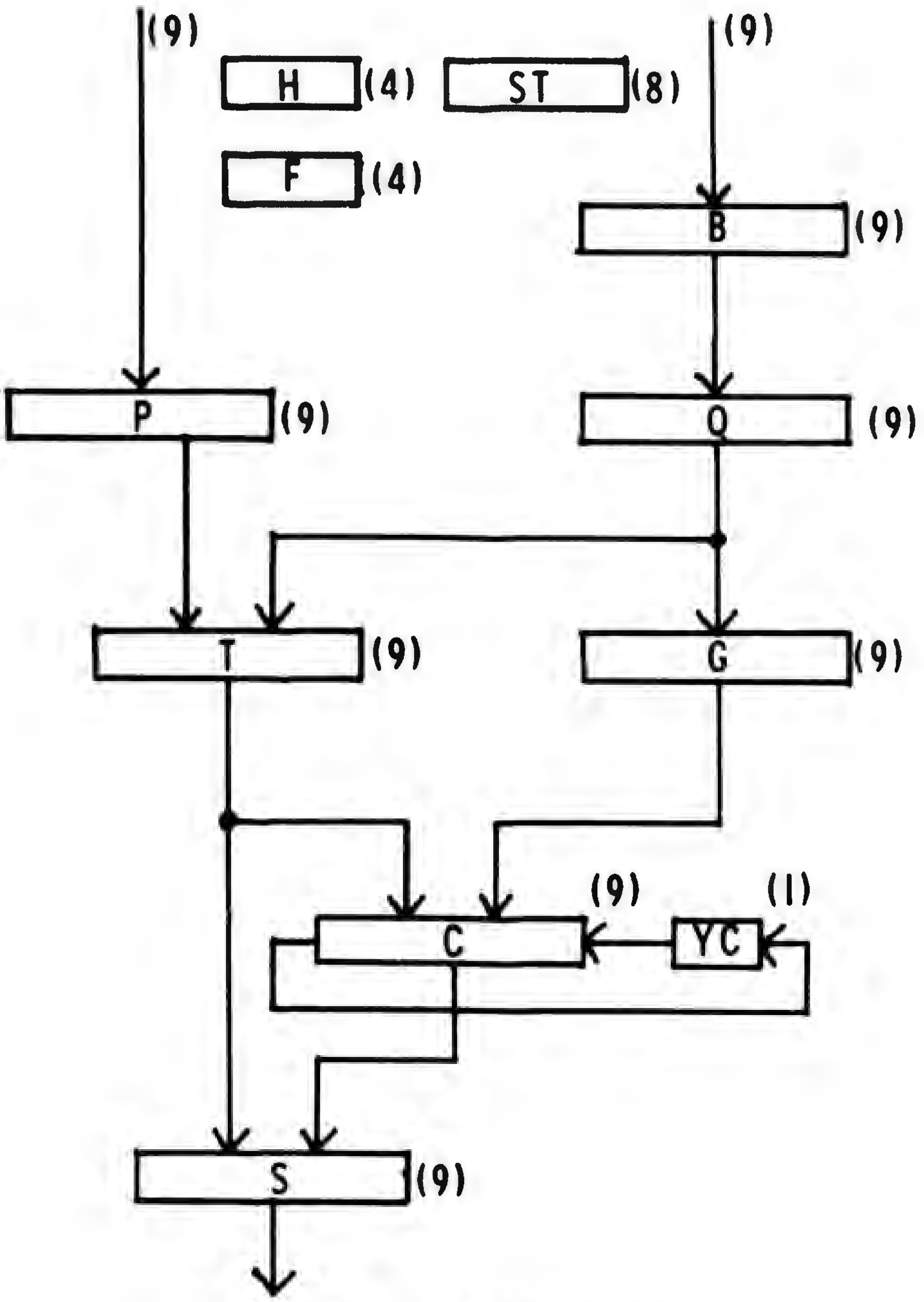
Figure 5.25

5.4.5 An Error-Tolerant BALU

The techniques of multiplexing, and reconfiguration can be extended so as to allow an ALU to tolerate failures and still function correctly. An ALU can be designed so that the bit planes are relatively independent, being coupled mainly for the purposes of shifts and carry propagation. By switching between bit planes the effect of a failure on that plane or in a position of control logic can be overcome. Thus the reconfiguration switch must not only buss the data into and out of the correct planes, it must also change the point where the carries and shifted bits are sent.

For purposes of this ARC a byte ALU (BALU) design will be given. This design follows closely the ALU design for the 360/model 40 including all external control signals, but with the addition of one spare bit plane and a duplicated internal control register. Since the 360/model 40 ALU can carry out all 360 ALU instructions, no time will be spent in proving that a true ALU has been designed.

The basic BALU schematic is shown in Figure 5.26. The numbers in parenthesis indicate the number of bit positions in each register or data path. For error detection, the BALU is assumed built using two wire logic (as is the model 40 ALU). At the input, the data is transformed from parity checked to two wire logic information. Following the model 40, the BALU carries out the arithmetic operations using two wire logic for checking its operations. The checked information is then transformed from two wire form back to parity checked form and transmitted over the multiplexed buss. This design, already covered in the model 40 and section 5.4.2.3 is not repeated here.



BALU SCHEMATIC

Fig. 5.26

In Figure 5. 26, B1 and B2 are buffer registers used for four bit shifting. P and Q are input registers, YC is the carry buffer also used for communication between BALUs. F (4 bits) holds the "function code" received from the PCU (F' is the spare). The bit positions of T generate any function of two variables; G performs pass through, complement, 2 bit right shift (end around) or all zero output; S performs exclusive-OR; and C contains the carry circuitry. These functions, with the proper ROS microcode, carry out all the S/360 instructions.

The logic for each position in T and S is the same as before, since the bit positions are independent (there is one more such bit position however).

$$T_i = k P_i Q_i \vee \ell P_i \bar{Q}_i \vee m \bar{P}_i Q_i \vee n \bar{P}_i \bar{Q}_i \quad i = 0, \ldots, 8$$

$$S_i = \bar{T}_i C_i \vee T_i \bar{C}_i \quad i = 0, \ldots, 8$$

The control signals k, l, m, n, y, (d), x, w, and the F combinations are explained in Figure 5. 27 taken from the model 40 ALU description. F is duplicated (F and F') and the outputs are inhibited if an error is detected in either box. The end around carry (YC) is TMR'd.

The BALU status register has 9 positions. Eight bits refer to the data paths and 1 bit to the use of F or F'. The usual table shows the plane made idle.

F	Mnemonic	Function	Controls
0	OR	$P \vee Q$	k ℓ m
1	EQ	$P = Q$	k n
2	DSQ	P - Q (dec)	k n y (d)
3	SUQ	P - Q (hex)	k n y
4	P	P	k ℓ
5	AND	$P \wedge Q$	k
6	DSP	Q - P (dec)	k n x (d)
7	SUP	Q - P (hex)	k n x
8	PNQ	$P \wedge Q$	ℓ m
9	Q	Q	k m
10	XOR	$P \neq Q$	ℓ
11	QNP	$\overline{P} \wedge Q$	m
12	RSH	1/2 Q	w
13	LSH	2Q	x
14	DAD	P + Q (dec)	ℓ m x (d)
15	ADD	P + Q (hex)	ℓ m x

Direct Functions

P + Q (hex)	ADD
P - Q (hex)	SUQ
$P \wedge Q$	AND
P @ Q	INDIRECT (F)

Figure 5. 27 ALU Control

Bit Plane	State Trigger							
Made Idle	ST 1	ST 2	ST 3	ST 4	ST 5	ST 6	ST 7	ST 8
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	1
3	0	0	0	0	0	1	1	1
4	0	0	0	0	1	1	1	1
5	0	0	0	1	1	1	1	1
6	0	0	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1

The logic functions determining which plane is idle are

$IL_0 = \overline{ST}_8$

$IL_{8-i} = \overline{ST}_i \; ST_{i+1} \qquad i=1,\ldots,7$

$IL_8 = ST_1$

The logic functions for G can be seen easily after considering the following table of shift connections. Each row in the table specifies the output bit or data plane to which that input plane is to be connected when the shift is performed. For example, input plane 1 is normally shifted to output plane 3 unles 1, 2, or 3 are idle. If 1 is idle the shift is ignored; when 2 or 3 are idle the shift must skip over one extra plane, so the connection is to 4.

Bit Plane Made Idle

Input Data Plane	0	1	2	3	4	5	6	7	8
0		3	3	2	2	2	2	2	2
1	3		4	4	3	3	3	3	3
2	4	4		5	5	4	4	4	4
3	5	5	5		6	6	5	5	5
4	6	6	6	6		7	7	6	6
5	7	7	7	7	7		8	8	7
6	8	8	8	8	8	8		0	0
7	1	0	0	0	0	0	0		1
8	2	2	1	1	1	1	1	1	

The equations <u>representing</u> this shift are

$$G_i = x\,Q_i \vee y\,A_i \vee w[(IL_{i-1} \vee IL_{i-2})Q_{i-2}$$

$$\vee (IL_{i-1} \vee IL_{i-2})Q_{i-3}] \quad i=0,1,\ldots,8 \bmod 9$$

The carry triggers are similarly modified

$$C_i = T_i G_i \vee T_i[IL_{i-1} C_{i-1} \vee IL_{i-1} C_{i-2}] \quad i=0,\ldots,8 \bmod 9.$$

The four bit shifting or pass through on the input line to Q is shown schematically in Figure 5.28. If the bit positions are numbered as shown and the input lines are called I_i, $i = 0,\ldots,8$, the logic is as follows.

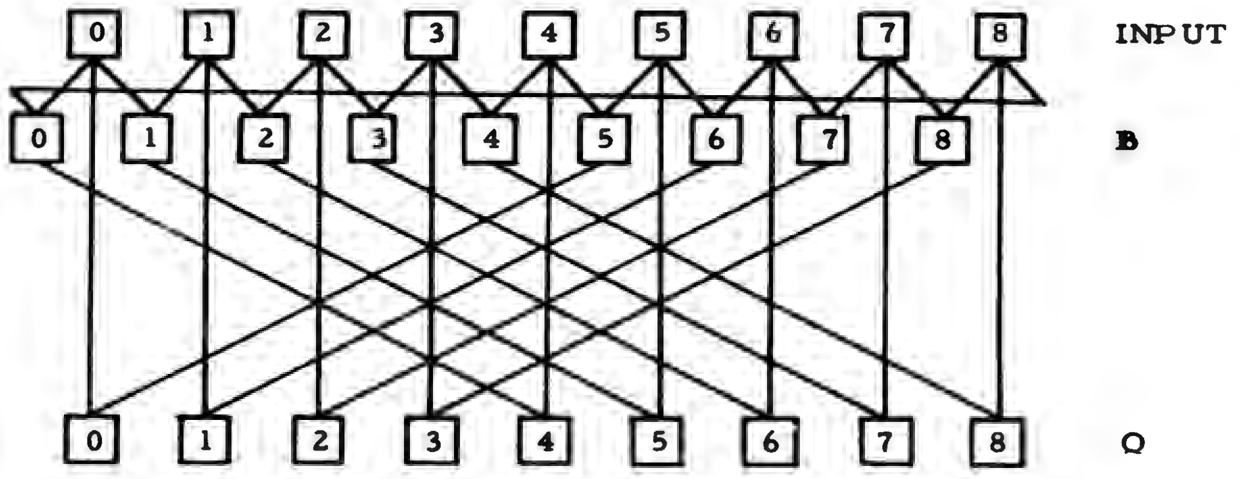


Fig. 5.28 BALU 4 bit shifting network with one spare plane.

$$B_0 = I_0\, ST_4 \quad v \quad I_8\, \overline{ST_4}$$

$$B_k = I_k\, (ST_{4-k} \; v \; \overline{ST_{9-k}}) \quad v \quad I_{k-1}\, \overline{ST_{4-k}}\, ST_{9-k} \quad k=1, 2, 3$$

$$B_4 = I_4\, \overline{ST_5} \quad v \quad I_3\, ST_5$$

$$B_5 = I_5\, \overline{ST_4} \quad v \quad I_4\, ST_4$$

$$B_k = I_k\, \overline{ST_{9-k}}\, ST_{13-k} \; v \; I_{k-1}\, (ST_{9-k} \quad v \quad \overline{ST_{13-k}}) \quad k=6, 7, 8$$

$$Q_k = \overline{SH} \;\wedge\; I_k \quad v \quad SH \;\wedge\; B_{k+5 \bmod 9}$$

(SH ≡ logic signal indicating a 4 bit shift is requested)

The setting of the status register is discussed in section 5.4.2.3.

This design can be easily extended to an ALU with q data bits and m spare bits in a word, by enlarging the size of the status register and permitting the carries and shifts to enter any one of m+1 adjacent bit planes.

5.4.6 TMR/Sparing.

5.4.6.1 Background. Masking redundancy schemes based on voting (Triple Modular Redundancy - TMR, etc.) have been known for a long time [N.1]. They have the advantage of providing quick, easily implemented solutions to the problem of correct computer operation in the presence of both solid and transient failures, but suffer from a low reliability for the amount of hardware invested, i.e. if n logic modules with probability of failure P are being used, n being an odd integer greater than 1, the probability of the voted system failing is

$$\sum_{i=\frac{1}{2}(n+1)}^{n} \binom{n}{i} P^{i}(1-P)^{n-i}$$

which, for small P, approaches

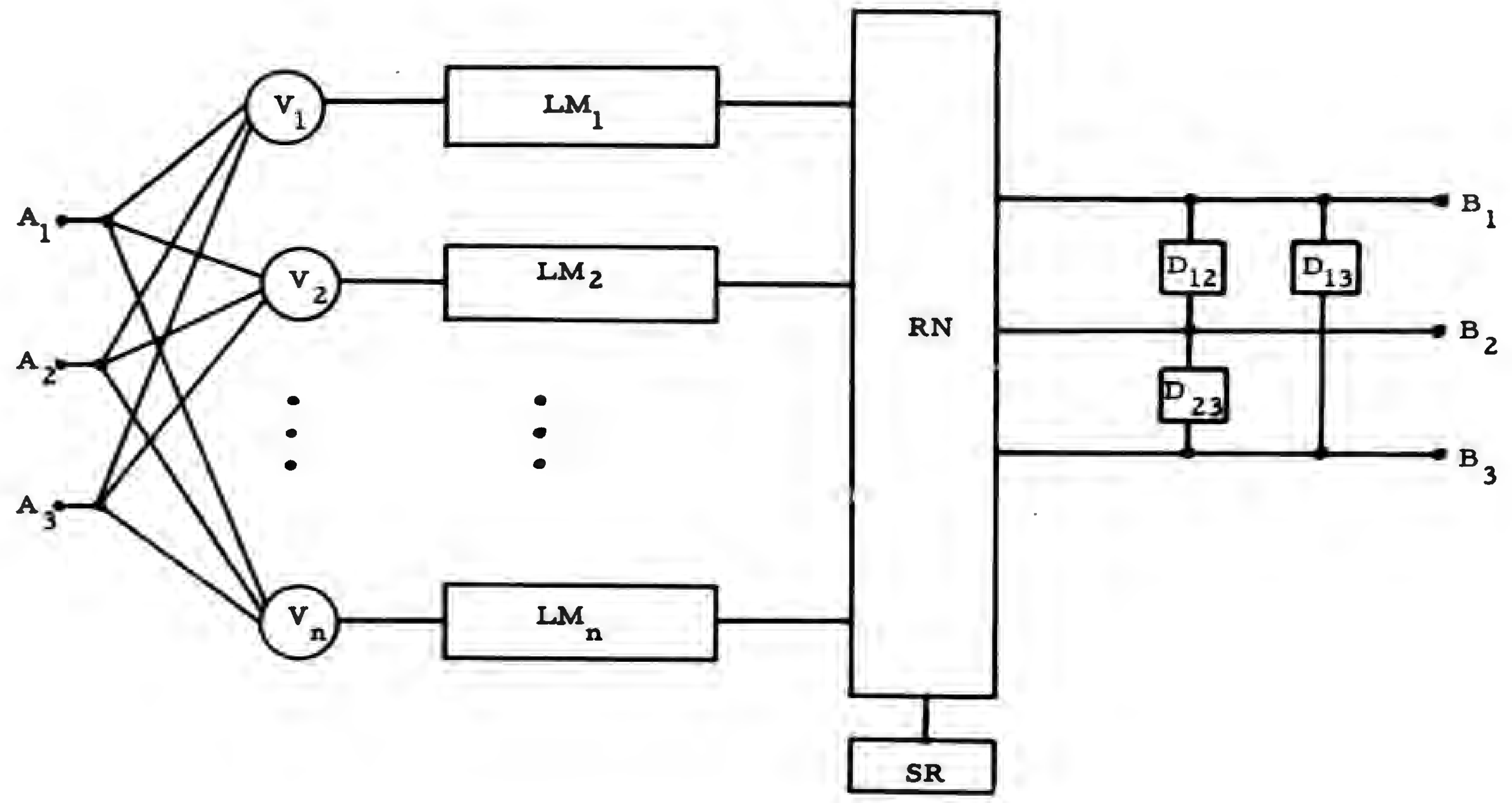
$$\binom{n}{\frac{1}{2}(n+1)} \qquad P^{\frac{n+1}{2}}$$

Sparing, or standby redundancy, activates only one module at a time. When a failure is detected by hardware implemented checking or diagnosis, the active module is switched out and one of its spares switched in. The main disadvantages of sparing are that it requires the computation and storage of diagnostic tests, or the implementation of comprehensive checking circuitry, and that transient errors are very easily overlooked. Its primary advantage is that the probability of failure can be made to approach P^n where, again, P is the probability of the individual module failure.

This section describes a scheme, called TMR/Sparing, which combines the failure-handling ability of masking redundancy with the high reliability of standby redundancy for computer applications requiring ultra-high availability and reliability.

<u>5.4.6.2 Basic TMR/Sparing Configuration.</u> Figure 5.29 depicts the general TMR/Sparing configuration. Each data path is actually composed of many individual data lines, so the paths shown in Figure 1 must be thought of as representing vectors of lines. Thus A_1, A_2, and A_3 are identical input vector-busses, or just busses, and B_1, B_2, and B_3 are identical output busses. Each of the n logic modules (LM) has a vector-voter (V) at its input in order to generate the majority function $A_{1j}A_{2j} \vee A_{1j}A_{3j} \vee A_{2j}A_{3j}$ for each line j in the input vector. The status register (SR) is used to select three of the n logic modules and the reconfiguration network (RN) connects these three module's outputs to the three output busses. A trio of vector-discriminators D_{11}, D_{12}, and D_{23} are connected across the output busses in a delta arrangement in order to detect disagreement among the three outputs.

<u>5.4.6.3 Reconfiguration: Special Case of n=4.</u> The structure of the RN and SR shown in Figure 5.30 is treated as a special case when n=4, (Section 5.4.6.4 handles n > 4). Using the state mapping table, the following configurations are allowed.



5.29 **TMR/Sparing Configurations**

Status Register			Buss Connections			Idle Module
SR_1	SR_2	SR_3	B_1	B_2	B_3	
0	0	0	LM_1	LM_2	LM_3	LM_4
0	0	1	LM_1	LM_2	LM_4	LM_3
0	1	1	LM_1	LM_3	LM_4	LM_2
1	1	1	LM_2	LM_3	LM_4	LM_1

The selection logic shown in Figure 5.30 shows how the RN converts this SR information into the mapping of the three chosen module outputs to the three busses. Switching cells like this one are used for each data line in the vector making up the module's output; they all share the common SR.

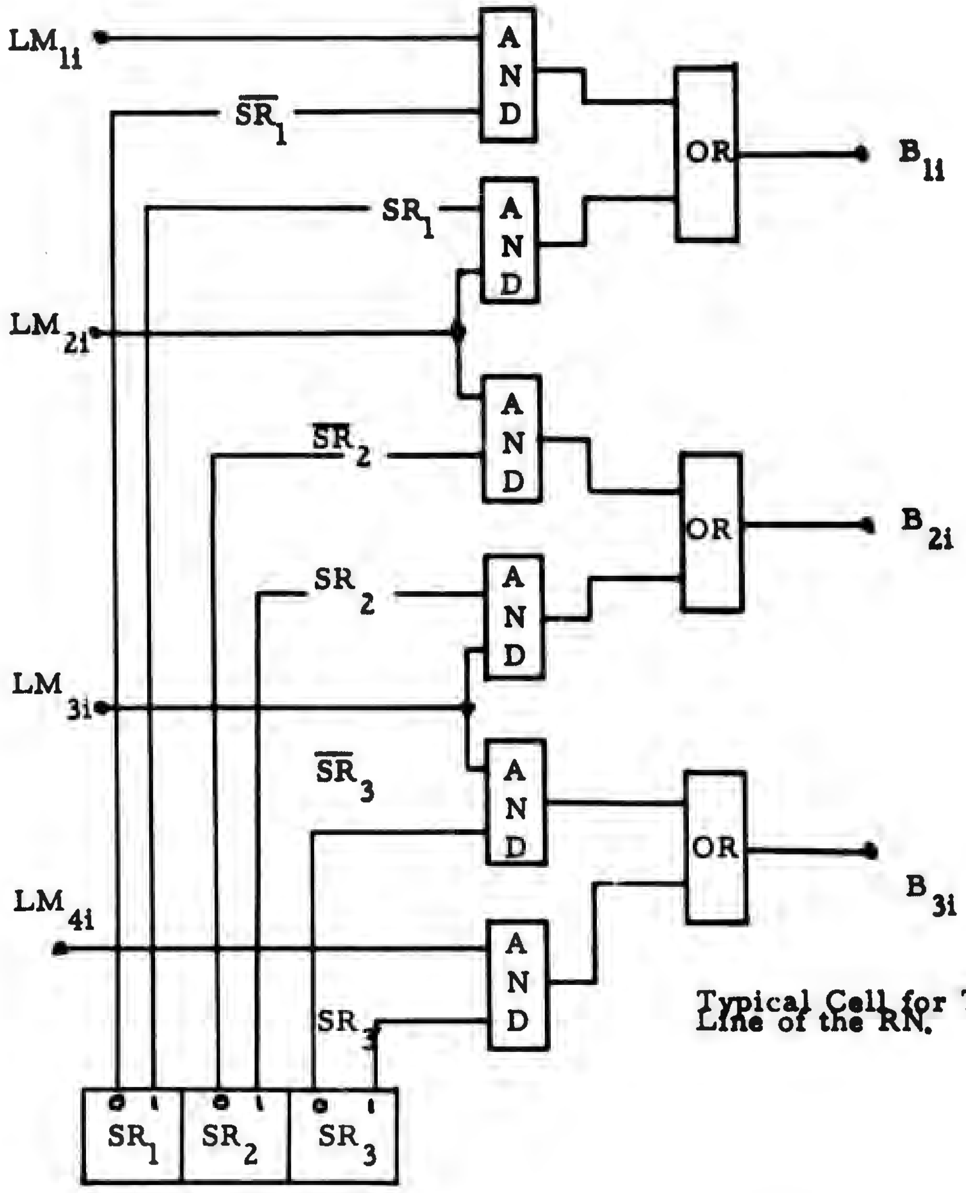
Initially LM_1, LM_2, and LM_3 are connected to the busses; SR is all zero. When a failure is signalled by the discriminators, SR is changed so that the failed module now becomes the idle one. This algorithm can be specified in Iverson notation as

```
→(~∨/F←(D12,D13),(D12∧D23∧~D13),D13∧D23∧~D12)/NOFAILURF
SR←(F[1],(F[1]∨F[2]),∨/F)∧~F∧SR
```

where DIJ is the output of the discriminator across busses I and J (DIJ = 1 when the busses disagree), and F[K] denotes the condition, "buss line K has failed". A typical circuit implementation for this algorithm is shown in Figure 5.31 wherein it is assumed that the setting of the bit positions in SR is controlled by a latch signal (not shown). Such a latching or change of configuration should occur at some convenient point, such as the end of a macro-instruction in the computer.

	LM_1	LM_2	LM_3	LM_4	
B_1	0	1	▨	▨	SR_1
B_2	▨	0	1	▨	SR_2
B_3	▨	▨	0	1	SR_3

Status Register States



Typical Cell for The i-th Data Line of the RN.

Figure 5.30 SR and RN for n = 4

When the second module failure is encountered, returning to the currently idle module means using the module that failed first. If this first failure was just a transient, correct operation will now result. If it was a solid failure, another reconfiguration results as soon as the failure produces an erroneous output. Thus the second failed module is brought back to see if it can be used; i.e. if its failure was transient. It is in this way that the TMR/Sparing scheme handles transient failures; by temporarily removing a module and bringing it back again when needed.

After two modules have solid failures this algorithm for operation results in a continual swapping of modules as their failures generate erroneous outputs. It would be possible to control this cycling by placing an extra input, say C, in all three upper AND gates in Figure 5.31. When $C = 1$ the normal reconfiguration action results: when $C = 0$ all error signals are suppressed and the configuration remains fixed. Setting $C = 0$ could be controlled by a counter which records the number of reconfigurations occurring in some fixed time, say an hour; if too many occur, C could be made 0.

It is important to realize that since all four modules are receiving data input, there is no need to halt normal computer operation in order to perform a system update whenever a reconfiguration occurs. Because of the simplicity of the circuitry, no extra time need be used to set the SR after a failure either. Thus reconfiguration can occur with no time degradation.

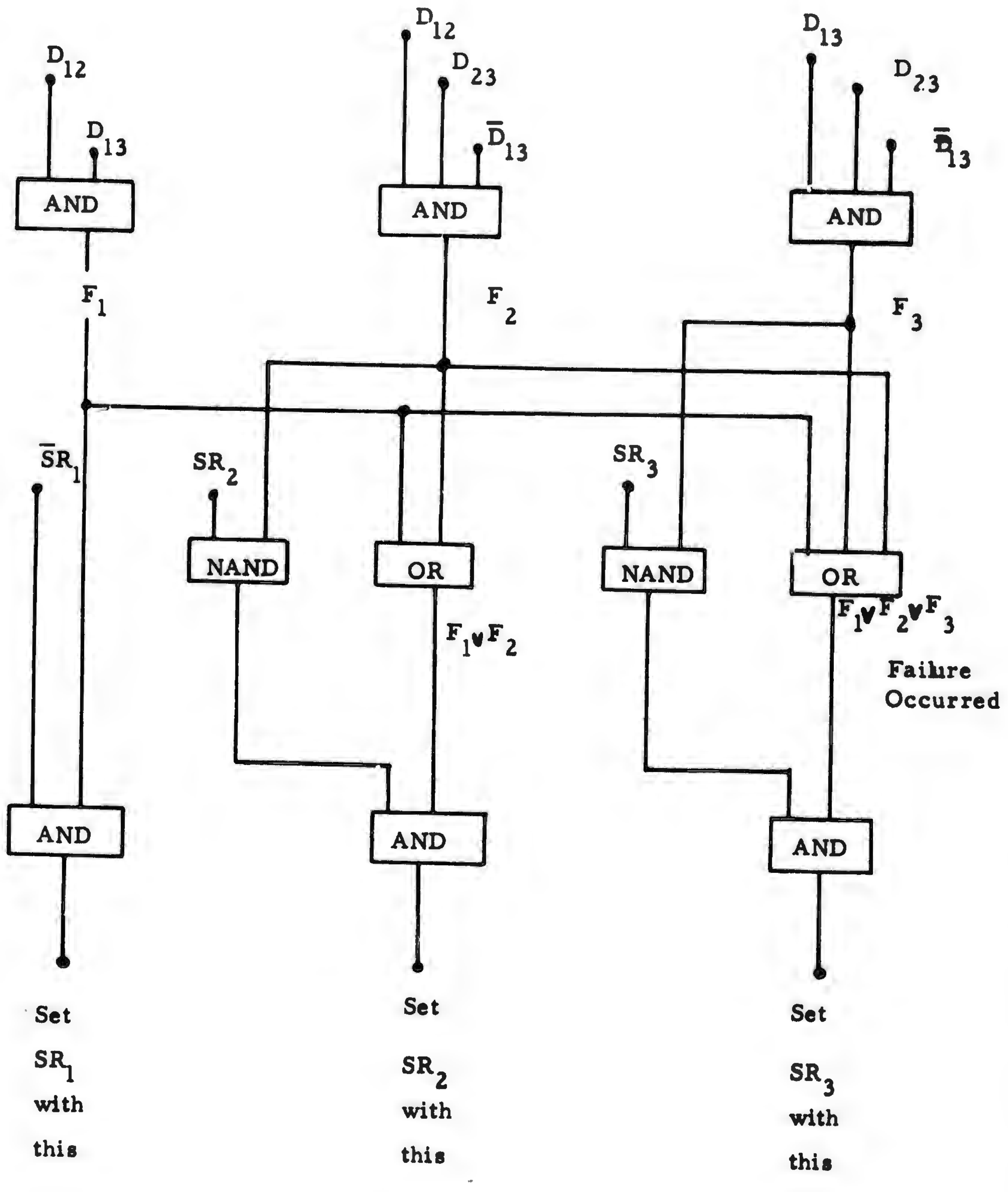


Figure 5. 31 Circuit used to set new SR state after a failure occurs.

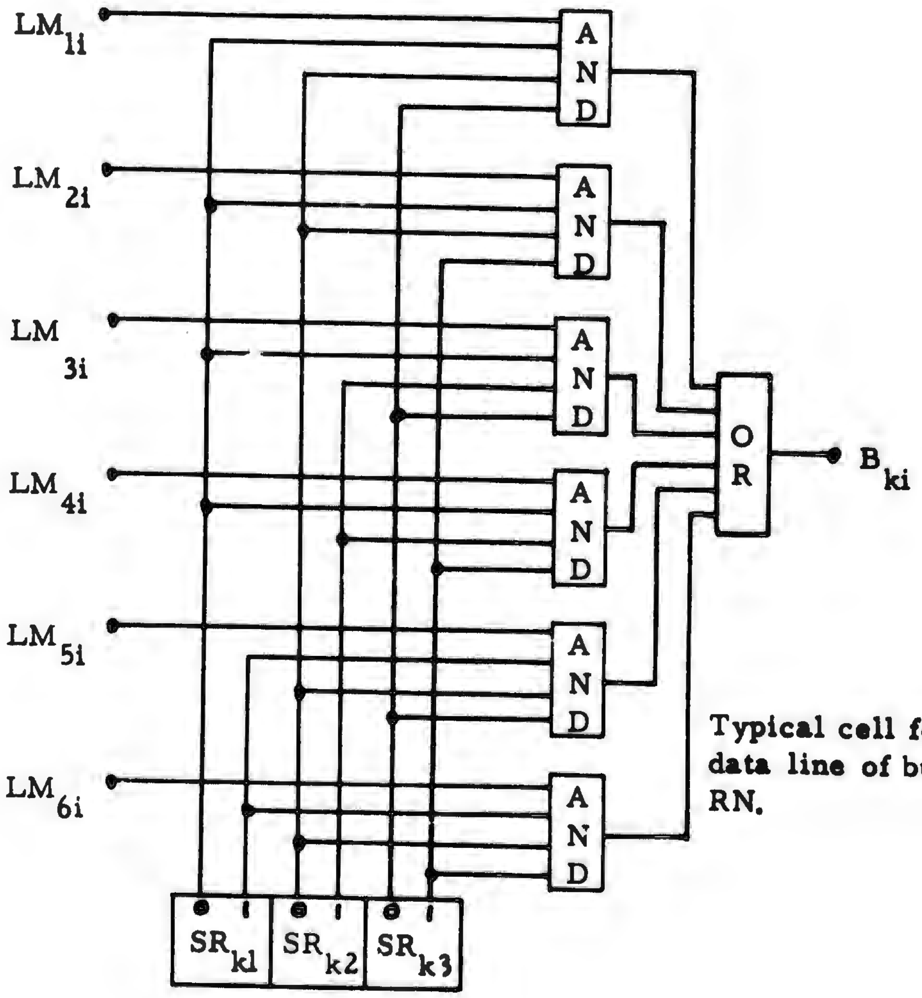
5.4.6.4 Reconfiguration: General Case $n > 4$. The RN is constructed so that each buss B_k, $k=1, 2, 3$, can be connected to any of the logic modules LM_j, $j=1, 2, \ldots, n$. The SR contains three segments SR_k, each containing $\lceil \log_2 n \rceil$ bits. When the contents of SR_k equals $j-1$, buss line k is connected to module j. Allowing each buss to span the complete set of modules simplifies the reconfiguration algorithm to be presented next. Thus, although it means a larger SR, the net circuitry in both the SR and its reconfiguration mechanism becomes small. The encoding of the states in the SR_k was also determined by the ease of specifying this algorithm: the encoding could be changed if it facilitated implementation in some particular technology. Figure 5.32 shows the SR states and their mapping interpretation together with a typical RN cell for the specific value $n=6$. The general case is merely an extension involving more states in the SR table, therefore, more AND gates in the RN cell.

It is convenient to carry along two auxiliary registers to speed the reconfiguration after a failure. The first, called MASK, contains n bits with $MASK_i = 1$ if and only if LM_i has failed. The second, called LAST, contains a binary number specifying the last failed module which the algorithm has tried to use. Initially LAST $= 0$ and remains so until the $n-2$ module failure, i.e. when a module fails and there are no more untried spares. The reconfiguration algorithm carried out after a failure occurs must perform the following three register updates (as specified in Iverson notation),

	LM_1	LM_2	LM_3	LM_4	LM_5	LM_6
B_1	000	001	010	011	100	101
B_2	000	001	010	011	100	101
B_3	000	001	010	011	100	101

SR_{11}	SR_{12}	SR_{13}
SR_{21}	SR_{22}	SR_{23}
SR_{31}	SR_{32}	SR_{33}

Status Register States



Typical cell for the ith data line of buss k in the RN.

Figure 5.32 SR and RN for n = 6

```
            →((~∨/LAST)∧∨/(N-1)=2⊥SR)/STARTLAST
            LAST←((⍴LAST)⍴2)⊤((↑MASK/⍳⍴MASK),0)[(MASK/⍳⍴MASK)⍳2⊥LAST]
            →INSERTMASK
 STARTLAST:LAST←((⍴LAST)⍴2)⊤(MASK/⍳⍴MASK)[1]
INSERTMASK:MASK[1+2⊥SR[F/⍳3;]]←1
            SR[F/⍳3;]←¯1+2⊥LAST
```

where F is as in Section 5.4.6.3 and SR is viewed as a binary matrix with the register SR_k as row k.

Figure 5.33 contains the sequence of SR states for a typical set of failure occurrences when n=6. As the first three failures occur in LM_3, LM_1 and LM_4 the usual shifting to the next unused module takes place. Then when LM_6 fails it is necessary to back up and try to use one of the previously failed modules. (As was stated in Section 3, this is the way modules which only experienced transient failures can be recovered.) The last three state mappings in Figure 5.33 show the progression of retries using LM_1, then LM_3, and finally LM_4, which is assumed to function correctly. As before it would be possible to use a control signal to inhibit cycling by blocking F when it is ascertained that n-2 modules have solid failures. The same comments made in Section 3 concerning time and speed of reconfiguration apply here.

The register LAST may be visualized as a ring counter as is shown in Figure 5.34 for the case n=6. Each state is triggered to the next state after a failure occurs. Before releasing from the initial state 000, one of the SR_k must indicate that the last module is currently being used. When a failure triggers a change in state the register proceeds forward only stopping at a state for which the MASK control bit is a 1.

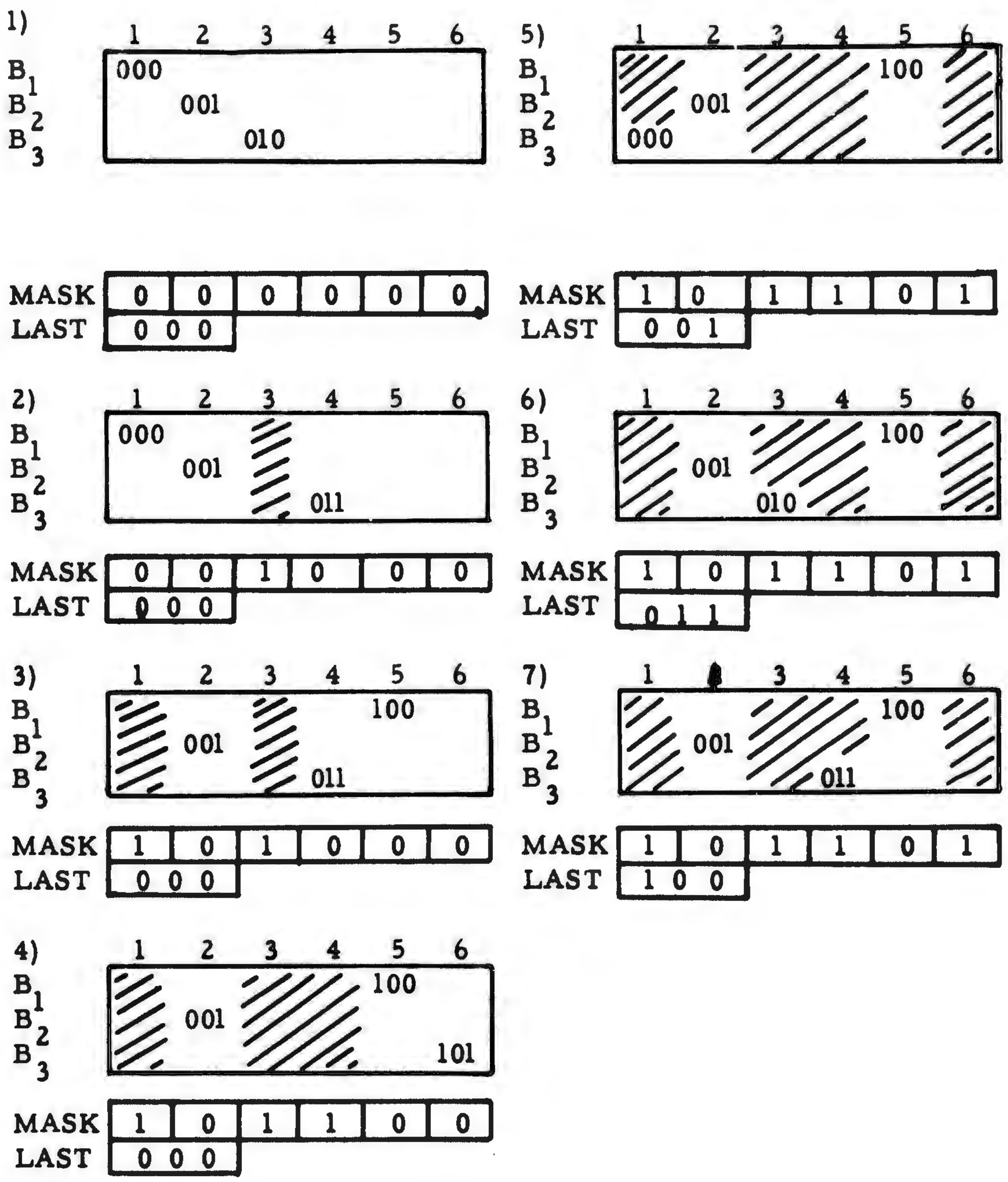


Figure 5.33 Sequencing of SR when n = 6 for a typical set of failure occurrences

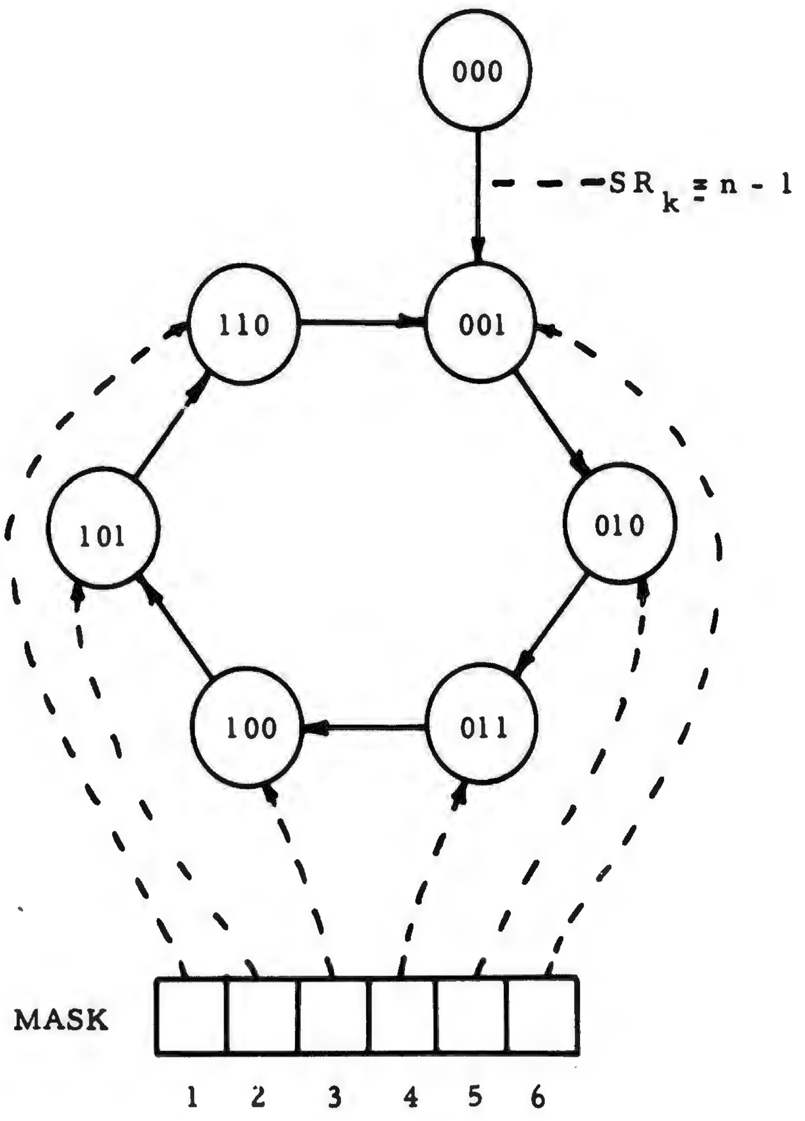


Fig. 5.34 LAST register states viewed as a controlled ring counter.

5.4.6.5 Reliability. Let P be the probability of failure for a logic module and its input voter (failures in V_i are indistinguishable from failures in LM_i). Then if the SR and RN function perfectly, the reliability of a TMR/Sparing configuration as shown in Chapter 4 is,

$$R = 1 - n P^{n-1} (1 - P)$$

and the probability of failure approaches nP^{n-1}. If the SR and RN do fail, it requires a specific combination of LM failures to bring the system down; i.e., an event with a low probability of occurrence. Further, it is possible to design the SR and RN, using redundancy, so that even when single failures occur, correct operation by these devices ensues.

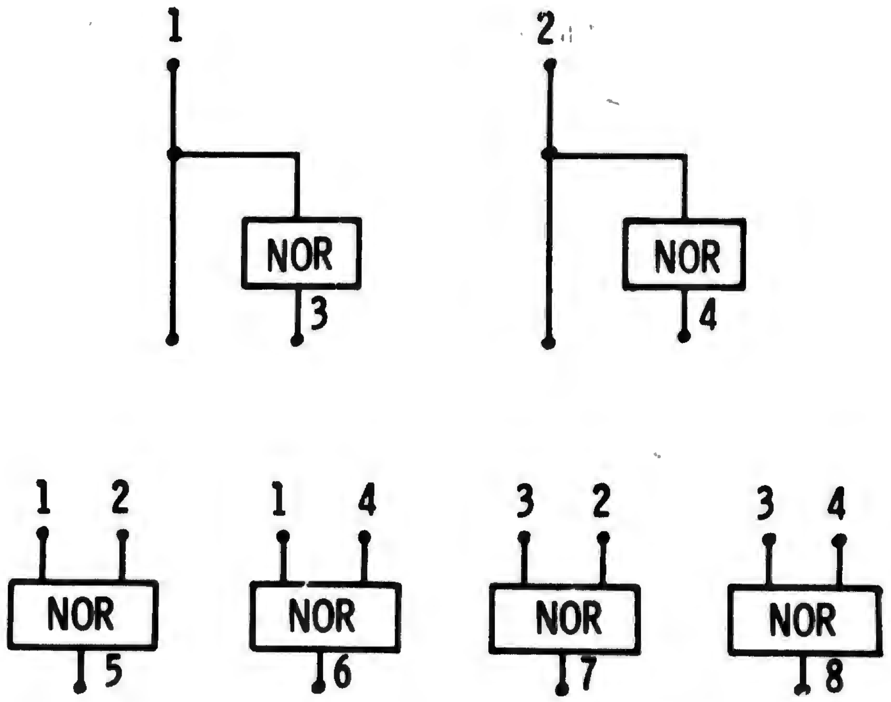
All these reliability calculations are somewhat pessimistic when the vector nature of the data paths are considered. For many situations where two, or even all three, of the TMR modules have failed, correct operation can occur. What is required for system breakdown is that two (or three) modules have erroneous outputs in the same component of the vector of output lines.

5.4.7 Design of Completely Checked Decoders

The operation-code decoder, address decoder, and many other such functions in a computer can be characterized by a circuit which accepts an n bit input and then causes exactly one of 2^n possible lines to be brought up. In what follows, unless otherwise stated, "failure" will mean a single stuck-at-1 or stuck-at-0 failure in a logic block. Then it will be shown that it is possible to design decoders with 100% dynamic decoder error detection and complete error detection in the checkers through diagnosis.

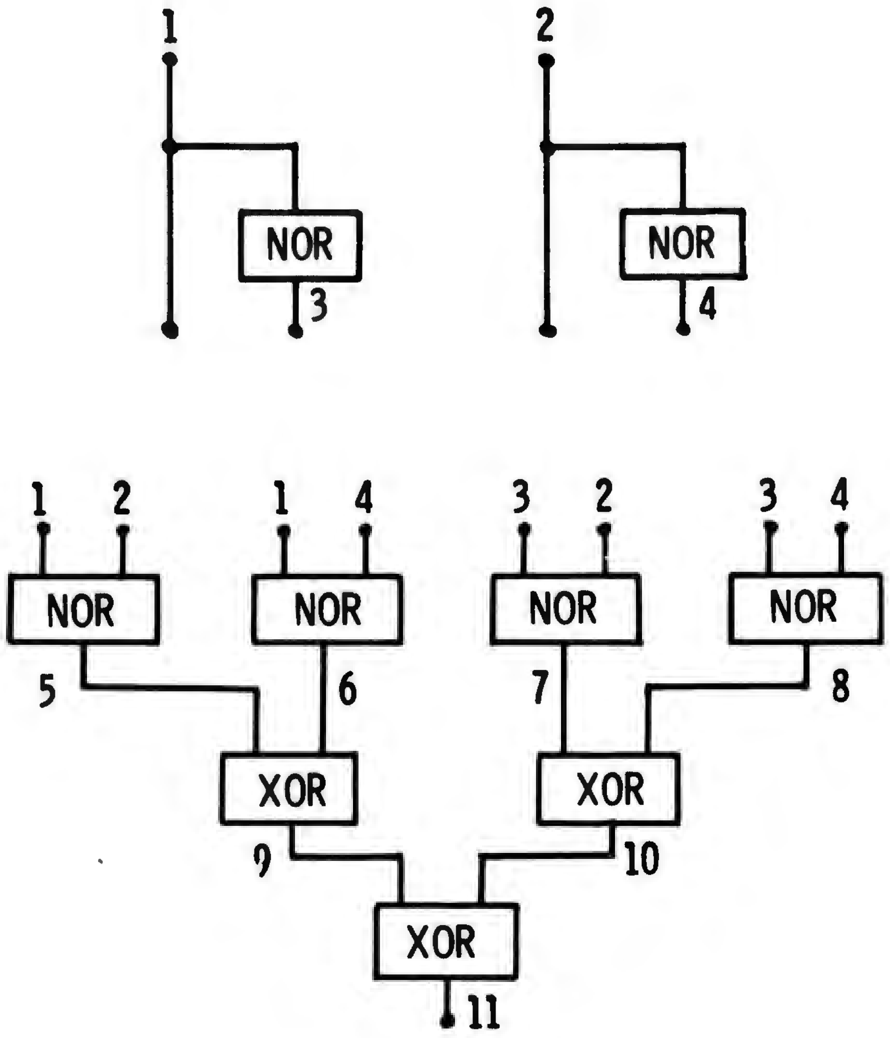
For the special case $n=2$ a typical decoder, constructed from NOR gates, is shown in Fig. 5.35a extension to $n>2$ is straightforward. Since all results to be developed concerning testability are independent of n, $n>2$, this circuit will be used as the starting point for all subsequent analysis. Various techniques for providing dynamic checking will be proposed and examined for effectiveness using the TESTDETECT algorithm. Then trade-offs concerning the amount of extra equipment required and the degree of checking obtained can be weighed.

The circuit in Fig. 5.35a produces erroneous outputs whenever any of its components fails. Placing an odd-parity check on its output as in Fig 5.35b provides the ability to dynamically check all failures occurring in the NOR gates and in XOR gates 9 and 10. Failures in the two input lines cannot be checked since they will always produce another legal or permissible input. The output XOR cannot be tested for a stuck-at-1 failure; this, in itself, does no harm since the circuit would continue to function correctly, but any second failure would go undetected.



Basic decoder for n = 2.

Figure 5.35a



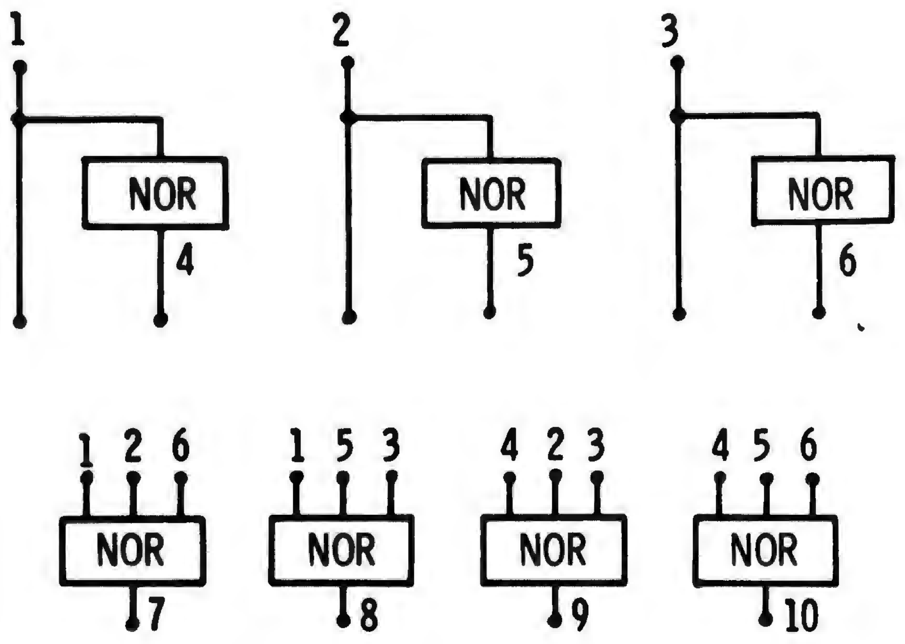
1	2	1	2	3	4	5	6	7	8	9	0	1
0	0			0	0	0	1	1	1	0	1	0
0	1			0	1	1	0	1	1	0	1	0
1	0			1	0	1	1	0	1	1	0	0
1	1			1	1	1	1	0	1	1	0	0

Checked decoder for n = 2.

Figure 5.35b

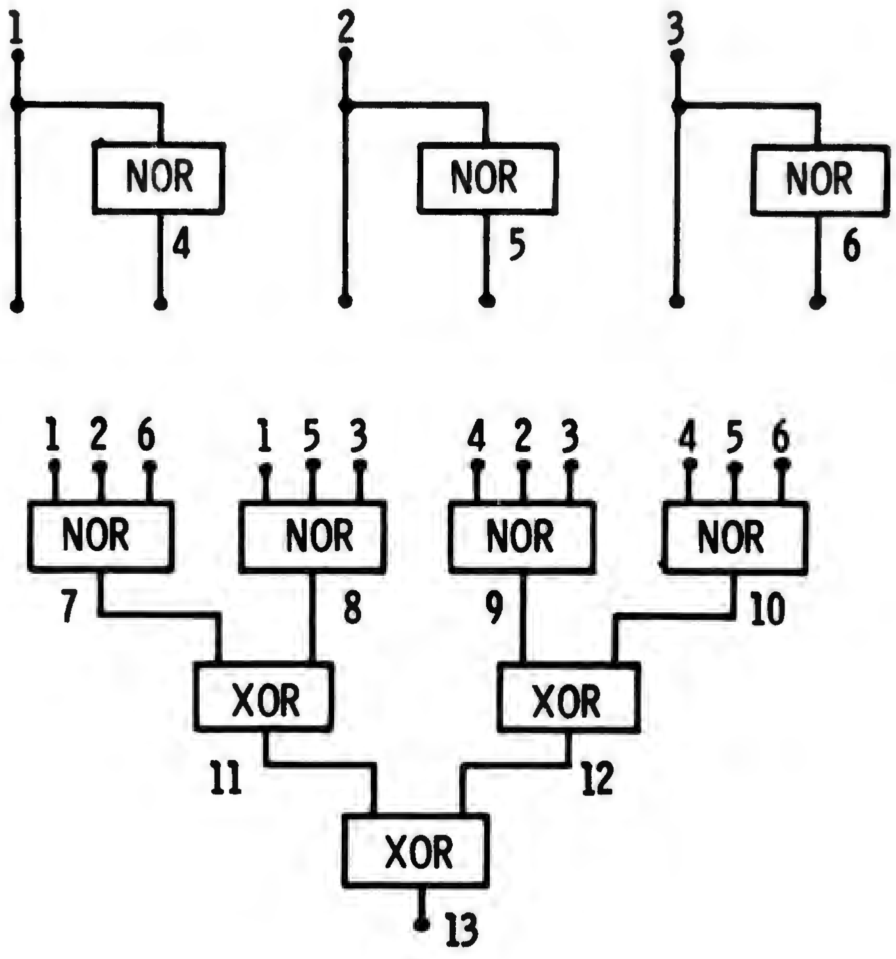
If an extra odd-parity line is added to this input, the basic decoder appears as in Fig. 5. 36a. All failures except a single inverter stuck-at-0 will cause errors in the output. In Fig. 5. 36b a parity check circuit has been placed across the four output lines. Now all failures except an inverter stuck-at-0 (which does not produce erroneous output) and the output XOR stuck-at-1 (which only causes trouble when a second failure occurs) are dynamically checked. These four unchecked failures can be tested by forcing the illegal even-parity input of all zeros to occur. Thus the proposed mode of operation would be to let the checker detect all failures which can produce erroneous outputs and then periodically run the illegal parity input to test for those unchecked failures which only become a problem in the presence of a second failure.

By adding a second parity check circuit across the outputs of the inverters, as in Fig. 5. 37 a dynamic test of all decoder failures can be obtained. As before, it is necessary to periodically use the all-zero-input test to check the two parity circuit outputs. The price paid for obtaining this extra check on the inverters is that two outputs must be scanned. This obstacle is overcome in Fig. 5. 38 by ANDing the two checking outputs. Now all decoder failures are dynamically checked at the single AND output. However, even when illegal parity inputs are used to diagnose the checking circuitry there is no way to test blocks 13 and 16 for stuck-at-1 or block 15 stuck-at-0; failures which in themselves do not produce errors but do mask detection of second failures. The solution to this problem involves insertion of an OR of the two checker outputs in the circuit as shown in Fig 5. 39. Under normal operation the single output of the AND is monitored to check for a decoder failure.



Basic decoder with an input parity line.

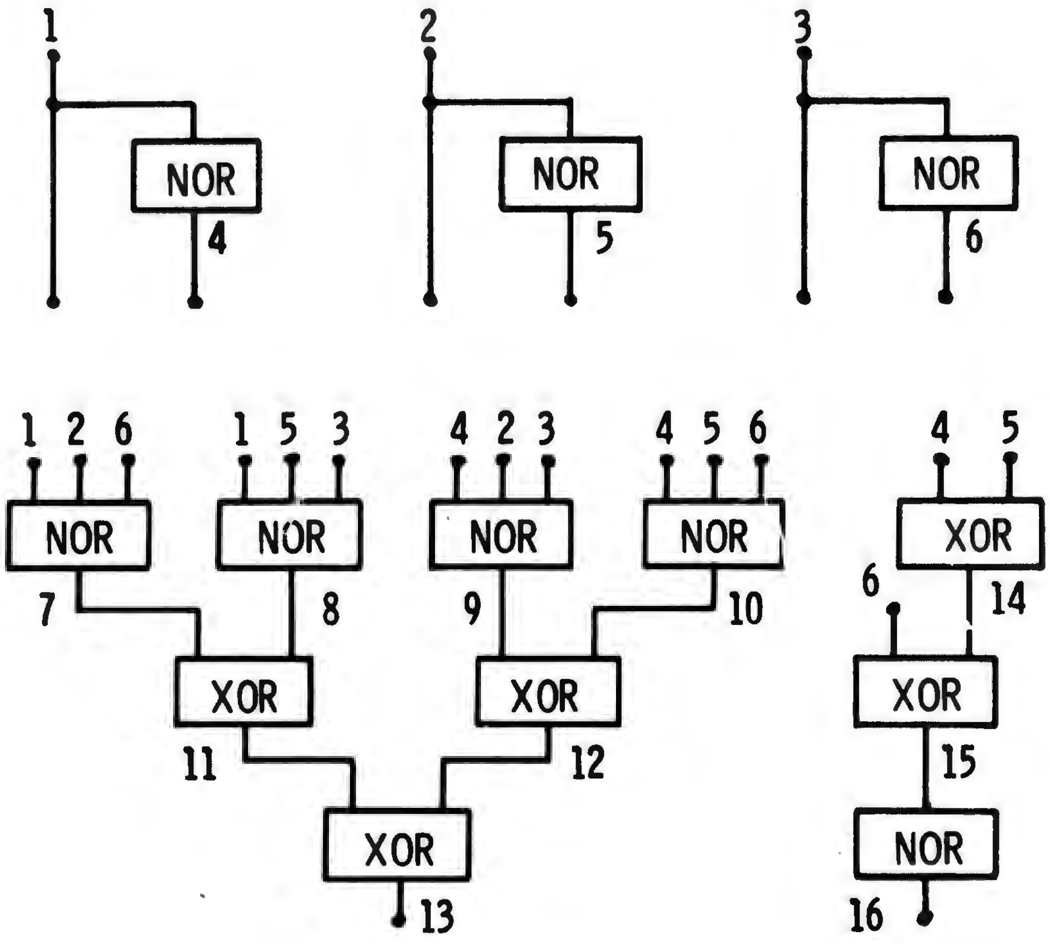
Figure 5. 36 a



1	2	3	1	2	3	4	5	6	7	8	9	0	1	2	3
0	0	1	1	1	0			1	0	1	1	1	0	1	0
0	1	0	1	0	1		1		1	0	1	1	0	1	0
1	0	0	0	1	1	1			1	1	0	1	1	0	0
1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	0
0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1

Checked decoder with parity on input.

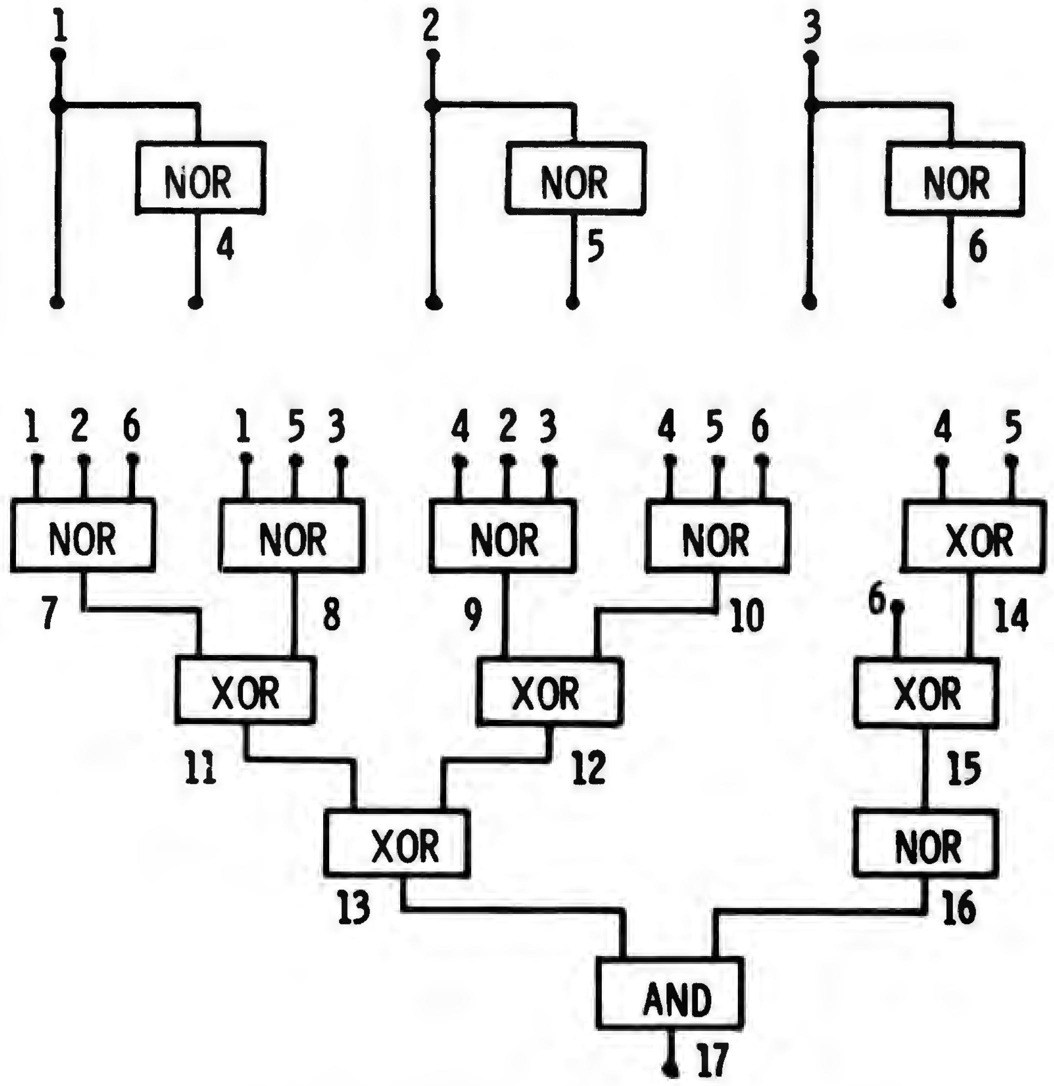
Figure 5. 36 b



1	2	3	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
0	0	1	1	1	0	0	0	1	0	1	1	1	0	1	0	1	1	0
0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1	0
1	0	0	0	1	1	1	0	0	1	1	0	1	1	0	0	0	1	0
1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	0	1	1	0
0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	1

Doubly checked decoder.

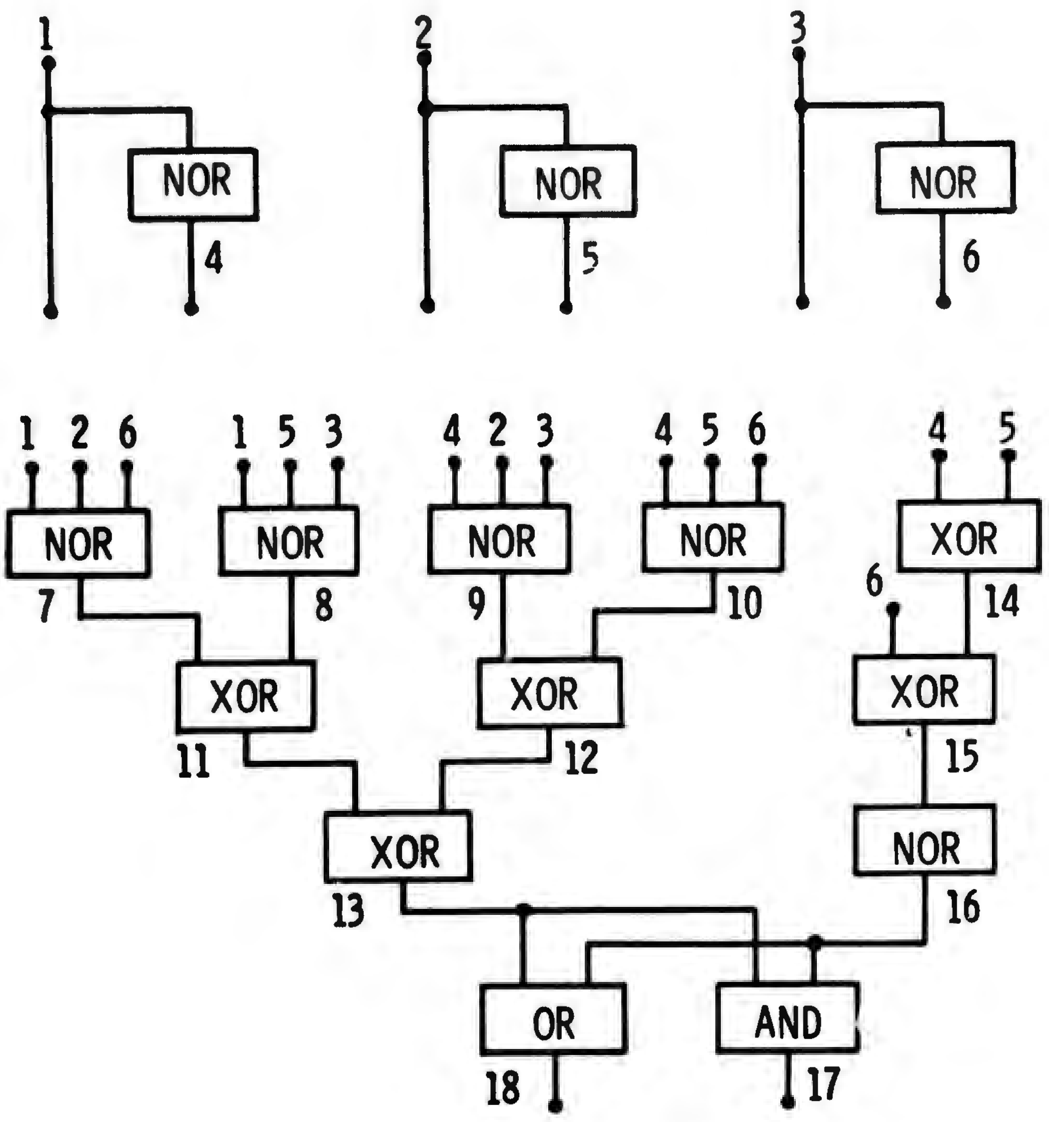
Figure 5.37



1	2	3	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7
0	0	1	1	1	0	0	0	1	0	1	1	1	0	1	0	1	1	0	0
0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1	0	0
1	0	0	0	1	1	1	0	0	1	1	0	1	1	0	0	0	1	0	0
1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	0	1	1	0	0
0	0	0	1	1	1	0	0	0											1

Double check with a single output.

Figure 5.38



1	2	3	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8
0	0	1	1	1	0	0	0	1	0	1	1	1	0	1	0	1	1	0	0	0
0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1	0	0	0
1	0	0	0	1	1	1	0	0	1	1	0	1	1	0	0	0	1	0	0	0
1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	0	1	1	0	0	0
0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1

Double check, double output decoder.

Figure 5.39

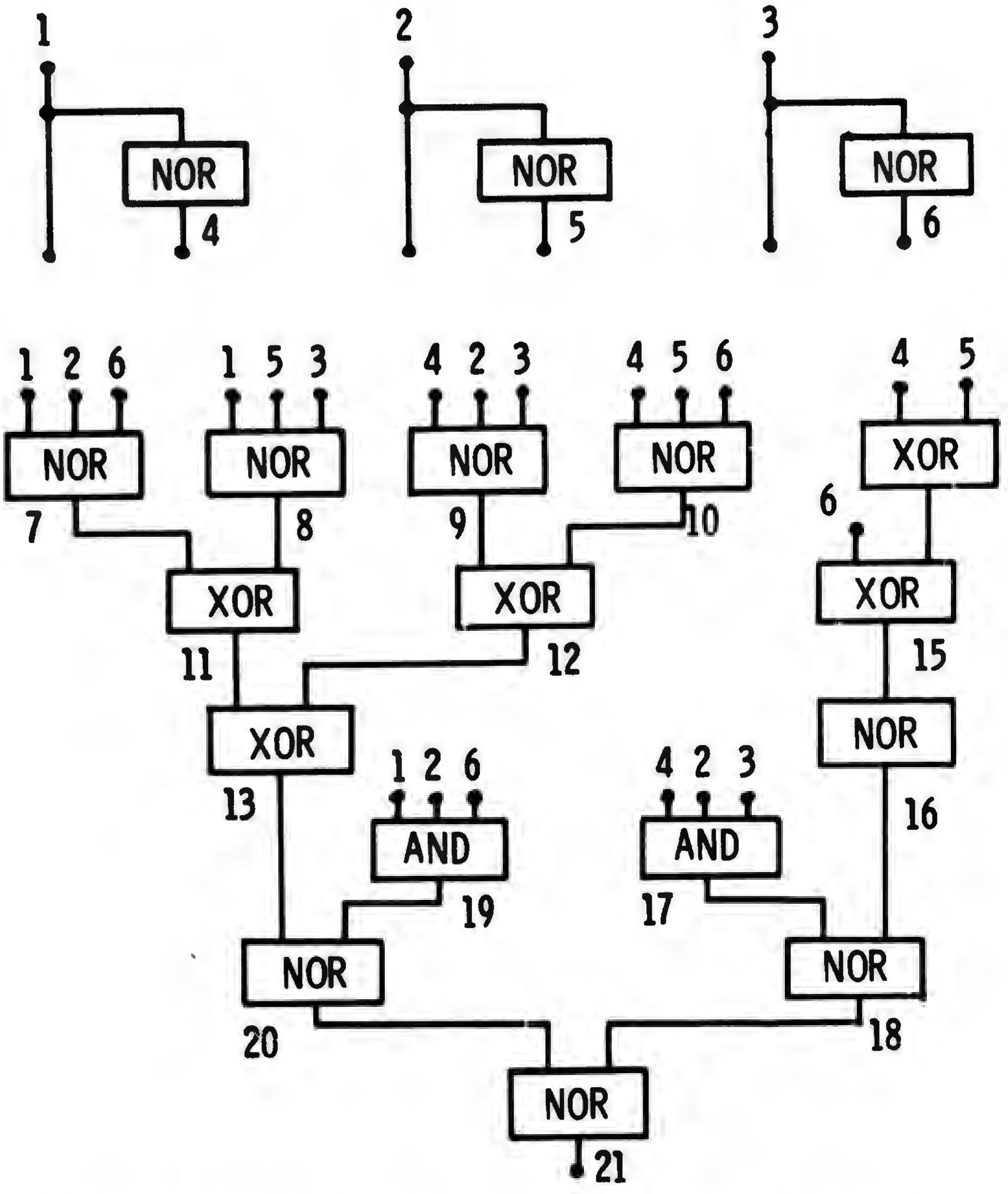
At periodic intervals the checker outputs are diagnosed by placing all zeros and then all ones on the inputs; in the former case both checker outputs should be 0 and in the latter, 1.

A second method for improving the circuit of Fig. 5.38 is shown in Fig. 5.40. A pair of illegal even-parity tests are used to selectively test each of the two checkers independently of the other; the extra AND gates at 17 and 19 are used to inhibit one of the checker outputs. Only one circuit output is used for both the dynamic decoder checks and the diagnosis of the checkers. The basic problem with this circuit is there is no way to test the AND gates used for selection purposes, 17 and 19, for stuck-at-0 failures. This is not too heavy a burden since unchecked system errors only result when the following sequence of errors occurs; one of the AND's fails stuck-at-0, then the output of the checker in the opposite branch fails stuck-at-1, and finally, a failure in the decoder occurs.

The question now arises as to exactly how much circuitry is required to do the checking illustrated in each of the previous examples. Let n be the number of bits to be decoded. Then a basic decoder, as in Fig. 5.35a requires $n+2^n$ gates. Including an input parity line requires 1 extra inverter be added to the circuitry. Checking the 2^n decoder outputs using two input XOR's in a tree arrangement uses

$$\sum_{i=1}^{n} 2^{n-i} = 2^n - 1 \quad \text{gates.}$$

Checking the outputs of the n+1 inverters requires n+1 XOR gates. With this data, the amount of hardware required for each of the previous 8 circuit types and the resulting fractional increase in hardware over the basic decoder is tabulated in Table 5.1. For all of the checked decoders, Figures 5.36-5.40, this extra investment in



1	2	3	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0	0	1	1	1	0	0	0	1	0	1	1	1	0	1	0	1	1	0		1		1	0
0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	0	1	0		1		1	0
1	0	0	0	1	1	1	0	0	1	1	0	1	1	0	0	0	1	0		1		1	0
1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	0	1	1	0		1		1	0
0	1	1	1	0	0	0			1	1	1	1	1	1	1						1	0	1
1	1	0	0	0	1	1	1	0								1	0	1	1	0			1

Selectively checked decoder.

Figure 5.40

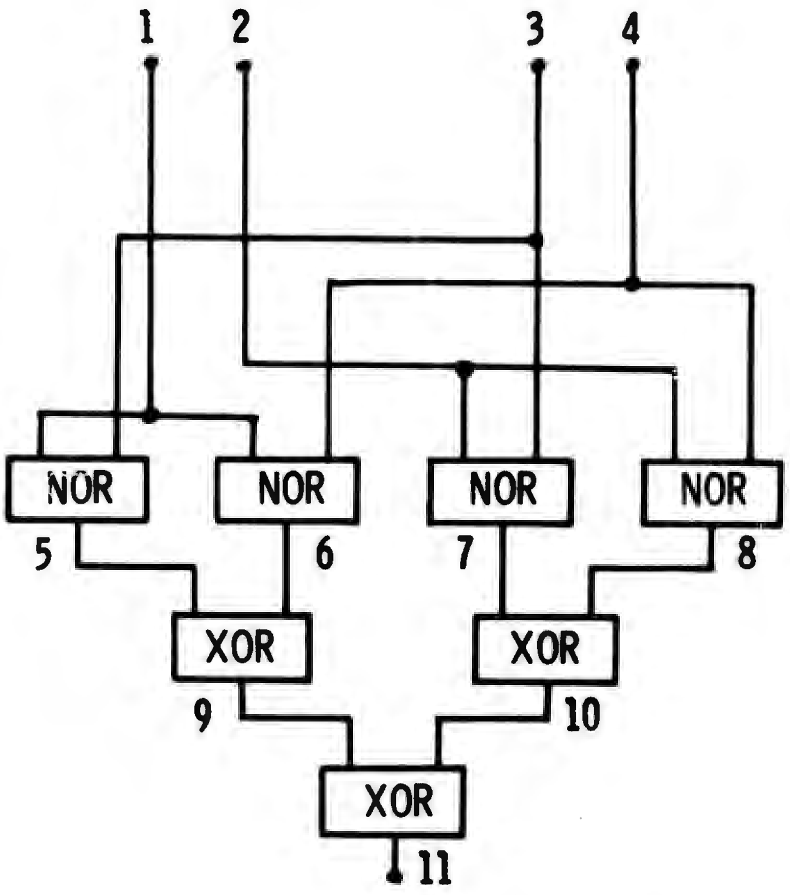
Fig.	Total hardware in circuit	Fractional increase
1	$n + 2^n = B$	0
2	$(n + 2^n) + (2^n - 1)$	$1 - (n + 1)/B$
3	$(n + 2^n) + 1$	$1/B$
4	$(n + 2^n) + 1 + (2^n - 1)$	$1 - n/B$
5	$(n + 2^n) + 1 + (2^n - 1) + (n + 1)$	$1 + 1/B$
6	$(n + 2^n) + 1 + (2^n - 1) + (n + 1) + 1$	$1 + 2/B$
7	$(n + 2^n) + 1 + (2^n - 1) + (n + 1) + 2$	$1 + 3/B$
8	$(n + 2^n) + 1 + (2^n - 1) + (n + 1) + 5$	$1 + 6/B$

Table 5.9 **Hardware required in the checked decoders.**

hardware is nearly equal to the hardware in the basic decoder. One might ask why the decoder isn't just duplicated. The obvious answer is that 1) 2^n XOR gates would still be required to test for disagreements, 2) input failures would not be detected unless each decoder had its input generated independently, and 3) diagnosis of the 2^n XOR gates would still be required.

Figure 5.41a shows a decoder with two-rail inputs and fully checked outputs. Even with this circuit configuration it becomes necessary to force illegal input patterns (all zeros, say) in order to test the checker output. Further, the amount of hardware required for the checking is comparable with that used in the previous examples employing parity checked inputs. In fact, if two rail outputs were also required from the decoder the hardware costs would be very much larger.

It is important to realize that once a circuit and its checking equipment has been designed, minor perturbations in the structure can radically reduce the checking coverage. The circuit in Fig. 5.42 was obtained from the one in Fig. 5.36 by replacing the decoder's NOR gates by AND gates with the inputs complemented (De Morgan's Law). Now the inverters are not checked for stuck-at-1 failures. As before, these unchecked inverter failures do not produce erroneous outputs but do prevent checking of subsequent second failures. However, in this case it takes two special even-parity inputs to diagnose these inverters, as seen in the bottom of the TESTDETECT table, neither of which is the all zero input. The conclusion is that one must be extremely careful in changing circuit topologies even when the functional characteristics of the circuit remain fixed.



1	2	3	4	1	2	3	4	5	6	7	8	9	0	1
0	1	0	1	1	0	1	0	0	1	1	1	0	1	0
0	1	1	0	1	0	0	1	1	0	1	1	0	1	0
1	0	0	1	0	1	1	0	1	1	0	1	1	0	0
1	0	1	0	0	1	0	1	1	1	1	0	1	0	0
0	0	0	0					0	0	0	0	1	1	1

(a)

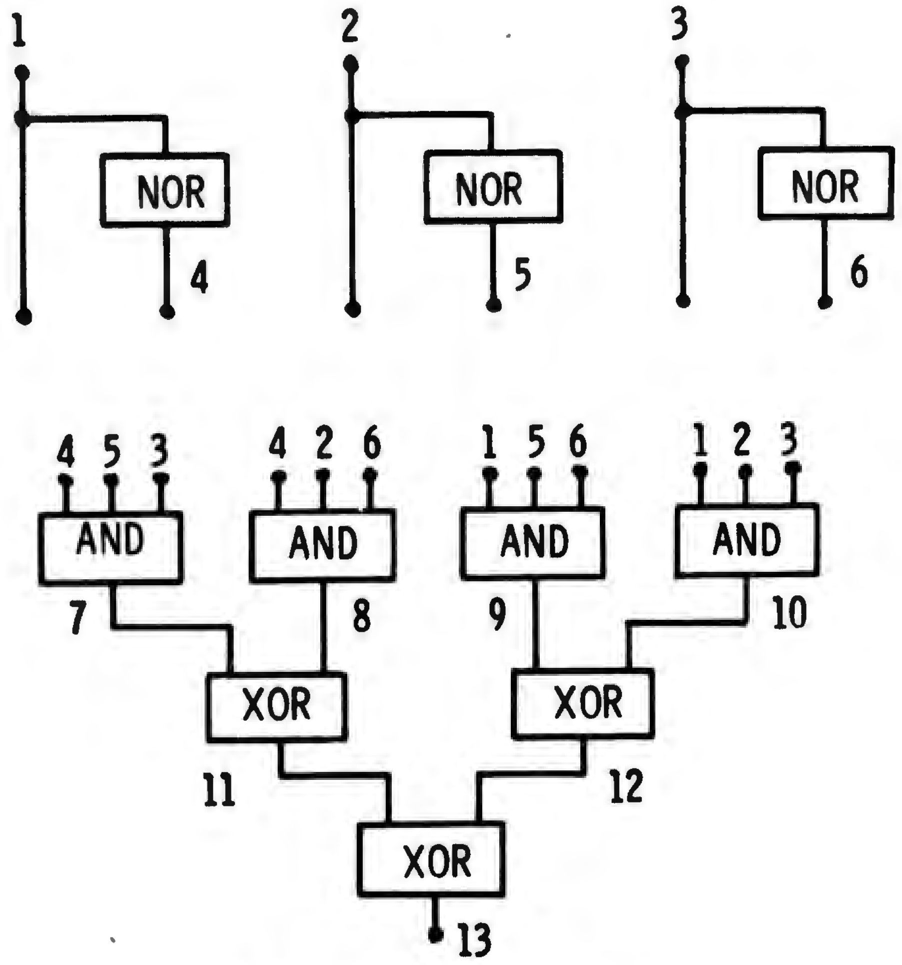
$$\text{HARDWARE} = 2^n + (2^n - 1)$$

$$\text{INCREMENT} = 1 - \frac{2n-1}{B}$$

(b)

Two rail decoder with checking.

Figure 5.41



1 2 3	1	2	3	4	5	6	7	8	9	0	1	2	3
0 0 1	1	1	0	0	0		0	1	1	1	0	1	0
0 1 0	1	0	1	0		0	1	0	1	1	0	1	0
1 0 0	0	1	1		0	0	1	1	0	1	1	0	0
1 1 1	0	0	0				1	1	1	0	1	0	0
0 0 0	1	1	1				1	1	1	1	1	1	1
0 1 1	1	0	0		1	1	1	1	1	1	1	1	1
1 0 1	0	1	0	1		1	1	1	1	1	1	1	1
1 1 0	0	0	1	1	1		1	1	1	1	1	1	1

Effect of a change in topology.

Figure 5.42

5.4.8 Design of Circuits to Generate Diagnostic Tests.

One of the initial steps necessary for computer testing is to test the parity check circuitry so it can be used in further test proceedures (the Bootstrap approach). Since the step is initial, test storage must be minimized, and the addition of circuitry to generate these tests should be considered. A specific test and test circuitry was devised for a specific (but typical) parity test circuit. The methods used can easily be generally applied, and will be stated later.

Figure 5.43 shows module #21, a 3-way exclusive OR circuit in a standard technology. If the exclusive OR desired is written as

$$f = A \text{ EOR } B \text{ EOR } C$$

then the following necessary input connections must be made:

Input Connection	Variable
D04	$\overline{A}$
C04	$\overline{B}$
B04	$\overline{C}$
C01	B
B01	A
D01	C

The output f is given on D02 and $\overline{f}$ on D03.

To implement a nine bit parity checking circuit

$$g = A_1 \text{ EOR } A_2 \text{ EOR } A_3 \text{ EOR } A_4 \text{ EOR } A_5 \text{ EOR } A_6 \text{ EOR } A_7 \text{ EOR } A_8 \text{ EOR } A_9$$

use a cascade of #21 modules with A_1, A_2, A_3 properly connected to the first module, A_4, A_5, A_6 to the second and A_7, A_8, A_9 to the last. The outputs from these modules are

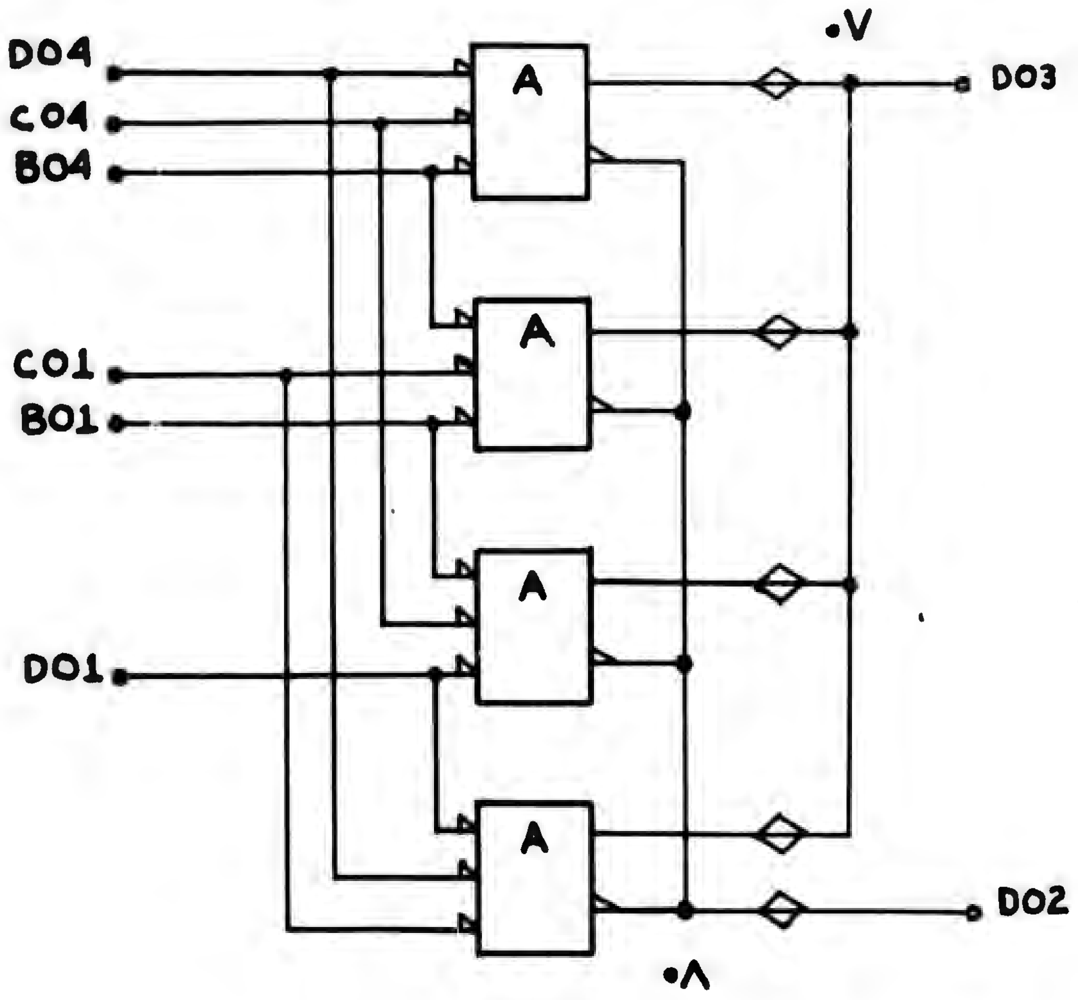
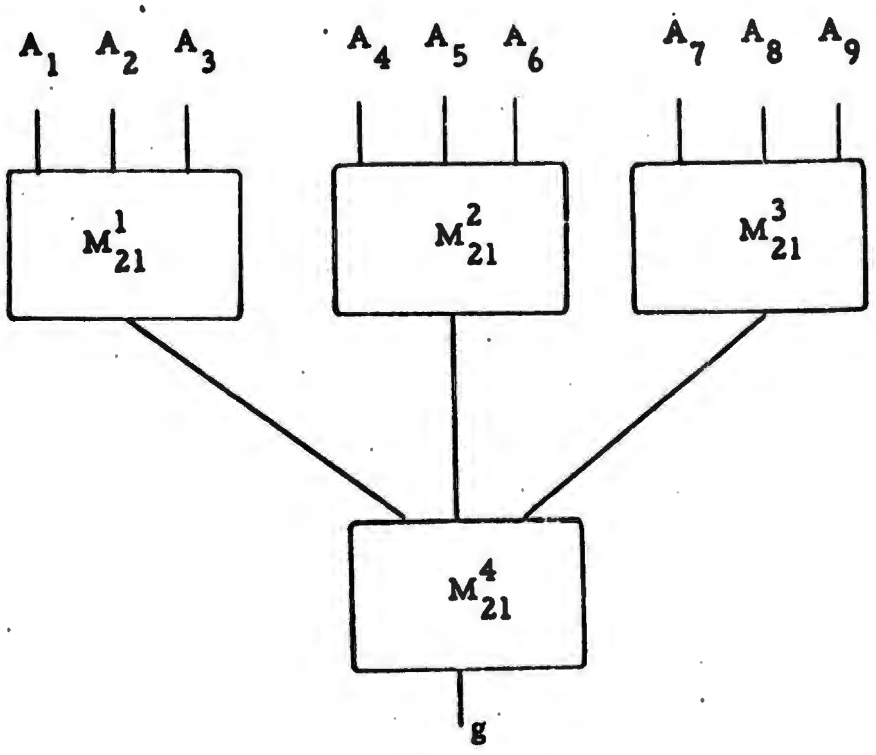


Fig. 5.43 MODULE #21.

connected to a fourth module to produce g and $\bar{g}$.



The first step is to produce a complete set of tests for a #21 module. The AND/OR direct logic representation for the module is

$$f = (A \cup B \cup C)(\bar{A} \cup \bar{B} \cup C)(\bar{A} \cup B \cup C)(A \cup \bar{B} \cup \bar{C})$$

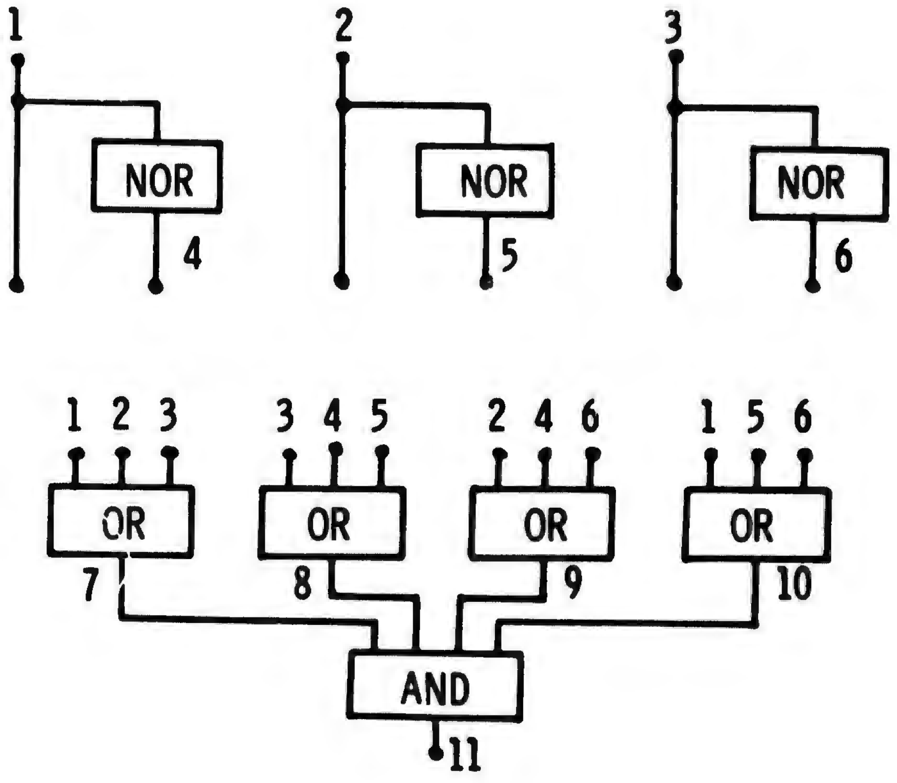
The minimum set of tests for a three way OR gate is given by (1, 0, 0), (0, 1, 0), (0, 0, 1) and (0, 0, 0). The minimum set of tests for a four way AND gate consists of (1, 1, 1, 1), (0, 1, 1, 1), (1, 0, 1, 1), (1, 1, 0, 1) and (1, 1, 1, 0).

Because of the input interconnections a complete set of tests for #21 module can be determined from Figure 5.44. In the figure A, B, C correspond to 1, 2, 3 respectively.

To derive a complete set of tests for the parity check circuit g, consider the four sets of tests necessary to test M_{21}^4, M_{21}^3, M_{21}^2, M_{21}^1. Choose the first test pattern necessary to test M_{21}^4 (pattern 100). This pattern may be produced by choosing pattern 100 for M_{21}^1, 000 for M_{21}^2 and 000 for M_{21}^3. The inputs for this first test pattern for g are (1000 000 000). Check the fact that the appropriate patterns have been used in the sets of tests for M_{21}^1, M_{21}^2 and M_{21}^3. Choose the second test pattern necessary to test M_{21}^4 (pattern 010). This pattern may be produced by choosing pattern 000 for M_{21}^1, 100 for M_{21}^2 and 110 for M_{21}^3 (unused patterns are choosen when possible). This gives the second input test pattern for g, (000 100 110). By continuing in this way it can be shown that a set of 8 input test patterns will test the parity check circuit completely.

So far no extensive use has been made of the freedom of choice in choosing input patterns for the modules M_{21}^1, M_{21}^2, and M_{21}^3 to produce proper input patterns for M_{21}^4; and no use has been made of the fact that the sequence of tests applied to M_{21}^4 is arbitrary.

To reduce storage from 72 bits, try to produce one test pattern from the previous test pattern by shifting the first pattern 1 bit and forming the new bit from a combination of some bits in the previous pattern. Suppose the initial pattern contains three consecutive 1's and three consecutive 0's and

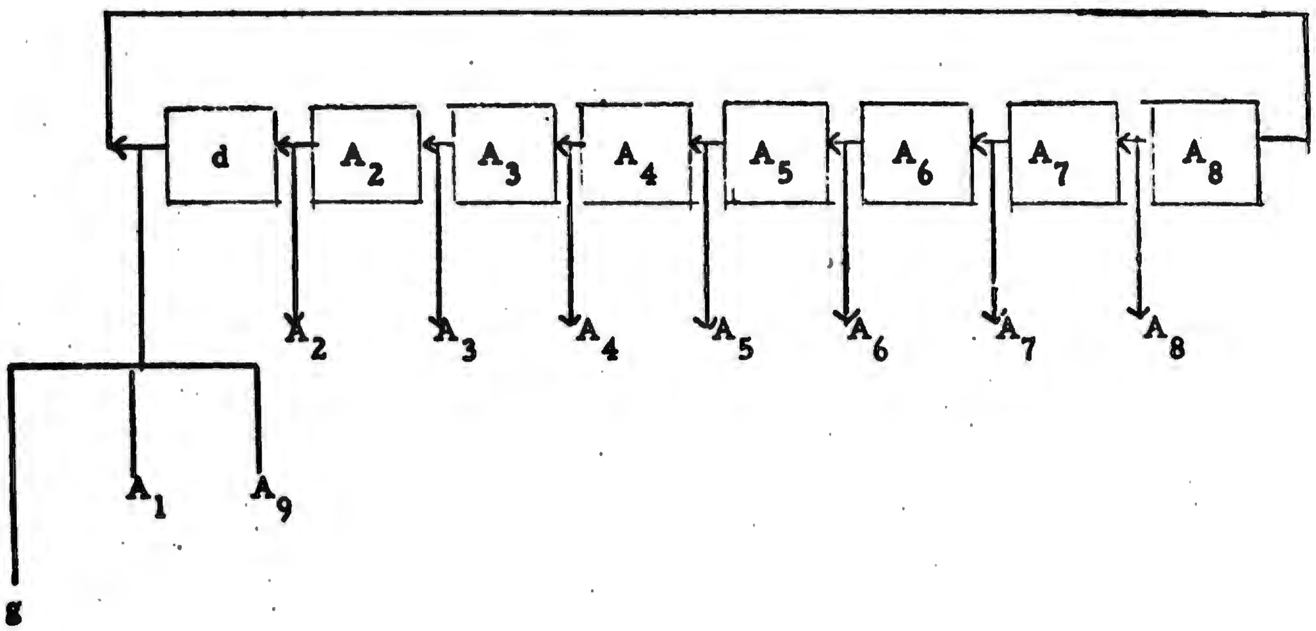


1	2	3	1	2	3	4	5	6	7	8	9	0	1
0	0	0	1	1	1				1				1
0	0	1	1	1	0	0	0		0	0	0	0	0
0	1	0	1	0	1	0		0	0	0	0	0	0
0	1	1	1	0	0		1	1				1	1
1	0	0	0	1	1		0	0	0	0	0	0	0
1	0	1	0	1	0	1		1			1		1
1	1	0	0	0	1	1	1			1			1
1	1	1	0	0	0				0	0	0	0	0

Fig. 5.44 PC - Parity Check.

these strings of 1's and 0's are of maximum length. Then the easiest way to produce a new bit from an old pattern is by end around shifting using the initial 8 bit pattern 1 0 1 1 1 0 0 0. When inserted into the proper circuitry and shifted this scheme defined a complete set of tests for g.

Circuitry:



The lines A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9 are used as input to the parity check circuitry, and the line g is used to test the output for error.

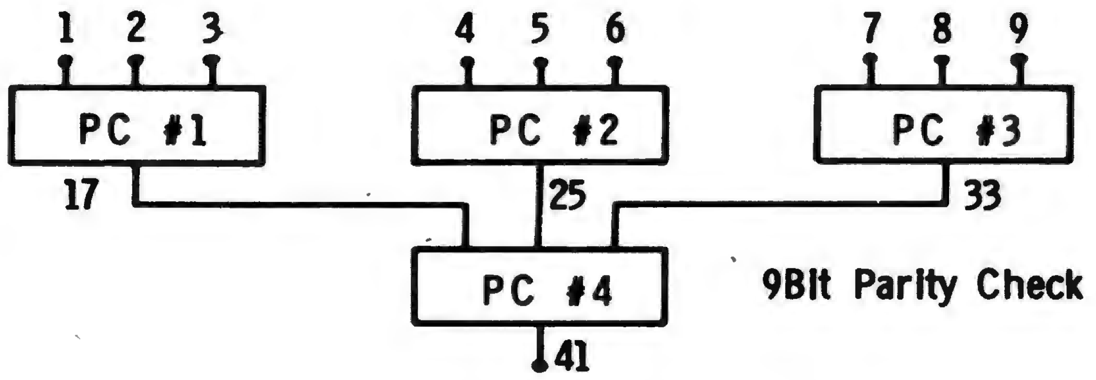
The basic pattern is 1 0 1 1 1 0 0 0. The tests and correct output are shown below.

Test Number	g	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
1	1	1	0	1	1	1	0	0	0	1
2	0	0	1	1	1	0	0	0	1	0
3	1	1	1	1	0	0	0	1	0	1
4	1	1	1	0	0	0	1	0	1	1
5	1	1	0	0	0	1	0	1	1	1
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	1	0	1	1	1	0	0
8	0	0	1	0	1	1	1	0	0	0

Figure 5. 45 shows the set of errors caught by each test. The numbering of lines is 1-9 for inputs, then x+6 for the pc#1 lines from Fig. 5. 44, then x+14 for pc#2 lines, then x+22 for pc#3 and x+30 for pc #4.

The basic rules (not developed to an algorithm) are:

1) Get a complete set of tests for a module.
2) Imbed the module in the tree structure.
3) Develop tests by specifying output from the module and choosing the set of module inputs which give this output. Choose each set of inputs as infrequently as possible.
4) Use these tests as a basis for tests with other desirable properties, using the freedom of choice in choosing tests, symmetries, human pattern recognition, etc.



```
1 2 3 4 5 6 7 8 9 | 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
------------------+----------------------------------------------------------------------------------
1 0 1 1 1 0 0 0 1 | 0 1 0 0 0 1 1 1 0 1   1     1   1 1 1     1     1 0 0   0 0 0 0 0 0 0   0 0 0 0 0
0 1 1 1 0 0 0 1 0 | 1 0 0 0 1 1 1 0 1   1 1       1 1   0 0 0 0 0 0 0 0   0 0 0 0 0 0   1 1       1 1
1 1 1 0 0 0 1 0 1 | 0 0 0 1 1 1 0 1 0       0 0 0 0 0       1       1 1   1     1   1   0 0 0 0 0 0 0
1 1 0 0 0 1 0 1 1 | 0 0 1 1 1 0 1 0 0 1 1     1     1 0 0   0 0 0 0 0   1 1       1 1 0   0 0 0 0 0 0
1 0 0 0 1 0 1 1 1 | 0 1 1 1 0 1 0 0 0   0 0 0 0 0 0 0 0   0 0 0 0 0 0       0 0 0 0 0       0 0 0 0 0
0 0 0 1 0 1 1 1 0 | 1 1 1 0 1 0 0 0 1       1       1 1   1     1   1 1 1     1     1       1       1
0 0 1 0 1 1 1 0 0 | 1 1 0 1 0 0 0 1 1 0 0   0 0 0 0 0   1 1       1 1   0 0 0 0 0 0 0 1   1     1   1
0 1 0 1 1 1 0 0 0 | 1 0 1 0 0 0 1 1 1 0   0 0 0 0 0 0       0 0 0 0 0       1       1 1 1     1     1
```

Fig. 5.45 9 Bit Parity Check.

5.5 ARC Operation

5.5.1 Normal Operation

5.5.1.1 General Procedures

When the ARC is carrying out procedures, it acts similarly to any computer. However, it is emphasized - again - that the control of a computing system lies in the set of programs and procedures that are used for total system control. For example; J. McCarthy[2] defines the "state vector" of a program as all values of all variables and all other information that together with the program itself determines the further course of the computation. Most functions will occur, and methods to detect malfunctions and restore a correct environment must be devised. The possibility of a malfunction at any time increases the information in the "vector". Computer configuration information must be included. A first step is to determine an overall strategy. Recovery from an error involves:

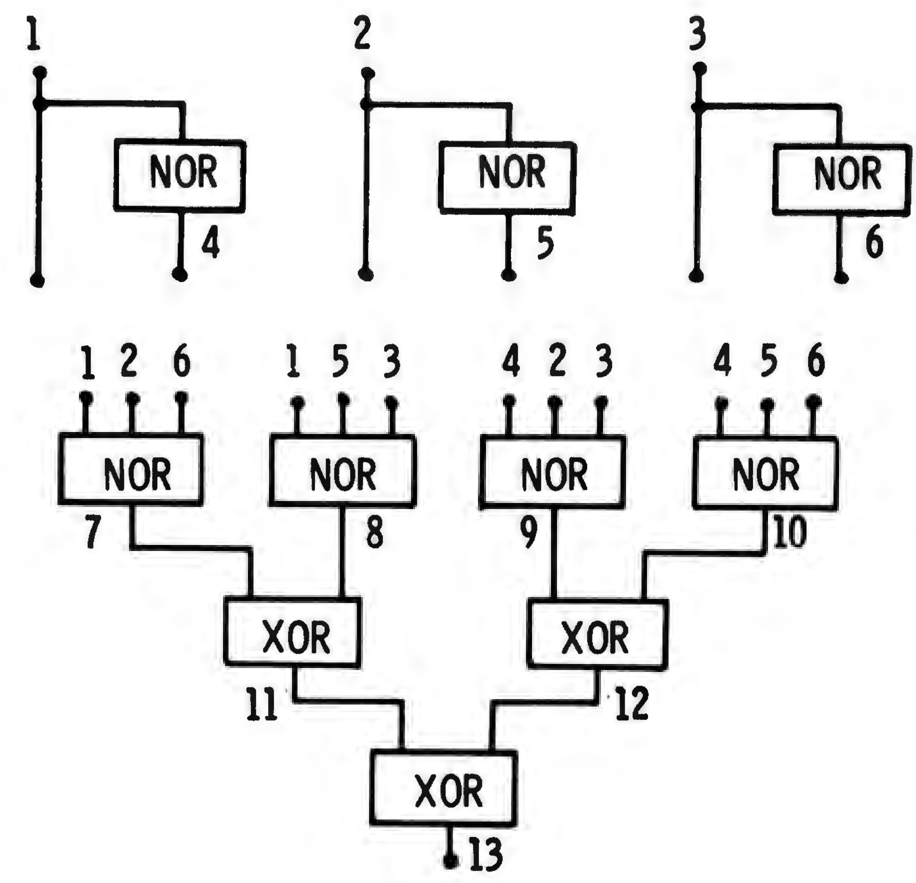
1) Detection of the malfunction and preventing the propagation of erroneous information.
2) Assessment of the damage to critical hardware (diagnosis).
3) Replacement or self-repair of critical hardware, if necessary.
4) Assessment of the damage to critical information (i.e., programs and data).
5) Reconstruction of critical information, if necessary.
6) Restarting critical tasks.

The ARC general organization was chosen to facilitate carrying out these tasks. The first problem to be faced is that of detection of a malfunction.

5.5.1.2 Error Detection Parity

It is "well known" that parity check circuits will "detect the occurrence of a single error" and will "not detect the occurrence of a double error"[Peterson, 61]. What does this mean in terms of computer component malfunctions? Will a parity check detect a malfunctioning component? If not, what may happen when two components fail?

A natural circuit to look at is a decoder, with n inputs and 2^n outputs, only one of which should be one at a time. As shown in section 5.4.7, just adding a parity check across the outputs will not check the circuit - specifically the input. The circuit shown in Figure 5.36b (reproduced here as Figure 5.46) checks all elements except the three inverters stuck at zero. If one of these fails the circuit still operates correctly, and a single test will check for all errors. These properties begin to look foolproof, until the effect of double errors is shown. In Figure 5.46 a correct set of inputs with the accompanying Test Detect pattern is shown in the top section of the table. All elements except the inverters 4, 5 and 6 stuck at zero are tested. Since the circuit is symmetric, assume here that 4 is stuck at zero. Use line 4 as an input, and run Test Detect again obtaining the results in the lower table. Now line 1 is not tested for stuck at zero. In the lower part of the table it is seen that the erroneous input combinations 0, 0, 0 and 0, 1, 1 will not produce an error signal. Thus, during the interval between testing, half the erroneous patterns will not be caught. If the error signal fails the same statement can be made. This information can be contaminated without error signal. Clearly it is not enough to blindly insert parity tests and declare "the machine will catch single errors".



1	2	3	1	2	3	4	5	6	7	8	9	0	1	2	3
0	0	1	1	1	0			1	0	1	1	1	0	1	0
0	1	0	1	0	1		1		1	0	1	1	0	1	0
1	0	0	0	1	1	1			1	1	0	1	1	0	0
1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	0
0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1

1	2	3	4	1	2	3	4	5	6	7	8	9	0	1	2	3
0	0	1	0	1				0	1	0	1	1	1	0	1	0
0	1	0	0	1				1	0	1	0	1	1	0	1	0
1	0	0	0		1	1	1			1	1	0	1	1	0	0
1	1	1	0		0	0	1	1	1	1	1	1	0	1	0	0
0	0	0	0				1	0	0	1	1	0	1	1	0	0
0	1	1	0				1	1	1	1	1	1	0	1	0	0
1	0	1	0	0	1	0		0		1	1	1	1	1	1	1
1	1	0	0	0	0	1			0	1	1	1	1	1	1	1

Fig. 5.46 Effect of double errors.

The circuit shown in Figure 5.37 will signal an error if any element fails, except one of the checking signal output. If line 16 is stuck at 1, then this circuit has all the properties of the former circuit without error. If line 13 is stuck at 1, then only inverter failures are checked. Information can still be contaminated although with more difficulty. It is clear that the checkers must be checked periodically (with diagnostic patterns) and that contaminated information must be stored for roll-back procedures. The frequency of testing will be a function of the probability of circuit failure, the number of circuits which can fail without error indication, and the procedures devised to indicate externally induced failures (reasonableness checks, etc.) which are application dependent.

The deleterious effects of topology changes were shown in section 5.4.7, Figure 5.42. Because of this, and the preceding discussion, it is clear that to design an ARC which uses replacement repair, a procedure like the one outlined in Figure 5.47 must be used.

The fact that decoders can be designed with 100% dynamic decoder error detection and complete error detection in the checkers through diagnosis (and with a simple test) shows that the procedures in Figure 5.47 are feasible. The use of on-line computing facilities and TEST DETECT are essential. The error coverage must be determined - and the percentage of error signals produced by single failures determined.

5.5.1.3 <u>Sentinel Turn-On Procedures</u>

As planned, the ARC will not operate continuously; however, the clock and a small amount of hardware must be operating so that the ARC can be ready for correct operation at

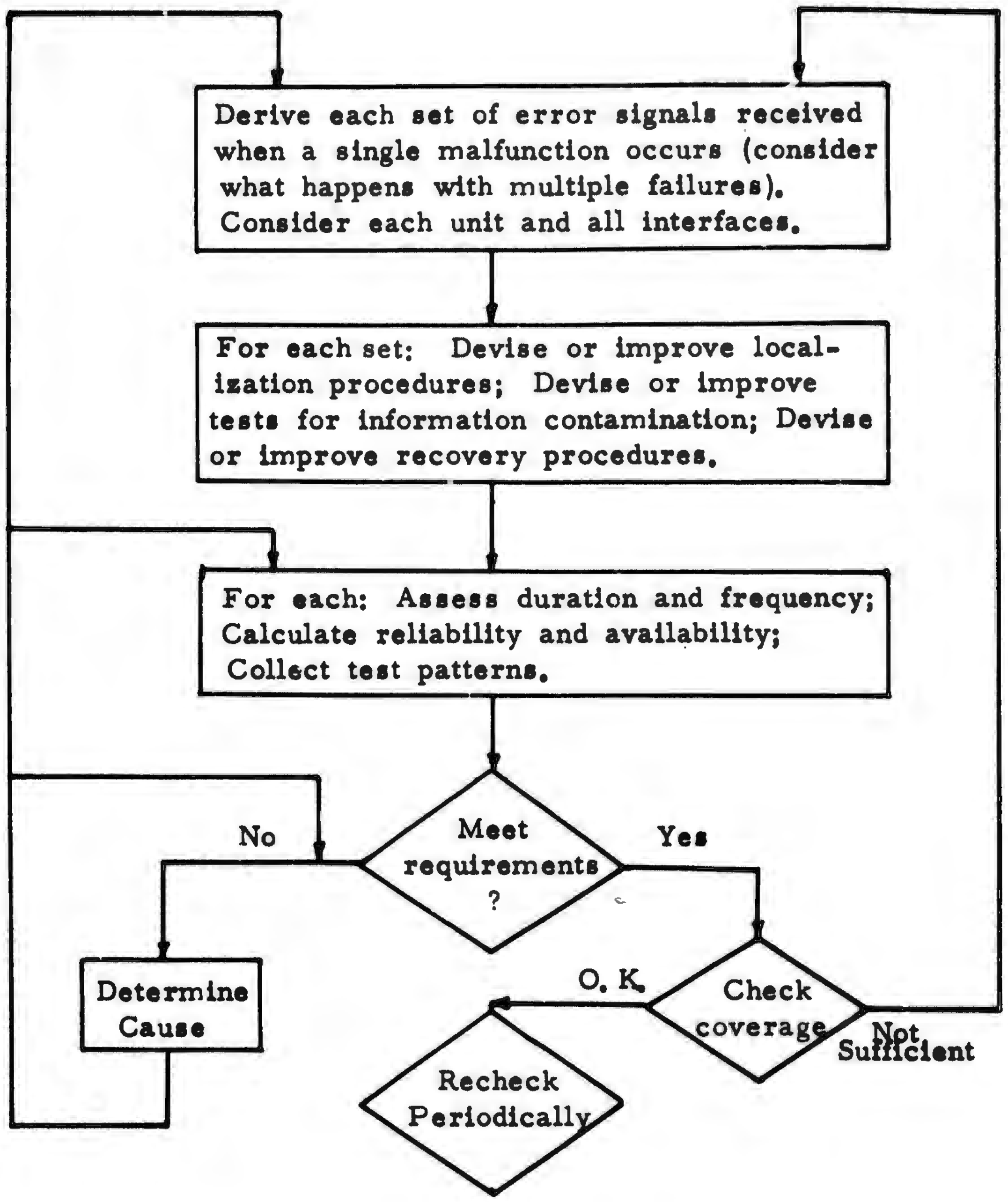

Derive each set of error signals received when a single malfunction occurs (consider what happens with multiple failures). Consider each unit and all interfaces.
For each set: Devise or improve localization procedures; Devise or improve tests for information contamination; Devise or improve recovery procedures.
For each: Assess duration and frequency; Calculate reliability and availability; Collect test patterns.
Meet requirements ?
No
Yes
Determine Cause
Check coverage
O. K.
Not Sufficient
Recheck Periodically

Fig. 5.47

predetermined times or after the receipt of external signals. The procedures to be followed are similar to those discussed in section 5.5.2, Diagnosis.

5.5.2 Diagnosis

5.5.2.1 General Statements

After an error signal has been initiated, electrical isolation must be initiated to prevent propagation of errors in information - either data or machine configuration. This must be planned using general strategies such as shown in Figure 5.47. Electrical isolation is possible [DS, 66] but is specific design dependent. Dividing the computer into independent sections, as in ARC, helps, as does the reconfiguration switches. Now, information about the computer state must be collected. This is called 'log-out' in System/360. Next, the location of the malfunctioning component must be determined. Ideally, in the ARC design, failures in the interface switches will look like module failures. So the first step is to determine the unit which contains the malfunctioning element. Experience with the FAA 9020 [Wood, 65] has shown that signals to aid in analysis for error location must be designed from the first design specification stages.

Planning diagnosis is to a large extent dependent upon detailed engineering and application specification. However, a few general statements can be made.

a) If a procedure such as that outlined in Figure 5.47 is used, circuitry can be built so that the existance of an error signal will initiate a micro-instruction retry as described in section 5.5.1.3.

b) Following similar procedures, circuitry can be designed so that an instruction retry can be initiated, as discussed in section 5.5.1.3.

c) If a micro-instruction or instruction retry is not indicated by hardware, programs have been devised to analyze the error indications (which are stored) and determine if single is possible or if rollback procedures are recessed[Wood', 65].

d) If a rollback procedure must be initiated, then a connect hardware computer configuration must be determined. The problems of information contamination and recovery will be discussed later, in section 5.5.4.

5.5.2.2 Diagnosis Ab Initio

To determine a correct hardware computer configuration, the status information in the status registers must be specified, and a set of correctly functioning modules determined so that computer functions can be correctly contained. These basic procedures must be stored in the ROS. The ROS replication insures that they are stored at least twice and will several errors. If more detailed analysis is necessary, it must wait careful identification of the procedures.

The ARC begins by bootstrapping its way into operation. A procedure to be followed has been described by Forbes, et. al [AFS, 67] As stated in [AFS, 67], there must be at least two diagnostic subsystems, and each must perform the functions:

F1. Supply the linkage to enable expectation of a given sequence consisting of the operations F2, F3, F4, F5 appearing in any order.

F2. Transmit predetermined stimuli to all, or a selected portion, of the inputs of the subsystem S.

F3. Obtain the outputs of the subsystem S and compare them with given patterns.

F4. Proceed to the next operation in the sequence or branch to a different point in the sequence of operations, depending on the result of the last F3 operation.

F5. Employ an alternate diagnostic subsystem. If results agree and no error is signaled, reconfigure ARC or state status is correct as found. If results disagree, use the signal without error indication. If all results have error indications, signal the external world and try again.

The control to carry out F1 through F5 is simple enough. The real problem is obtaining and storing the predetermined stimuli - or diagnostic test patterns.

The previous design work, using TEST DETECT and occasionally DALG-II, will have provided a set of diagnostic test patterns. The main problem is storage.

One step to reduce storage is to test the parity checking. circuits, then use them to signal the existance of malfunctioning elements. In section 5.4.8 it is shown that test patterns could be simply generated to test 9 bit parity check circuits. In section 5.4.7 and this section the design of circuits so that malfunctioning components could be detected was discussed. Note that for successful testing the expected output signal must be stored and compared with the output received. The fewer status registers to be examined and the fewer tests to be stored the better. If tests are to be read from external storage, then all paths leading to the storage must be checked before the tests are read. Then the tests must be verified (say, with two-dimensional parity coding) before they are used.

If the diagnostic tests have been employed on a regular basis, diagnostic analysis is aided. The best way is to employ the diagnostic tests a portion at a time. The basic idea is that not all of the computer need be diagnosed at the same time. We have termed this technique "interleaving" because the diagnostic procedure is interleaved with the operational program. Several mechanisms suggest themselves as possible ways of interrupting the operational program in order to do some diagnosis. One method depends on the clock; after a predetermined span of time, a diagnostic interrupt occurs. Another method is to have an instruction counter which causes a diagnostic interrupt after a predetermined number of instructions have been executed. Since the switching repair action can take place in a matter of micro seconds, it behooves us to speed up the diagnosis in order to know precisely what self-repair action to take. Interleaving the application of various portions of a complete set of diagnostic tests not only serves to monitor the computer and validate its correct performance but also enables the diagnosis to concentrate on those computer elements that have been suffering intermittent errors. This will speed up the diagnosis if the speculation is correct that many solid type failures start as transient ones. This does not mean that anything so complex as recording the history of errors is involved, but rather that a unit status register for each unit must be provided for and employed in the manner described next.

The distributed functions which are joined together to form the ARC are each self checking. The assessment of malfunctioning hardware is made easy by a consideration of the history of past successful dynamic checking and diagnosis.

Building a few types of units, each of which has some parallel operation (multiples of 9 bits) appears to simplify the design and checking, aid in the diagnosis (n identical relatively small objects), simplify the micro programming problems of control, and allow speed by parallel operation. Using the diagnostic aids, the efficiency of checking can be determined. Good diagnostics to determine the nature and location of malfunctions can be stored in the control memory, and use data from the bulk memory. The diagnostic aids will allow an exhaustive testing of each distributed function unit. The next step is to use these algorithms to test the interconnections (which should increase less rapidly than the combinatorial number of all possible inputs). By putting more and more function into distributed hardware control, but leaving flexible interconnections, with all interfaces defined accurately, the individual interconnecting steps are simple, and can be diagnosed by our algorithms.

Dividing the system into distributed functions aids in the assessment of the damage to critical information by localizing the effects of the error. Rigidly defined interfaces and procedures aid in this.

One important hardware control feature is that during tests all checking circuits should have time to react before the next test begins.

5.5.3 Retry

If a failure is intermittant, then a dynamic form of error detection, like parity checking with carefully designed circuits, is necessary to detect it. As stated in section 2, the ratio of intermittant to solid failures is measured to be at least 5 to 1. While detection is a problem, correction is not. If the error can be localized, one form of correction of the induced errors is to do the process a second time and see if the failure has gone away. Any successful retry operation must have uncontaminated initial data and control information. For instruction retry much information which is discarded by conventional computer organizations must be saved. If this is done, however, computer control is relatively unimportant since most operations essentially start from scratch. For micro instruction retry, less information must be saved, since the operational time is less. However, the detection of an error must be rapid, and control information which guides the next micro instruction step must be preserved (e.g., the micro instruction steps in a multiplication are interrelated adds and shifts). Selection of the proper mix of such retrys depends upon the computer organization, since no single method is universally best.

Operation retry is one means of trading extra speed for higher reliability. Since the present speed requirements for space-borne computers is not excessive, this exchange is a worthwhile design objective.

Reconstruction of critical information is also eased by the distributed function and rigidly defined interfaces. Thus reconstruction is simplified. The amount of information to be stored for restarting is easily specified and may be stored asynchronously. Reinitialization will follow rigid rules. A detailed discussion can only follow a specification of the computer command and micro command codes - and this specification will be influenced by the retry necessity.

5.5.4 Reconfiguration

The hardware and procedures for individual switchable interfaces were discussed in section 5.4. After diagnosis (see section 5.5.2) has determined the desired correct state of the ARC, then control stored in the ROS must be used to carry out the necessary steps. While these steps are very design dependent, it is clear from section 5.4 that much information will have to be transferred unless status registers are distributed. If there are shift registers, then hardware is increased but information transfer is reduced.

In any case, after reconfiguration verification of correct operations must be performed. For specific applications there may be considerable time pressure if all these operations are carried out serially. The ARC must have circuits and units which are designed for complete dynamic checking, fast diagnosis, rapid repair, and efficient verification of correct operation.

5.5.5 Information Recovery

Recovery is defined as the continuation of system functions after an error. From a total systems point of view, the levels of continuation may be defined as follows:

Unchanged:	Continues at same level of time and space after error handling.
Restart:	Continues at the same level of time and space but at a different program point after error handling.

Degraded:	Continues at lower level of time and/or space and/or different program point after error handling.
Terminated:	Ended with notification and attempt at information retrieval.

Clearly, continuation (in general) implies certain requirements. The effects of incomplete jobs must be removed from the system. Jobs that cannot be restarted must be analyzed. To perform these tasks irvolves the reconstruction of secondary storage as well as a re-analysis and resubmission of the job.

After an error has occurred, a 'damage report' must be available which will show first, los of function and, secondly, extent of contamination. Loss of function has been well treated from the hardware design view, and is eased with the UAC organization.

Information contamination is another problem. The operating environment is asynchronous. How immediate is 'immediate detection of errors' and how fast must it be to prevent error propagation and global contamination? What about a programming or procedural error and its effect on secondary storage? The proposed distribution of function in the UAC design aids greatly in preventing the spread of contamination of information, particularly if the error detection procedures recommended by Davis and Stafford are followed [DS, 66].

Recovery implies the existence of state information. For an analysis of system recovery, it is necessary to return to the concept of system state vector. Consider the information which should be available to make system restart possible.

Level 1. Registers accessible to the user, program status information (PSW), etc. both old and new...

Level 2. Program Control Blocks - for example:

a) Job Management

b) Task Information

c) Data Management

d) Physical Channel and Unit Management (Queues and Tables).

Level 3. Input Journal

a) This journal of system input activity includes all data which enters the system via the job stream and from the terminals. (Input to main storage from secondary storage is not a system input.)

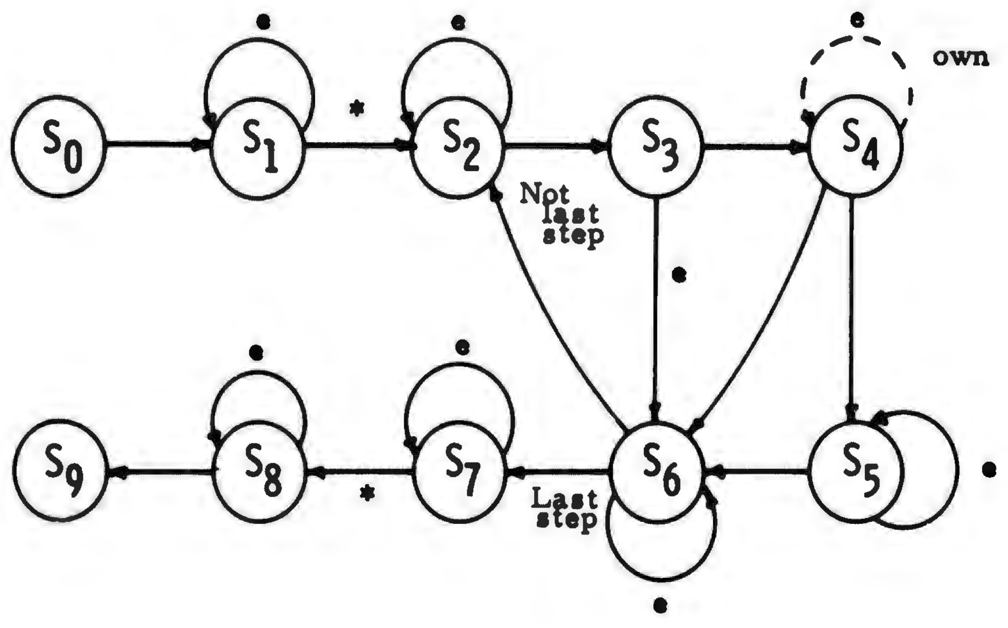
b) Secondary Storage Activity Journal. This records all newly entered data sets, changes to existing data sets, and deleted data-sets.

In terms of Levels 1-3, restart capability requires state information for CPU, core, and secondary storage. Further, total state information cannot be provided by a user checkpoint (as normally understood) alone.

Assuming the availability of the above information, job states within the system state vector can be considered. For example:

Job States	Description
S0	Waiting in Input Job Stream
S1	Read in, not queued
S2	Enqueued, not allocated
S3	Enqueued and allocated
S4	In Execution
S5	Step terminated
S6	Unallocated
S7	System Output enqueued
S8	System Output active
S9	Job complete

Fig. 5.48 shows the paths that control should follow in progressing through these states. The lines labeled 'e' are followed in case of error, the others, if there is no error.



S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
e
e
e
own
*
Not last step
e
e
e
e
e
*
Last step
e

Fig. 5.48

The action to be taken at each step clearly influences the information saved. It should be noted that if S4 is long, it needs its own initiation of checkpoint(s), which must be supported by system state information. Much analysis has been spent on this particular case. For many cases, a set of simple and satisfactory restart points are those starred before S2 and S8. Much of the information on the action to be taken depends on the operating environment and will be discussed in the next section. Before turning to this question, some general remarks are appropriate concerning the time correlation of state information; data integrity, data set redundancy and data set dependencies; and some basic steps required for recovery.

The items in the Journals must be time stamped to permit correlation of the secondary storage rollback. Journal contents are related to checkpoints.

For data integrity the following rules are necessary:

1. All critical data sets are Read Only from back up between checkpoints.
2. A data set is updatable in back up at checkpoint only.
3. Checkpoint is on a system basis.

In general, there must be two copies maintained of all vital system information (e.g., Job Queue, Task Queue with dependencies, Secondary Storage Activity Journal...). Keeping system status implies a real need for duplication on the fly - or in a fail safe way - in order to keep the time of non-reconstructability short.

Rules 1 and 2 above show that the record hold and interlock problems apply to secondary storage. The ARC design allows the CCU to solve these problems with its own control.

The effects of data contamination are a direct function of dependencies between data sets. There are three basic types of dependency:

a) Ancestry - one taks generates another, or one data set is used to generate other(s).

b) Common - information or tasks used together, explicitly or implicitly stated.

c) Common physical resources.

Such dependency relations show that contamination can spread rapidly. However, the relatively independent CCU can stop most of it, using its detailed local control and checking. The alternate paths will aid in recovery information.

The basic recovery steps are:

1. Purge and unallocate volumes.
2. Restore secondary storage. If impossible, collect information to aid in the information retrieval problem.
3. Refresh job queue.
4. Refresh control.
5. Restart using specified initial conditions (not reset conditions).

In state S4, in Figure 5.48 the time between checkpoints must be a USER input. The user must specify his rollout points. However, the system must use them with discretion to protect the user against gliches occurring during the system operation of his program - errors in the supervisor, his own program, or in other multiprogrammed tasks. This makes for an interesting accounting problem. Other user input will be discussed in the environment section.

It is clear that the ability to restart implies:

a) System status information in concise form and readily available.

b) The ability to do a "warm start" - i.e., to begin at arbitrary points with non-empty queues, etc.

c) For efficiency, the functions of the CCU should be included (needed for ultra availability anyway).

The type of system environment most frequently met in spaceborne computers is that of a dedicated system with cyclic input. In these systems there is one main job, repeated frequently. The input arrives in quantity and disregardirg the input for a short period of time does no harm. Processing can begin without reconstruction. Examples of this type of system are the FAA, orbit tracking, and many types of industrial control. These systems have high vulnerability (they can't go back in time), low file orientation and usually require a fast response time.

Recovery Procedures:

Requirements:

a) Back up of critical files.

Algorithm:

a) Run the program to access damage report, reconfigure system and report results.

b) Reconstruct core contents, bring in back up program.

c) Start with new cycle of data.

The length of time available for recovery must be specified by the user. If it is short (e. g., FAA) then problems may arise, but they have been solved in all cases to date.

Step a) in the algorithm is by far the most difficult, and is still application dependent, except for hardware retry.

REFERENCES

1872

[Boole, 72]. Boole, G. Calculus of Finite Differences, Chelsea Publishing Company, 1872.

1932

[Khirtchine, 32]. Khirtchine, A. Y., Mathematisches über die Erwartung von einen öffentliches Schatter, 1932.

1940

[V.D. Waerden, 40]. Van der Waerden, B.L. Modern Algebra, Volume 2, Frederick Ungas Publishing Company, 1940.

1941

[Feller, 41]. Feller, W., On the Integral Equation of Renewel Theory, Am. Math. Statist., Volume 12, pp. 243-267, 1941.

1949

[Harvard, 49]. Proceedings of a Second Symposium on Large Scale Digital Calculating Machinery, Annals of the Computation Laboratory, Harvard University, Volume XXVI, 1949.

1950

ERA, 50]. High Speed Computing Devices, ERA, McGraw-Hill, 1950.

1952

[Davis, 52]. Davis, D. J., An Analysis of Some Failure Data, Journal of American Statist. Association, Volume 47, No. 258, pp. 113-150, 1952.

1953

[Eachus, 53]. Eachus, J. J., Group Discussion on Diagnostic Checks, EJCC, 1953.

[ES, 53]. Epstein, B. and Sobol, M., Life Testing, Journal of American Statist. Association, Volume 48, No., 263, pp. 486-502, 1953.

[SYMP, 53]. Symposium, Diagnostic Programs and Marginal Checking for Large Scale Digital Computers, Convention Record of IRE, National Convention, 1953.

[WR, 53]. Wheeler, D. J., and Robertson, J. P., Diagnostic Programs for the Illiac, Proc. of IRE, October 1953.

1954

[Huffman, 54]. Huffman, D.A., The Synthesis of Sequential Switching Circuits, J. Franklin Institute, pp. 161-190, 275-302, March - April 1954.

[Felker, 54] Felker, J. H., Performance of TRADIC Transistor Digital Computer, EJCC, 1954.

1955

[CL, 55]. Coddington, E. A. and Levinson, N., Theory of Ordinary Differential Equations, McGraw Hill, 1955.

[Mealy, 55]. Mealy, G.H. A Method for Synthesizing Sequential Circuits, Bell System Technical Journal, Volume 34, No. 5, pp. 1045-1080, September 1955.

[Roth, 55]. Roth, J. P., Algebraic Topological Methods for the Synthesis of Switching Systems, I, Transactions of the American Mathematical Society 88, 2, 301-326, July 1958. Cf. also Institute for Advanced Study, Princeton, New Jersey., ECP 56-02, April 1956 and General Electric Company Report No. R55GL345, Schenectady, New York, September 1955.

[Weik, 55]. Weik, M.H. A Survey of Domestic Electronic Digital Computing Systems, Ballistic Research Laboratories, Report No. 971, Aberdeen Proving Grounds, Maryland, December 1955.

1956

[Creveling, 56]. Creveling, C. J., Increasing the Reliability of Electronic Equipment by the Use of Redundant Circuits, Proc. IRE, Volume 44, pp. 503-505, April 1956.

[Moore, 56]. Moore, E. F., Gedanken - Experiments on Sequential Machines, Automata Studies, No. 34, pp. 129-156, Princeton, New Jersey, 1956.

[MS, 56]. Moore, E. F., and Shannon, C. E., Reliable Circuits Using Less Reliable Relays, Journal of the Franklin Institute, Volume 26, pp. 191-208, September 1956, pp. 281-297, October 1956.

[von Neumann, 56]. von Neumann, J., Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components, Automata Studies, Annals of Mathematics, Study No. 34, Princeton, pp. 43-98, 1956.

1957

[EZB, 57]. Everett, R. P., Zraket, C. A., and Bennington, H. D., SAGE - A Data Processing System for Air Defense, EJCC, p. 148, 1957.

[Fein, 57]. Fein, L., The Place of Self-Repairing Facilities in Computers with Deadlines to Meet, FJCC, Washington, D. C., 1957.

[Feller, 57]. Feller, W. S., An Introduction to Probability Theory and Its Application, Wiley and Sons, 1957.

[SH, 57]. Schneider, S., and Wager, D. H., Error Detection in Redundant Systems, WJCC, 1957.

[Rosenblatt, 57]. Rosenblatt, J. R., On Prediction of System Performance from Information on Component Performance, Proc. WSCC, 1957.

[Weik, 57]. Weik, M. H., A Second Survey of Domestic Electronic Digital Computing Systems, Ballistic Research Laboratories, Report 1010, Aberdeen Proving Grounds, Maryland, June 1957.

1957 (cont'd)

[WJCC, 57]. 1957 Western Joint Computer Conference Proceedings. Boldyriff, A. W., Reliability from a System Point of View; Wave, W. W., Reliability and the Computer; Glantz, H. T Reliability in Barriers Systems; Smith, B. K., The Interpretation and Attainment of Reliability in Industrial Data Systems.

[Zoger, 57]. Zoger, H. F., Evolution of Failure Data, Proc. WSCC, 1957.

1958

[Flehinger, 58]. Flehinger, B. J. Reliability Improvement through Redundancy at Various System Levels, IBM Journal of R & D, April 1958.

[RR, 58]. Reid, L. W., and Raymond, G. A., The Athena Computer, a Reliability Report, EJCC, 1958.

1959

[Eldred, 59]. Eldred, R. D., Test Routines Based in Symbolic Logic Statements, J. ACM, Volume 6, No. 1, pp. 33-36, January 1959.

[Roth, 59]. Roth, J. P., Algebraic Topological Methods in Synthesis in Proceedings of an International Symposium on the Theory of Switching, 2-5 April 1957, Part 1: The Annals of the Computation Laboratory of Harvard University, Volume XXIX, pp. 57-73, Harvard University Press, Cambridge, Mass. 1959.

1960

[Brown, 60]. Brown, D. T., Error Detecting and Correcting Binary Codes for Arithmetic Operations, IEEE TEC, Volume 9, pp. 333-337, 1960.

[GMRW, 60]. Griesmer, J. H., Miller, R. E., Roth, J. P., and Wagner, E. G., Design of a Self-Repairing Cryogenic Machine, IBM Watson Research Report RC-204, 1960.

[Howard, 60]. Howard, R. A., Dynamic Programming and Markov Processes, Wiley and Sons, 1960.

[Roth, 60]. Roth, J. P., Minimization over Boolean Trees, Reprinted from IBM Journal of Research and Development, Volume 4, No. 5., November 1960.

1961

[Calingaert, 61]. Calingaert, Peter. Two-Dimensional Parity Checking, J. ACM, April 1961.

[GNR, 61]. Galey, J. M., Norby, R. E., and Roth, J. P., Techniques for the Diagnosis of Switching Circuit Failures, IBM Corporation, Poughkeepsie, New York, 1961.

[Peterson, 61]. Peterson, W. W., Error-Connecting Codes, M. I. T. Press, 1961.

[PP, 61]. Perry, M. H., and Plugge, W. P., American Airlines SABRE Electronics Reservations System, WJCC, 1961.

[Roth, 61]. Roth, J. P., Lectures on the Design of Automata, Lecture Notes of the Special Summer Session on Logic, Switching Systems and Automata, Moore School of Electrical Engineering, University of Pennsylvania, 1961.

[Rutledge, 61]. Rutledge, J. D., Self-Replacement Circuit for Crygenic Systems, IBM Technical Disclosure Bulletin, Volume 3, pp. 38-39, May 1961.

[Weik, 61]. Weik, M. H., A Third Survey of Domestic Electronic Digital Computing Systems, Ballistic Research Laboratories. Report No. 1115, Aberdeen Proving Grounds, Maryland, March 1961.

1962

[Buchholz, 62]. Buchholz, W., editor, Planning a Computer System, Project STRETCH, McGraw-Hill, 1963.

[Carter, 62]. Carter, W. C., Mathematical Analysis of Merge-Sorting Techniques, Proceedings of IFIP Congress, North Holland Publishing Company, 1962.

[GMR, 62]. J. H. Griesmer, Miller, R. E., Roth, J. P., The Design of Digital Circuits to Eliminate Catastrophic Failures, Redundancy Techniques for Computing Systems, Spartan Books, 1962.

[Kautz, 62]. Kautz, W. H., Codes and Coding Circuitry for Automatic Error Correction Within Digital Systems, Redundancy Techniques for Computing Systems, Spartan Books, 1962.

1962 (cont'd)

[Kemp, 62]. Kemp, J. C., Redundant Digital Systems, Redundancy Techniques for Computing Systems, Spartan Books, 1962.

[Mann, 62]. Mann, W. C., Restorative Processes for Redundant Computing Systems, Redundancy Techniques for Computing Systems, Spartan Books, 1962.

[RK, 62]. Roth, J. P., Karp, R. M., Minimization over Boolean Graphs, IBM Journal 6, 2, 1962.

[SAPP, 62]. Serrell, R., Astraham, M. M., Patterson, G. W., and Pyne, I. B., The Evolution of Computing Machines and Systems, IRE, 50 p. 1051, May 1962.

[Toeste, 62]. Toeste, R., Design of a Repairable Redundant Computer, IRE TEC, October 1962.

[Tryon, 62]. Tryon, J. G., Quadded Logic, Redundancy Techniques for Computing Systems, Spartan Books, 1962.

1963

[Dumey, 63]. Dumey, A. I., A Note on Stochastic Matrices, Comm. of the ACM, September 1963.

[MA, 63]. Maling, K., and Allen, E. L. Jr., A Computer Organization and Programming System for Automated Maintenance, IEEE TEC, Volume 12, No. 6, pp. 887-895, 1963.

[Poage, 63]. Poage, J. F., Derivation of Optimal Tests to Detect Faults in Combinational Circuits, Mathematical Theory of Automata, Polytechnic Press, Brooklyn, New York, pp. 433-523, 1963.

[Roth, 63]. Roth, J. P., Algorithms for Computing Test Response, for each Failure of a Circuit, IBM Data Systems Div., Poughkeepsie, New York, 1963. Dept. B49, Bldg. 991, June 19, 1963, Appendix A: A Proposal for the Insertion of Test Points.

[SCF, 63]. Saturn Computer Staff, Saturn V Guidance Computer Technical Proposal, IBM Space Guidance Center, Owego, New York, prepared for the George C. Marshall Space Flight Center, Huntsville, Alabama, IBM No. 63-928-145, November 1963.

1964

[CMPR, 64]. Carter, W. C., Montgomery, H. C., Preiss, R. J., and Reinheimer, H. J., Design of Serviceability Features for the IBM System/360, IBM Journal, Volume 8, No. 2., 1964.

[ESS, 64]. No. 1 ESS Issues, Bell System Technical Journal, volume 43, No. 5, Parts 1 and 2, pp. 1831-2610, Sept. 1964.

[FIS, 64]. Falkoff, A. D., Iverson, K. E., and Sussenguth, E. H., A Formal Description of System 360, IBM Systems Journal, Volume 3, No. 3, pp. 193-262, 1964.

[GNR, 64]. Galey, J. M., Norby, R. E., and Roth, J. P., Techniques for the Diagnosis of Switching Circuit Failures, Transactions of the IEEE Communications and Electronics 83, 74, pp. 509-514, September 1964.

[Hennie, 64]. Hennie, F. C., Fault-Detecting Experiments for Sequential Circuits, Proc. Fifth Annual Symposium on Switching Theory and Logical Design, pp. 95-111, October 1964.

[Lewis, 64]. Lewis, P. A., A Branching Poisson Process Model for the Analysis of Computer Failure Patterns, Journal of the Royal Statistical Society, Series B., Volume 26, December 1964.

[PM, 64]. Poage, J. F., and McCluskey, E. J. Jr., Derivation of Optimum Test Sequences for Sequential Machines, Proc. of Fifth Annual Symposium on Switching Theory and Logical Design, pp. 121-132, October 1964.

[Porter, 64]. Porter, D. C., Failure Analysis of Electronic Parts, IEEE Trans. on Aerospace, Volume 2, No. 2, April 1964.

[SBBH, 64]. Sellers, F. F., Bearnson, L. W., Brown, D. T., and Hsiao, M. Y., Handbook of Error-Detecting Logic Circuits, IBM Report TROO. 1228, Poughkeepsie, New York December 1964.

[Urbano, 64]. Urbano, R. H., On the Convergence and Ultimate Reliability of Iterated Neural Nets, IEEE Trans. on Electronic Computers, Volume EC-13, No. 3, June 1964.

[Weik, 64]. Weik, M. H. A Fourth Survey of Domestic Electronic Digital Computing Systems, Ballistic Research Labs., Report No. 1227, Aberdeen Proving Grounds, Maryland, January 1964.

1965

[AES-EPO, 65]. AES-EPO Study Program - Final Study Report, IBM No. 65-562-012, Owego, New York, Dec. 1965.

[ARSY, 65]. Agnew, P. W., Rutherford, D. H., Suhocki, R. J., and Yen, C. M., An Architectural Study for a Self-Repairing Computer, Final Technical Documentary Report No. D-TR-65-159, U. S. Air Force, Space Systems Division, Los Angeles, California, (AD4-749-976)(IBM No. 65-928-91), November. 1965.

[BP, 65]. Barlow, R. E., and Proschos, F., Mathematical Theory of Reliability, John Wiley and Sons, 1965.

[DK, 65]. Dunning, M., and Kolman, B., Reliability and Fault-Masking in n-variable NOR Trees, 1965 IEEE Conference Record on Switching Circuit Theory and Logical Design, 1965.

[Dorrough, 65]. Dorrough, D. C., Redundancy and System Reliability, Fall SIAM Meeting, 1965, to appear in TECHOMETRICS.

[FRST, 65]. Forbes, R. E., Rutherford, D. H., Stieglitz, C. B., and Tung, L. H., A Self-Diagnosable Computer, AFIPS Conference Proceedings, Volume 27, part 1, 1965.

[Gluck, 65]. Gluck, Simon, Impact of Scratchpads in Design: Multifunctional Scratchpad Memories in the Burroughs B8500, Proc. AFIPS, FJCC, 1965.

[Gurzi, 65]. Gurzi, K. J., Estimates for Best Placement of Voters in a Triplicated Logic Network, IEEE TEC, Volume 14, No. 4., pp. 711-717, October 1965.

[HS, 65]. Hackl, F. J., and Shirk, R. W., An Integrated Approach to Automated Computer Maintenance, 1965 IEEE Conference Record on Switching Circuit Theory and Logic Design, pp. 289-302, October 1965.

[McCarthy, 65]. McCarthy, J., Problems in the Theory of Computation, Proc. IFIPS, Congress, May 1965.

1965 (cont'd)

[Nerber, 65]. Nerber, P. O., Power-Off Time Impact on Reliability Estimates, IEEE International Convention Record, Part 10, pp. 1-15, March 22-26, 1965, New York.

[Pierce, 65]. Pierce, W. H., Failure Tolerant Computer Design, Academic Press, 1965.

[Quatse, 65]. Quatse, J. T., Strobes Shared-Time Repair of Big Electronic Systems, AFIPS Conference Proc., Volume 27 Part 1, 1965.

[Raymond, 65]. Raymond, G. A., Second Thoughts on Reliability, IEEE Trans. on Reliability, March 1965.

[Saturn, 65]. Saturn Computer Staff, Reliability Assessment Report for Saturn V Guidance Computer and Data Adapter, IBM No. 65-394-036, Owego, New York, July 1965.

[TM, 65]. Terris, I., and McKannoff, M. A., Investigation and Simulation of a Self-Repairing Digital Computer, IEEE Conference on Switching Circuit Theory and Logic Design, 1965.

[U. S., 65]. U. S. Naval System Effectiveness Control Manual, U. S. Naval Applied Science Lab., Brooklyn, New York, July 1965.

[Wood, 65]. Wood, J. R., The 9020: A Modular Multiprocessing System for NAS, IBM No. TR21.187, Kingston, New York, November 1965.

1966

[AHT, 66]. Alonso, R. A., Hopkins, A. L., and Thuler, Design Criteria for a Spacecraft Computer, Spaceborne Multiprocessing Seminar, Boston, Massachusetts, October 1966.

[Armstrong, 66]. D. B. Armstrong, On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets, IEEE TEC, Volume EC15, No. 1, pp. 66-73, February 1966.

[Avizienis, 66]. Avizienis, A., System Organization of the JPL Self-Testing and Repairing Computer and its Extension to a Multiprocessor Configuration; Spaceborne Multiprocessing Seminar, Boston, Massachusetts, October 1966.

[BE, 66]. Burnstine, D. C., and Eppard, W. H., Maintenance Strategy Diagramming Technique, 1966 Annual Symposium on Reliability.

[BH, 66]. Ball, M., and Hardie, F. H., Architecture for an Extended Mission Aerospace Computer, IBM No. 66-825-1753, Owego, New York, May 1966.

[BW, 66]. Burke, T. E., and Yang, C. Y., Spaceborne Multiprocessing Organizations, WESCON, August 1966.

[DS, 66]. Davis, C. M., and Stafford, T. S., System Design for High Availability, IBM TR 00.1546, October 24, 1966.

[FI, 66]. Falkoff, A. D., and Iverson, K. E., The APL Terminal System: Instructions for Operation, IBM Watson Research Center, Yorktown Heights, New York 10593, March 1966 (available upon request).

[Garner, 66]. Garner, H. C., Arithmetic Error Correction, Space-borne Multiprocessing Seminar, Boston, Massachusetts, October 1966.

[Goldberg, 66]. Goldberg, J., Logic Design Techniques for Error Control, WESCON, August 1966.

[Gunderson, 66]. Gunderson, D. C., Associative Memories in Space Applications, WESCON, August 1966.

1966 (cont'd)

[HM, 66]. Hasett, R. P., and Miller, E. H., Multithreading Design of a Reliable Aerospace Computer, 1966 Aerospace and Electronic Systems Convention Proc., May 3-5, 1966.

[Hokom, 66]. Hokom, R., Executive Program Control for Spaceborne Multiprocessors, Spaceborne Multiprocessing Seminar, Boston, Massachusetts, October 1966.

[Joseph, 66]. Joseph, E. C., Self-Repair: Fault Detection and Automatic Reconfiguration, Seminar on Spaceborne Multiprocessing, Boston, Massachusetts, October 1966.

[Kime, 66]. Kime, C. R., A Failure Detection Method for Sequential Circuits, TR 66-13, Department of Electrical Engineering, University of Iowa, Iowa City, Iowa, January 1966.

[LC, 66]. Lovell, G., and Carney, R., The Dual Redundant Concept of Computer Operation, 1966 Aerospace Computer Symposium, Santa Monica, California, October 1966.

[Miller, 66]. Miller, E. H., Reliability Aspects of the RCA/USAF Variable Instruction Computer, 1966 Aerospace Computer Symposium, Santa Monica, California, October 1966.

[RBCF, 66]. Roth, J. P., Bouricius, W. G., Carter, W. C., and Forbes, R. E., Proposal for Phase II of an Architectural Study for a Self-Repairing Computer, IBM Watson Research Center, Yorktown Heights, New York, 10598, February 11, 1966.

[RBS, 66]. Roth, J. P., Bouricius, W. G., and Schneider, P. R., Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits, Aerospace Computer Symposium, Santa Monica, California, October 1966.

[RGMSW, 66]. Roth, J. P., Griesmer, J. H., Miller, R. E., Selfridge, J. L., Wagner, E. G., Serially Connected Inhibitor Logic Stages with Means for Bypassing a Selected Stage, Patent 3,235,842, Filed July 29, 1960, issued February 15, 1966.

[Roth, 66]. Roth, J. P., Diagnosis of Automata Failures: A Calculus and a Method, IBM Journal, Volume 10, No. 4., July 1966.

1966 (cont'd)

[Roth, 66F]. Roth, J. Paul, A Calculus for Failure Diagnosis, IBM Watson Research Center, Yorktown Heights, New York, UAC 67-2, February 1966. Invited presentation at Symposium on the Organization of Reliable Automata, Sponsored by IEEE and University of California, Los Angeles, Los Angeles, California, February 1966.

[SG, 66]. Schaeman, P. S., and Gruman, E. L., Functional Requirements of Spaceborne Computers on Advanced Manned Missions, Spaceborne Multiprocessing Seminar, Boston, Massachusetts, October 1966.

[Vandling, 66]. Vandling, G. C., C-32 Principles of Operations, IBM No. 66-533-010, Owego, New York, July 1966.

[WKB, 66]. Williman, A., Koczela, L., and Burnett, G., Multiprocessing Seminar, Boston, Massachusetts, October 1966.

[WS, 66]. Weisberg, S. A., and Schmidt, J. H., Computer Techniques for Estimating System Reliability, 1966 annual Symposium on Reliability.

1967

[AFS, 67]. Agnew, P. W., Forbes, R. E., and Stieglitz, C. B. An Approach to Self-Repairing Computers, First Annual IEEE Computer Conference, September 6-8, 1967.

[BCJ, 67]. Babb, C. D., Crenshaw, H. L., and Jones, H. L., (no title) NAECON, 1967.

[BCRS, 67]. Bouricius, W. G., Carter, W. C., Roth, J. P., and Schneider, P. R., On-line Reliability Calculations to Achieve a Balanced Design of an Automatically Repaired Computer, NAECON, May 1967.

[Feller, 67]. Feller, W., An Introduction to Probability Theory and its Application, Volume 2, Wiley and Sons, 1967.

[Schneider, 67]. Schneider, P. R., On the Necessity to Examine D-Chain in Diagnostic Test Generation - An Example, IBM Journal, Volume II, January 1967.

APPENDIX 1 - DALG II and TESTDETECT

Appendix 1.1 DALG-II

Glossary of Major Variables

L	- a vector whose I^{th} entry specifies the logic associated with line I; entries where $I \in PI$ can be any value.
PI	- a vector containing the line numbers of the primary inputs.
PO	- a vector containing the line numbers of the primary outputs.
PL, IPL	- the predecessor list (PL) vector and an index vector (IPL) on PL such that the predecessors of block I are stored in PL from $PL[IPL[I]+1]$ through $PL[IPL[I+1]]$; no predecessor entries are made for the primary input lines, i.e. for $I \in PI$.
SL, ISL	- same as PL, IPL except the successors of block I are stored.
LINES	- a vector of line numbers which are to be tested.
VALUES	- a vector with the I^{th} component giving the value the I^{th} line specified in LINES is assumed to be stuck at.
PDCFIN	- a vector giving the line numbers for the inputs in the primitive D-cube of failure (pdcf).
MPDCF	- a matrix with each row containing the actual line values for a given pdcf in the order specified by PDCFIN, PDCFO.

PDCFO	- a scalar giving the output line number for pdcf.
TC	- a vector of rank equal to the number of lines in the circuit (it corresponds to tc^k of the manual method).
A	- activity vector.
C	- the "consistency vector" containing a list of those blocks whose coordinates in TC are 1 or 0 and whose inputs cannot account for this value without making an arbitrary decision, e.g. an AND with a 0 output and all inputs x.
DB	- the block for which a pdc is being generated in DALG (during D-drive), or the one for which an arbitrary decision is being made in CDRIVE(during consistency).
INC	- a list of line numbers which have been used to increment A (during D-drive) or C (during consistency) since the last arbitrary decision.
DEC	- a list of line numbers which have been used to decrement A or C since the last arbitrary decision.
SETCO	- a list of the numbers of those lines which have been set or changed from x to 0, 1, D, or $\overline{D}$ since the last arbitrary selection of a line in A: these lines must be checked by one of the BDRIVE's.
JINT	- an index on SETCO which points to the last entry which

was examined by a BDRIVE function.

MODIN - a list of the line numbers of blocks which have had some of their inputs (predecessors) modified or set since the last application of an FDRIVE function; these lines are candidates for a forward implication.

POTEST - an indicator to signal that a primary output has been reached by the D-chain.

FAIL - an indicator which when equal to 1 denotes then an inconsistency has been encountered.

Subordinate Function Description

Since the major functions used in DALG-II (circuit, DALG-II, DALG, DBDRIVE, DFDRIVE, CDRIVE, CBDRIVE, CFDRIVE) were briefly discussed in section 3.4.2, only the subordinate functions used by these will be described here.

DEFINE: All variables are defined in this function. In the APL description, it turned out to be convenient to encode the variables 0, 1, x, D and $\overline{D}$ as o, 1, -1, 2 and -2 respectively. Then use could be made of the facts that $\overline{D} \equiv -D$ and that adding +3 to a line signal value would make this value available as a subscript in the XOR table lookup with the matrix XORM. The coding for the logic function names was arbitrary. Implementing some of the tests in implication was eased by defining the matrix M which was then used in classifying the four vertex functions.

P: The predecessors of block B are extracted from PL and returned.

S: The successors of block B are extracted and returned.

CLIP: the first N components of the vector V are clipped off the V and the result used to respecify Z.

LMERGE: The two low order sorted vectors V and W are merged to form a low order sorted vector Z. When duplicate entries appear in V and W, only one such entry is placed in Z.

HMERGE: This function is similar to LMERGE except it uses high order sorting on all vectors.

AUGMODIN: All the successors of the blocks listed in the first argument, V, which are not already in MODIN or are not equal to the second argument, B, are placed in MODIN. In practice, V will be the list of predecessors of block B which were just set to 1 or 0. When MODIN is updated, there is no reason to add B to MODIN since it has just been checked.

ADEC: If B is in the activity vector, A, it is deleted and the increment and decrement lists updated: if B is in INC, it is deleted, if not, then B is added to the decrement list.

STORE-EXTRACT: These are a complementary pair of functions for storing and extracting the lists necessary to record the status of the solution at a given point. The particular catenation process for storing vectors which is used here was necessary because the APL interpreter did not have a "list of lists" feature. The saving of SETCO, INC., and DEC are rather obvious. The variable TAG is used to record information concerning why this particular save was made:

TAG =	0	first pdc intersection for an exclusive -OR.
=	1	activity vector had more than one entry.
=	n	the input being examined on DB during CDRIVE.

Before STORE is executed, it is customary to store the decision block number, DB, in a list LDB. During CDRIVE , it is stored negatively in order to distinguish this mode.

<u>CLEARLIST</u>: The lists used to store the status of the global variables are initially cleared with this function.

<u>CDEF:</u> The initial consistency vector C is defined with this function. Only those blocks below the latest block for which pdc was formed need be examined.

<u>CDEC:</u> If block B is in the consistency vector, it is deleted and either the increment or decrement list updated depending on where B is stored.

<u>CHANGEPL</u>: The predecessors for block I can be respecified in PL.

<u>SLDEF</u>: Given PL and IPL, SL and ISL are defined.

<u>LISTPL</u>: The predecessors for each block are listed.

<u>LISTSL</u>: The successors for each block are listed.

```
      ∇ DALGII;I;DV;R;LI
[1]     LI←⍳⍴L
[2]     I←0
[3]     'ENTER LINES TO BE TESTED.'
[4]     LINES←,⎕
[5]     'ENTER STUCK VALUES FOR THESE LINES.'
[6]     VALUES←,⎕
[7]     DGT:→(I≥⍴LINES)/0
[8]     I←I+1
[9]     PDCFO←LINES[I]
[10]    PDCFIN←P PDCFO
[11]    VALUE←VALUES[I]
[12]    ('TEST FOR LINE ';PDCFO;' STUCK-AT-';VALUE)
[13]    DV←(D,-D)[VALUE+1]
[14]    →(L[PDCFO]≠XOR)/DGV
[15]    MPDCF←(2 3)⍴1,VALUE,DV,0,(~VALUE),DV
[16]    →DGC
[17]    DGV:R←⍴PDCFIN
[18]    →(((~VALUE)∧L[PDCFO]∊NAND,OR)∨VALUE∧L[PDCFO]∊AND,NOR)/DGM
[19]    MPDCF←(1,R+1)⍴(R⍴L[PDCFO]∊AND,NAND),DV
[20]    →DGC
[21]    DGM:MPDCF←(R,R+1)⍴(L[PDCFO]∊OR,NOR),(R+1)⍴X
[22]    MPDCF[;R+1]←R⍴DV
[23]    DGC:DALG
[24]    →FAIL/DGN
[25]    (' D̈X01D')[1+((2×(⍴LI))⍴ 0 1)\TC[LI]+3]
[26]    →DGT
[27]    DGN:'NO TEST'
[28]    →DGT
      ∇
```

```
     ∇ DALG;FC;PDB;PDBX
[1]    CLEARLIST
[2]    FC←0
[3]    DPDCF:→(FC=ρMPDCF[;1])/DFAIL
[4]    POTEST←0
[5]    FC←FC+1
[6]    TC←(ρL)ρX
[7]    TC[PDCFIN,PDCFO]←MPDCF[FC;]
[8]    SETCO←PDCFO,(TC[PDCFIN]≠X)/PDCFIN
[9]    A←S PDCFO
[10]   INC←A
[11]   MODIN←DEC←⍳0
[12]   SETCO AUGMODIN PDCFO
[13]   JINT←1
[14]   DB←PDCFO
[15]   →DBPO
[16]   DEXT:→(0=ρA)/DBACKUP
[17]   DB←A[1]
[18]   TAG←0
[19]   →(L[DB]≠XOR)/DSTORE
[20]   DATEST:TAG←1
[21]   →(1=ρA)/DRIVE
[22]   DSTORE:LDB←DB,LDB
[23]   STORE
[24]   SETCO←INC←DEC←⍳0
[25]   DRIVE:PDB←P DB
[26]   PDBX←(TC[PDB]=X)/PDB
[27]   TC[PDBX]←(ρPDBX)ρ(TAG,L[DB]∊AND,NAND)[1+L[DB]≠XOR]
[28]   TC[DB]←(D,-D)[1+(∨/TC[PDB]=D)=(L[DB]∊NOR,NAND)∨(L[DB]=XOR)∧TAG=1]
[29]   DUPDATE:JINT←1+ρSETCO
[30]   SETCO←SETCO,DB,PDBX
[31]   ADEC DB
[32]   (PDBX,DB)AUGMODIN DB
[33]   DBPO:→(~DB∊PO)/DBDRIV
[34]   POTEST←1
[35]   DBDRIV:DB←DB
[36]   DBDRIVE
[37]   →FAIL/DBACKUP
[38]   DFDRIVE
[39]   →(FAIL∨∧/TC[PO]∊ 0 1)/DBACKUP
[40]   →(~POTEST)/DEXT
[41]   LDB←DB,LDB
[42]   STORE
[43]   CDRIVE
[44]   →(~FAIL)/0
[45]   POTEST←0
[46]   DBACKUP:→(0=ρLTAG)/DPDCF
[47]   MODIN←⍳0
[48]   TC[SETCO]←(ρSETCO)ρX
[49]   A←((~A∊INC)/A)LMERGE DEC
[50]   EXTRACT
[51]   DBACK←DBACK+1
[52]   →(~TAG)/DATEST
[53]   ADEC DB
[54]   →DEXT
[55]   DFAIL:FAIL←1
     ∇
     ∇ CDRIVE;PDBX
[1]    CDEF
[2]    CNEXT:→(0=⍴C)/CTEST
[3]    DB←C[1]
[4]    PDBX←(TC[P DB]=X)/P DB
[5]    TAG←⍴PDBX
[6]    CSETIN:→(L[DB]≠XOR)/CVERT
[7]    TC[PDBX]←(TAG-1),(TAG-1)≠TC[DB]
[8]    SETCO←PDBX
[9]    →CINT
[10]   CVERT:TC[PDBX[TAG]]←L[DB]∊OR,NOR
[11]   SETCO←,PDBX[TAG]
[12]   CINT:JINT←0
[13]   SETCO AUGMODIN DB
[14]   INC←⍳0
[15]   DEC←⍳0
[16]   CBDRIVE
[17]   →FAIL/CBACKUP
[18]   CFDRIVE
[19]   →FAIL/CBACKUP
[20]   LDB←(-DB),LDB
[21]   C←C CLIP 1
[22]   DEC←DB,DEC
[23]   STORE
[24]   C←((~C∊DEC)/C)HMERGE INC
[25]   →CNEXT
[26]   CBACKUP:MODIN←⍳0
[27]   TC[SETCO]←(⍴SETCO)⍴X
[28]   TAG←TAG-1
[29]   →(0≠TAG)/CSETIN
[30]   EXTRACT
[31]   CBACK←CBACK+1
[32]   →(DB>0)/0
[33]   DB←-DB
[34]   C←((~C∊INC)/C)HMERGE DEC
[35]   PDBX←(TC[P DB]=X)/P DB
[36]   →CBACKUP
[37]   CTEST:FAIL←0
     ∇
     ∇ DBDRIVE;J;LINE;PRED;XPRED;TCV
[1]    DBBRAN←4ρDBFAIL,DBXOR,DBSTEP,DBSTEP
[2]    J←JINT
[3]    DBSTEP:→(J=ρSETCO)/DBEND
[4]    J←J+1
[5]    LINE←SETCO[J]
[6]    →(LINE∊PI)/DBSTEP
[7]    PRED←P LINE
[8]    XPRED←(TC[PRED]=X)/PRED
[9]    MODIN←(MODIN≠LINE)/MODIN
[10]   →(L[LINE]≠XOR)/DBVERT
[11]   →DBBRAN[XORM[TC[PRED[1]]+3;TC[PRED[2]]+3;TC[LINE]+2]]
[12]   DBXOR:TC[XPRED]←TC[LINE]≠TC[(PRED≠XPRED)/PRED]
[13]   →DBUPDATE
[14]   DBVERT:TCV←(∨/M=L[LINE])/TC[LINE],~TC[LINE]
[15]   →((TCV=L[LINE]∊OR,NOR),(∨/TC[PRED]∊D,(-D),~TCV),(0=ρXPRED),1)/
       DBTWO,DBFAIL,DBSTEP,DBSET
[16]   DBTWO:→(((∨/TC[PRED]=TCV)∨1<ρXPRED),0=ρXPRED)/DBSTEP,DBFAIL
[17]   DBSET:TC[XPRED]←(ρXPRED)ρTCV
[18]   DBUPDATE:SETCO←SETCO,XPRED
[19]   XPRED AUGMODIN LINE
[20]   →DBSTEP
[21]   DBFAIL:FAIL←1
[22]   →0
[23]   DBEND:FAIL←0
     ∇


     ∇ CBDRIVE;J;LINE;PRED;XPRED;TCV
[1]    CBBRAN←4ρCBFAIL,CBXOR,CBSTEP,CBADTOC
[2]    J←JINT
[3]    CBSTEP:→(J=ρSETCO)/CBEND
[4]    J←J+1
[5]    LINE←SETCO[J]
[6]    →(LINE∊PI)/CBSTEP
[7]    PRED←P LINE
[8]    XPRED←(TC[PRED]=X)/PRED
[9]    MODIN←(MODIN≠LINE)/MODIN
[10]   →(L[LINE]≠XOR)/CBVERT
[11]   →CBBRAN[XORM[TC[PRED[1]]+3;TC[PRED[2]]+3;TC[LINE]+2]]
[12]   CBXOR:TC[XPRED]←TC[LINE]≠TC[(PRED≠XPRED)/PRED]
[13]   →CBUPDATE
[14]   CBVERT:TCV←(∨/M=L[LINE])/TC[LINE],~TC[LINE]
[15]   →((TCV=L[LINE]∊OR,NOR),(∨/TC[PRED]∊D,(-D),~TCV),(0=ρXPRED),1)/
       CBTWO,CBFAIL,CBSTEP,CBSET
[16]   CBTWO:→((∨/TC[PRED]=TCV),(0=ρXPRED),1=ρXPRED)/CBSTEP,CBFAIL,
       CBSET
[17]   CBADTOC:INC←INC HMERGE LINE
[18]   →CBSTEP
[19]   CBSET:TC[XPRED]←(ρXPRED)ρTCV
[20]   CBUPDATE:SETCO←SETCO,XPRED
[21]   XPRED AUGMODIN LINE
[22]   →CBSTEP
[23]   CBFAIL:FAIL←1
[24]   →0
[25]   CBEND:FAIL←0
     ∇
     ∇ DFDRIVE;LINE;PRED;XPRED;VI;VC
[1]    DFBRAN←9ρDFFAIL,DXI,DFSTEP,DFSTEP,DXO,DXD,DXDB,DFSTEP,DFADA
[2]    DFSTEP:→(0=ρMODIN)/DFEND
[3]    LINE←MODIN[1]
[4]    MODIN←MODIN CLIP 1
[5]    PRED←P LINE
[6]    XPRED←(TC[PRED]=X)/PRED
[7]    →(L[LINE]≠XOR)/DFVERT
[8]    →DFBRAN[XORM[TC[PRED[1]]+3;TC[PRED[2]]+3;TC[LINE]+2]]
[9]    DXI:TC[XPRED]←TC[LINE]≠TC[(PRED≠XPRED)/PRED]
[10]   →DFMOD
[11]   DXO:TC[LINE]←TC[PRED[1]]≠TC[PRED[2]]
[12]   →DFDEC
[13]   DXD:TC[LINE]←D
[14]   →DTPO
[15]   DXDB:TC[LINE]←-D
[16]   →DTPO
[17]   DFVERT:VI←L[LINE]∊OR,NOR
[18]   VC←L[LINE]∊NAND,NOR
[19]   →(((∨/TC[PRED]=VI)∨(∨/TC[PRED]=D)∧∨/TC[PRED]=-D),(1=ρXPRED),
       0=ρXPRED)/DFOUT,DFONEX,DFNOX
[20]   →(TC[LINE]≠X)/DFSTEP
[21]   DINTEST:→(∧/TC[PRED]∊ 0 1 ,X)/DFSTEP
[22]   DFADA:→(LINE∊A)/DFSTEP
[23]   A←A LMERGE LINE
[24]   INC←INC LMERGE LINE
[25]   →DFSTEP
[26]   DFONEX:→(TC[LINE]=X)/DINTEST
[27]   TC[XPRED]←VI
[28]   DFMOD:JINT←ρSETCO
[29]   SETCO←SETCO,XPRED
[30]   XPRED AUGMODIN LINE
[31]   DBDRIVE
[32]   →FAIL/0
[33]   →DFSTEP
[34]   DFNOX:→(TC[LINE]≠X)/DFFAIL
[35]   →(∧/TC[PRED]=~VI)/DFSETO
[36]   TC[LINE]←((∨/TC[PRED]=D)=∨/M=L[LINE])/D,-D
[37]   DTPO:→(~LINE∊PO)/DFDEC
[38]   POTEST←1
[39]   DFDEC:ADEC LINE
[40]   DFUPDATE:SETCO←SETCO,LINE
[41]   LINE AUGMODIN 0
[42]   →DFSTEP
[43]   DFSETO:TC[LINE]←VI=VC
[44]   →DFUPDATE
[45]   DFOUT:→(TC[LINE]≠X)/DFSTEP
[46]   TC[LINE]←VI≠VC
[47]   →DFDEC
[48]   DFFAIL:FAIL←1
[49]   →0
[50]   DFEND:FAIL←0
     ∇
     ∇ CFDRIVE;LINE;PRED;XPRED;VI;VC
[1]    CFBRAN←9⍴CFFAIL,CXI,CFDEC,CFFAIL,CXO,CXD,CXDB,CFSTEP,CFSTEP
[2]    CFSTEP:→(0=⍴MODIN)/CFEND
[3]    LINE←MODIN[1]
[4]    MODIN←MODIN CLIP 1
[5]    →(LINE>DB)/CFSTEP
[6]    PRED←P LINE
[7]    XPRED←(TC[PRED]=X)/PRED
[8]    →(L[LINE]≠XOR)/CFVERT
[9]    →CFBRAN[XORM[TC[PRED[1]]+3;TC[PRED[2]]+3;TC[LINE]+2]]
[10]   CXI:TC[XPRED]←TC[LINE]≠TC[(PRED≠XPRED)/PRED]
[11]   →CFMOD
[12]   CXO:TC[LINE]←TC[PRED[1]]≠TC[PRED[2]]
[13]   →CFADD
[14]   CXD:TC[LINE]←D
[15]   →CFADD
[16]   CXDB:TC[LINE]←-D
[17]   →CFADD
[18]   CFVERT:VI←L[LINE]∊OR,NOR
[19]   VC←L[LINE]∊NAND,NOR
[20]   →(((∨/TC[PRED]=VI)∨(∨/TC[PRED]=D)∧∨/TC[PRED]=-D),(1<⍴XPRED)
       ,0=⍴XPRED)/CFOUT,CFSTEP,CFNOX
[21]   →(TC[LINE]=X)/CFSTEP
[22]   TC[XPRED]←VI
[23]   CFMOD:LINE←LINE
[24]   CDEC LINE
[25]   JINT←⍴SETCO
[26]   SETCO←SETCO,XPRED
[27]   XPRED AUGMODIN LINE
[28]   CBDRIVE
[29]   →FAIL/0
[30]   →CFSTEP
[31]   CFNOX:→(TC[LINE]≠X)/CFFAIL
[32]   TC[LINE]←((∧/TC[PRED]=~VI),(∨/TC[PRED]≠~VI)∧(∨/TC[PRED]=D)=
       ∨/M≠L[LINE])/(VI=VC),D,-D
[33]   CFADD:SETCO←SETCO,LINE
[34]   LINE AUGMODIN 0
[35]   →CFSTEP
[36]   CFOUT:→(TC[LINE]≠X)/CFDEC
[37]   TC[LINE]←VI≠VC
[38]   →CFADD
[39]   CFDEC:CDEC LINE
[40]   →CFSTEP
[41]   CFEND:FAIL←0
[42]   →0
[43]   CFFAIL:FAIL←1
     ∇
```

```
      ∇ DEFINE
[1]     D←2
[2]     X←-1
[3]     AND←1
[4]     NAND←2
[5]     OR←3
[6]     NOR←4
[7]     XOR←5
[8]     M←(2 2)ρOR,AND,NOR,NAND
[9]     XORM←(5 5 3)ρ 5 3 1 9 1 1 7 1 1 6 1 1 5 1 3
        9 1 1 8 4 4 8 2 2 8 2 2 9 1 1 7 1 1 8 2 2 5
        3 1 5 1 3 6 1 1 6 1 1 8 2 2 5 1 3 5 3 1 7 1
        1 5 1 3 9 1 1 6 1 1 7 1 1 5 3 1
[10]    ALL←12321
[11]    END←98789
[12]    TIME←NUM←CLOCK←9ρ0
      ∇


      ∇ CLEARLIST
[1]     LDB←LTAG←⍳0
[2]     LNSET←LNINC←LNDEC←⍳0
[3]     LSETCO←LINE←LDEC←⍳0
      ∇


      ∇ Z←P B
[1]     Z←⍳0
[2]     →(0≥B←B-ρPI)/0
[3]     Z←PL[IPL[B]+⍳IPL[B+1]-IPL[B]]
      ∇


      ∇ Z←S B
[1]     Z←SL[ISL[B]+⍳ISL[B+1]-ISL[B]]
      ∇


      ∇ Z←V CLIP N
[1]     Z←(~(ρV)⍺N)/V
      ∇
```

```
     ∇ Z←V LMERGE W;K
[1]    K←0
[2]    Z←,V,W
[3]    →(0=ρV)/0
[4]    W←,W
[5]    →(K=ρW)/0
[6]    K←K+1
[7]    Z←((Z<W[K])/Z),W[K],(Z>W[K])/Z
[8]    →5
     ∇


     ∇ Z←V HMERGE W;K
[1]    K←0
[2]    Z←,V,W
[3]    →(0=ρV)/0
[4]    W←,W
[5]    →(K=ρW)/0
[6]    K←K+1
[7]    Z←((Z>W[K])/Z),W[K],(Z<W[K])/Z
[8]    →5
     ∇


     ∇ V AUGMODIN B;K
[1]    V←,V
[2]    K←0
[3]    →(K=ρV)/0
[4]    K←K+1
[5]    MODIN←MODIN LMERGE((S V[K])≠B)/S V[K]
[6]    →3
     ∇


     ∇ ADEC B
[1]    →(~B∊A)/0
[2]    A←(A≠B)/A
[3]    →(B∊INC)/6
[4]    DEC←DEC LMERGE B
[5]    →0
[6]    INC←(INC≠B)/INC
     ∇
```

```
      ∇ CDEF;K;PK;PKX;DI
[1]     C←⍳0
[2]     K←⍴PI
[3]     →(K≥DB-1)/0
[4]     K←K+1
[5]     →((~TC[K]∊ 1 0),L[K]=XOR)/ 3 7
[6]     →(((TC[K]=1)∧L[K]∊AND,NOR)∨(TC[K]=0)∧L[K]∊OR,NAND)/3
[7]     PK←P K
[8]     PKX←(TC[PK]=X)/PK
[9]     →(((L[K]=XOR)∧2=⍴PKX),L[K]=XOR)/ 12 3
[10]    DI←L[K]∊OR,NOR
[11]    →((0=⍴PKX)∨(∨/TC[PK]=DI)∨(∨/TC[PK]=D)∧∨/TC[PK]=-D)/3
[12]    C←K,C
[13]    →3
      ∇


      ∇ CDEC B
[1]     →((B∊C),(B∊INC),1)/ 2 4 0
[2]     DEC←DEC HMERGE B
[3]     →0
[4]     INC←(INC≠B)/INC
      ∇


      ∇ STORE
[1]     LTAG←TAG,LTAG
[2]     LNSET←(⍴SETCO),LNSET
[3]     LNINC←(⍴INC),LNINC
[4]     LNDEC←(⍴DEC),LNDEC
[5]     LSETCO←SETCO,LSETCO
[6]     LINC←INC,LINC
[7]     LDEC←DEC,LDEC
      ∇


      ∇ EXTRACT
[1]     TAG←LTAG[1]
[2]     LTAG←LTAG CLIP 1
[3]     DB←LDB[1]
[4]     LDB←LDB CLIP 1
[5]     NSET←LNSET[1]
[6]     LNSET←LNSET CLIP 1
[7]     NINC←LNINC[1]
[8]     LNINC←LNINC CLIP 1
[9]     NDEC←LNDEC[1]
[10]    LNDEC←LNDEC CLIP 1
[11]    SETCO←LSETCO[⍳NSET]
[12]    LSETCO←LSETCO CLIP NSET
[13]    DEC←LDEC[⍳NDEC]
[14]    LDEC←LDEC CLIP NDEC
[15]    INC←LINC[⍳NINC]
[16]    LINC←LINC CLIP NINC
      ∇
```

```
     ∇ CIRCUIT;I;V
[1]    'ENTER PRIMARY INPUTS.'
[2]    I←ρPI←,⎕
[3]    'ENTER PRIMARY OUTPUTS.'
[4]    PO←,⎕
[5]    'ENTER LOGIC.'
[6]    L←(Iρ0),(L≠0)/L←,⎕
[7]    'ENTER PREDECESSORS FOR THE INDICATED BLOCK.'
[8]    PL←SL←⍳IPL←ISL←0
[9]    ('BLOCK ';I←I+1)
[10]   PL←PL,⎕
[11]   IPL←IPL,ρPL
[12]   →(I<ρL)/9
[13]   I←1
[14]   SL←SL,(ρPI)++/((PL=I)/⍳ρPL)∘.>IPL
[15]   ISL←ISL,ρSL
[16]   →((ρL)≥I←I+1)/14
[17]   'END OF CIRCUIT DEFINITION.'
[18]   →(0=ρI←(L=XOR)/⍳ρL)/0
[19]   →(∧/2=IPL[I+1]-IPL[I←I-ρPI])/0
[20]   'INCORRECT NUMBER OF INPUTS ON XOR.'
     ∇


     ∇ CHANGEPL I;J
[1]    I←I-ρPI
[2]    J←ρPL
[3]    PL←PL[⍳IPL[I]],⎕,PL[IPL[I+1]+⍳(ρPL)-IPL[I+1]]
[4]    IPL←IPL+(Iρ0),((ρIPL)-I)ρ(ρPL)-J
     ∇


     ∇ SLDEF;I
[1]    SL←⍳ISL←0
[2]    I←1
[3]    SL←SL,(ρPI)++/((PL=I)/⍳ρPL)∘.>IPL
[4]    ISL←ISL,ρSL
[5]    →((ρL)≥I←I+1)/3
     ∇


     ∇ LISTPL;I
[1]    I←1+ρPI
[2]    (I;' :  ';P I)
[3]    →((ρL)≥I←I+1)/2
     ∇


     ∇ LISTSL;I
[1]    I←1
[2]    (I;' :  ';S I)
[3]    →((ρL)≥I←I+1)/2
     ∇
```

Appendix 1.2 TESTDETECT

Glossary of Major Variables

TESTDETECT uses many of the defined in DALG-II plus the following:

FD	- a vector of lines which are failure detected by the given test.
DL	- a vector containing a list of lines which have D's on them and successors in A.
FO	- the fan-out or number of successors in A for each corresponding line in DL.
A1	- the first (and lowest) element in A: the one for which D-chain extension is being sought.
PA1	- the list of predecessors of A1.
PDL	- the list of predecessors of A1 which are also in DL, i.e. the ones which are thought of as having D and/or $\overline{D}$ values.
PNDL	- the list of predecessors of A1 which are not in DL
SA1	- successors of A1.
TM	- a matrix with each row containing a test for which TEST-DETECT is to be run.

```
     ∇ TESTDETECT;I;DL;FO;A1;PA1;PNDL;PDL
[1]    TDRIVE
[2]    FD←PO
[3]    I←1+(ρL)[1]
[4]    NEXT:→(0=I←I-1)/PRINT
[5]    →(I∊FD)/NEXT
[6]    DL←I
[7]    A←S DL
[8]    FO←ρA
[9]    PICK:A1←A[1]
[10]   A←A CLIP 1
[11]   PA1←P A1
[12]   PNDL←(~PA1∊DL)/PA1
[13]   PDL←(PA1∊DL)/PA1
[14]   FO←FO-(DL∊PDL)
[15]   DL←(FO≠0)/DL
[16]   FO←(FO≠0)/FO
[17]   →(L[A1]=XOR)/TXOR
[18]   →(0=ρPNDL)/DTEST
[19]   →(L[A1]∨.=OR,NOR)/TOR
[20]   TAN:→(∧/TC[PNDL])/DTEST
[21]   →DLTEST
[22]   TOR:→(∨/TC[PNDL])/DLTEST
[23]   DTEST:→(1=ρPDL)/A1TEST
[24]   →((∧/TC[PDL])∨(∧/~TC[PDL]))/A1TEST
[25]   →DLTEST
[26]   TXOR:→(1≠ρPDL)/DLTEST
[27]   A1TEST:→(A1∊PO)/AUG
[28]   DL←DL,A1
[29]   SA1←S A1
[30]   FO←FO,ρSA1
[31]   A←A LMERGE SA1
[32]   DLTEST:→(1<ρDL)/PICK
[33]   →(0=ρDL)/NEXT
[34]   →((DL=I),~DL∊FD)/PICK,NEXT
[35]   AUG:FD←I,FD
[36]   →NEXT
[37]   PRINT:A←(ιρL)∊FD
[38]   I←(1+TC[PI]),3,(A∧TC)+2×A∧~TC
[39]   (' 01|')[1+((2×ρI)ρ 0 1)\I]
     ∇


     ∇ TDRIVE;I;T
[1]    I←(ρ,PI)[1]+1
[2]    T←TC[P I]
[3]    TC[I]←(L[I]=AND,NAND,OR,NOR,XOR)/(∧/T),(~∧/T),
       (∨/T),(~∨/T),(≠/T)
[4]    →((ρL)≥I←I+1)/2
     ∇
     ∇ PATTERNS;I;J
[1]    TM←⍳J←0
[2]    'ENTER PRIMARY INPUT PATTERNS, ONE AT A TIME.'
[3]    'TYPE  ALL  IF ALL 2*⍴PI PATTERNS ARE DESIRED.'
[4]    'TYPE  END  TO STOP DATA ENTRY.'
[5]    I←,⎕
[6]    →(I[1]=END,ALL)/ 0 16
[7]    →((⍴I)=⍴PI)/10
[8]    'INCORRECT NUMBER OF DIGITS. REENTER.'
[9]    →5
[10]   →(0≠(⍴TM)[1])/14
[11]   TM←(1,⍴PI)⍴I
[12]   →(J=0)/5
[13]   →17
[14]   TM←((⍴TM)+ 1 0)⍴(,TM),I
[15]   →12
[16]   J←0
[17]   →((2*⍴PI)<J←J+1)/0
[18]   I←((⍴PI)⍴2)⊤J-1
[19]   →10
     ∇


     ∇ DETECT;I;END
[1]    I←(' 0123456789')[1+((2×⍴L)⍴ 0 1)\1+10|⍳⍴L]
[2]    I[⍳2×⍴PI],' |',I
[3]    ((2×⍴PI)⍴'-'),'-+',(⍴I)⍴'-'
[4]    I←0
[5]    TC←(⍴L)⍴X
[6]    →(((⍴TM)[1])<I←I+1)/0
[7]    TC[PI]←TM[I;]
[8]    TESTDETECT
[9]    →5
     ∇
```

Appendix 2. Reliability Derivations

2.1 Derivation of Miscellaneous Formulae

2.1.1 A bound for the approximation error in equation 4.15 for $^1R_1(\lambda, \mu)$.

As shown previously, if $K = \lambda/\mu$,

$$^1R_1(\lambda, \mu) = e^{-\lambda T}(1+\lambda T) + K^2 e^{-\lambda T}\{1 + \lambda T)[1-(1+\mu T)-e^{-\mu T}]$$

$$-2(K-D[1-(1+\mu T+\frac{(\mu T)^2}{2})e^{-\mu T}]\}$$

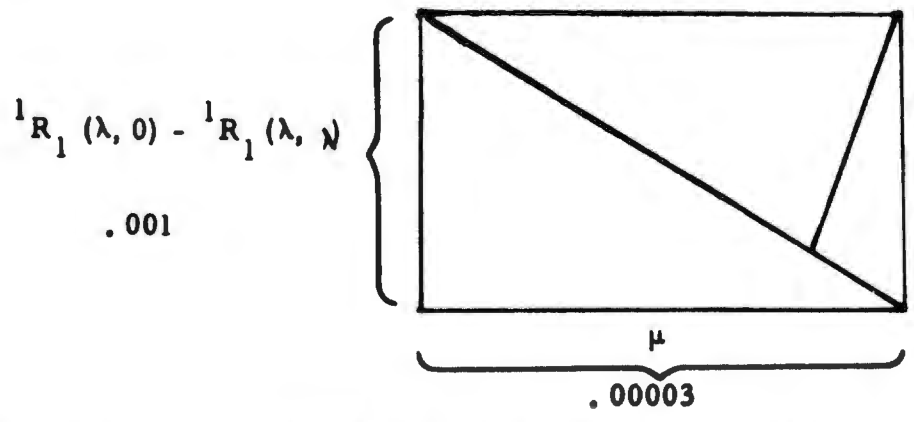
So,

$$\frac{\partial\, ^1R_1(\lambda, 0)}{\partial\mu} = \frac{-\lambda^3 T^4}{12} e^{-\lambda T}$$

$$\frac{\partial\, ^1R_1(\lambda, \lambda)}{\partial\mu} = -2Te^{-\lambda T}[1-(1+\lambda T+\frac{(\lambda T)^2}{2})e^{-\lambda T}]$$

If $T = 10,000$, $\lambda T = .3$, $\frac{\partial\, ^1R_1(\lambda, 0)}{\partial\mu} = -16.7$,

$\frac{\partial\, ^1R_1(\lambda, \lambda)}{\partial\mu} = -54$. The error lies in the triangle shown.



The max error is .00005 or 5% of the failure (.001).

An APL program was written to determine bounds in following cases:

q	1	λT	${}^{f}R_{m}^{q}(\lambda, 0)$	Error Bound
1	1	.3	0.99973	0.00110
2	1	.3	0.99812	0.00211
3	1	.3	0.99527	0.00306
4	1	.3	0.99280	0.00546
1	1	.2	0.99994	0.00025
2	1	.2	0.99958	0.00049
3	1	.2	0.99892	0.00072
4	1	.2	0.99830	0.00126
2	2	.3	0.99979	0.00017
3	2	.3	0.99989	0.00037
4	2	.3	0.99972	0.00064
1	2	.1	$1-1.3\times 10^{-9}$	1.01×10^{-7}

For all other cases with larger m or smaller λT the error bound was less than .00008. A more typical value is show above for $q = 1$, $m = 2$, and $\lambda T = 0.1$.

2.1.2 Algebraic Manipulations

The manipulations necessary to obtain eqs. 4.16 are based on the following identities, which can be proved by mathematical induction. The notation $\binom{x}{y}$ stands for a binomial coefficient.

$$a) \sum_{j=0}^{x} \binom{x+z+1}{j} Y^j (1-Y)^{x-j} = \sum_{j=0}^{x} \binom{z+j}{j} Y^j$$

$$b) \; {}^{f}R(\lambda, \lambda) = e^{-\lambda T} \sum_{i=0}^{f} \frac{(\lambda T)^i}{j!} = 1 - \frac{(\lambda T)^{f+1}}{(f+1)!} \sum_{j=0}^{\infty} \frac{f+1}{f+1+j} (-1)^j \frac{(\lambda T)^j}{j!}$$

$$c) \sum_{j=0}^{x} \binom{z+j}{j} = \binom{z+x+1}{x}$$

If $\mu = \lambda$, the assemblage with q active elements and m spares is operative as long as any combination of q modules is operative.

$${}^{f}R^{q}_{m}(\lambda, \lambda) = \sum_{j=q+m}^{q} \binom{q+m}{j} ({}^{f}R(\lambda, \lambda))^j (1 - {}^{f}R(\lambda, \lambda))^{q+m-j}$$

$$= ({}^{f}R(\lambda, \lambda))^q \sum_{j=0}^{m} \binom{q+m}{j} ({}^{f}R(\lambda, \lambda))^{m-j} (1 - {}^{f}R(\lambda, \lambda))^j$$

Using a),

$${}^{f}R^{q}_{m}(\lambda, \lambda) = ({}^{f}R(\lambda, \lambda))^q \sum_{j=0}^{m} \binom{q+j-1}{j} (1 - {}^{f}R(\lambda, \lambda))^j$$

which is equation 4.16).

Representing the reliability as 1 minus the probability failure,

$$^{f}R^{q}_{m}(\lambda, \lambda) = 1 - \sum_{j=0}^{q-1} \binom{q+m}{j} ({}^{f}R(\lambda, \lambda))^{j} (1 - {}^{f}R(\lambda, \lambda))^{q+m-j}$$

$$= 1 - (1 - {}^{f}R(\lambda, \lambda))^{m+1} \sum_{j=0}^{q-1} \binom{q+m}{j} ({}^{f}R(\lambda, \lambda))^{j} (1 - {}^{f}R(\lambda, \lambda))^{q-j-1}$$

Using a) again

$$^{f}R^{q}_{m}(\lambda, \lambda) = 1 - (1 - {}^{f}R(\lambda, \lambda))^{m+1} \sum_{j=0}^{q-1} \binom{m+j}{j} ({}^{f}R(\lambda, \lambda))^{j}$$

which is equation 4.16 .

To prove equation 4.17 use c) in the summation term in equation 4.16 .

$$\binom{m+q}{q-1} = \sum_{j=0}^{q-1} \binom{m+j}{j} \geq \sum_{j=0}^{q-1} \binom{m+j}{j} ({}^{f}R(\lambda, \lambda))^{j} = B > 0$$

Write b) as

$$1 - \frac{(\lambda T)^{f+1}}{(f+1)!} \quad S,$$

and note that $0 < S < 1$ for $\lambda T < 1$. So, substituting in equation 4.16

$$^{f}R^{q}_{m}(\lambda, \lambda) = 1 - \left(\frac{(\lambda T)^{f+1}}{(f+1)!}\right)^{m+1} (1 - S)^{m+1} \quad B$$

$$= 1 - \binom{m+q}{q-1} \left[\frac{(\lambda T)^{f+1}}{(f+1)!}\right]^{m+1}$$

$$\left[\frac{(\lambda T)^{f+1}}{(f+1)!}\right]^{m+1} \quad \left[\binom{m+q}{q-1} - B(1 - s)^{m+1}\right]$$

Since $(i - S)^{m+1} < 1$, equation 4, 7is proved.

2.2 Derivation of Approximation Equations, Markov Process

2.2.1 Introduction

The models for reliability using spares employ equations which assume that the spares deteriorated as rapidly as the units being used. This is too pessimistic, so a more realistic equation which employs two failure rates; λ, the failure rate during power-on use and μ, the failure rate during (power-off) idleness will be developed. To facilitate the discussion, the following notation has been adopted:

λ -	failure rate druing power-on
μ -	failure rate during power-off
K -	λ/μ (usually > 1).
m -	number of spares initially available.
q -	number of modules required to be operating at all times (power-on).
n -	total number present $(m + q)$.
f -	number of failures each unit can withstand before malfunctioning.
T -	mission time.
c -	failure coverage, or probability, that given a failure occurs, it is detected by the checking and/or diagnostic tests and that successful recovery occurs.
D -	duty cycle or fraction of time the machine is required to be on.

The parameters which effect the form of the reliability equations are m, q, c, and f. In order to display the status of these parameters each reliability equation will be written as ${}^{f}_{c}R^{q}_{m}$. When any of the parameters q, c, or f are elided, convention will be used to establish the values $q = 1$, $c = 1$, and $f = 0$.

2.2.2 Formulas for R_m^q

The formula $R_0^q(T) = e^{-q\lambda T}$

$$2.1) \quad R_m^q(T) = e^{-q\lambda T}\{1 + \sum_{i=1}^{m}[\frac{(1-e^{-\mu T})^i}{i!}\prod_{j=0}^{i=1}(j+qK)]\} \qquad m \geq 0$$

will be proved first. The first equation is well known. The formulation is recursive for the second. The probability of successful operation with m spares during the time interval plus the probability of a failure in the (n-1) spare system at time $t < T$ multiplied by the probability that the m^th is good at time t, and the probability that the refreshed system lasts from t to T.

Assume an exponential failure rate. Then the probability of successful operation of a q unit is $e^{-q\lambda t}$ and the probability of successful shelf life of one unit is $e^{-\mu T}$. The integral equation to be solved is written as

$$2.2) \quad R_m^q(T) = R_{m-1}^q(T) + \int_0^T \frac{d}{dt}(1 - R_{m-1}^q(t)) e^{-\mu t} \cdot e^{-q(T-t)\lambda} dt$$

This integral equation has as solution $R_m^q(T)$ as expressed in equation 2.1), and moreover,

$$2.3) \quad \frac{d}{dt} R_m^q(T) = -q\lambda e^{-q\lambda T}\frac{(1-e^{-\mu T})^m}{m!}\prod_{j=1}^{m}(j+qK) \qquad m \geq 1$$

The proof is by induction. If $m = 1$,

$$R_1^q(T) = e^{-q\lambda T} + \int q\lambda e^{-q\lambda t} \cdot e^{-\mu t} \cdot e^{-q(T-t)} dt$$

$$= e^{-q\lambda T}(1 + qK(1 - e^{-\mu T}))$$

$$\frac{d}{dT}R_1^q(T) = -q\lambda e^{-q\lambda T}(1+qK)(1-e^{-\mu T})$$

and the formulae hold.

Given equation 2. 3, the induction step for equation 2. 1 follows directly from equation 2. 2, since the terms $e^{-q\lambda t}$ and $e^{q\lambda t}$ cancel and the integral left has the form of

$$2.4)\quad R^q_m(T) = R^q_{m-1}(T) + \frac{qKe^{-q\lambda T}}{m!}\left[\prod_{j=1}^{m}(j+qK)\right]\frac{1}{m+1}(1-e^{-\mu t})^{m+1}$$

which easily reduces to the correct form.

Differentiating equation 2. 4,

$$\frac{d}{dt}R^q_{m+1}(T) = \frac{d}{dt}R^q_m(T) + \frac{qK}{(m+1)!}\left[\prod_{j=1}^{m}(j+qK)\right][-q\lambda e^{-q\lambda T}(1-e^{-\mu T})^{m+1}$$

$$+(m+1)(1-e^{-\mu T})^m e^{-q\lambda T}\mu e^{-\mu T}]$$

$$= \frac{d}{dT}R^q_m(T) \frac{qK}{(m+1)!}\left[\prod_{j=1}^{m}(j+qK)\right]e^{-q\lambda T}(1-e^{-\mu T})^m$$

$$[q\lambda(1-e^{-\mu T}) - (m+1)\mu e^{-\mu T}]$$

$$= \frac{-q\lambda e^{-q\lambda t}}{(m+1)!}\left[\prod_{j=1}^{m}(j+qK)\right](1-e^{-\mu T})^m$$

$$[m+1+qK(1-e^{-\mu T})-(m+1)e^{-\mu T}]$$

which reduces to the form given in equation 2. 3) .

For further use consider the following recursive relationships.

2.5)
$$\frac{d^2}{dT^2} R^q_m(T) = q^2\lambda^2 e^{-q\lambda T} \frac{(1-e^{-\mu T})^m}{m!} \prod_{j=1}^{m}(j+qK)$$
$$-q\lambda\mu e^{-q\lambda T} \frac{(1-e^{-\mu T})^{m-1}}{(m-1)!} \prod_{j=1}^{m} (k+qK)$$
$$= -q\lambda \frac{d}{dt} R^q_m(T) + \mu(m+qK)\frac{d}{dt} R^q_{m-1}(T)$$

for $m-1 \geq 0$.

In general, now,

2.6)
$$\frac{d^n}{dT^n} R^q_m(T) = (-1)^{n-1}\lambda^{n-1}\frac{d}{dT} R^q_m(T)$$
$$+ \sum_{j=1}^{n-1} (-1)^{n-1-j}\binom{n-1}{j}\lambda^{n-1}G^j\left(\frac{d}{dT} R^q_{m-j}(T)\right) \prod_{i=0}^{j-1}(m-i+qK)$$

for $1 \leq n \leq m+1$.

Using induction again, it is true for $n = 2$. For the induction step,

$$\frac{d^{n+1}}{dT^{n+1}} R^q_m(T) = (-1)^{n-1}\lambda^{n-1}\frac{d^2}{dT^2} R^q_m + \sum_{j=1}^{n-1}(-1)^{n-1-j}\binom{n-1}{j}\lambda^{n-1}G^j$$
$$\left(\frac{d^2}{dT^2} R^q_{m-j}(T)\right)\prod_{1=0}^{j-1}(m-i+q)$$

Substituting for 2.5) and collecting terms, equation 2.6) can be seen to be true.

From equation 2.1), $\frac{d^j}{dT^j} R^q(T)\,|_{T=0} = (-q\lambda)^j$, and $R^q_m(0) = 1.$

From equation 2.3), $\frac{d}{dT} R^q_m(T)\,|_{T=0} = 0$ for $m \geq 1$

From equation 2.6),

2.7) $$\frac{d^n}{dT^n} R^q_m(T)\,|_{T=0} = 0 \quad \text{for } n \leq m$$

2.8) $$\frac{d^{m+1}}{dT^{m+1}} R^q_m(T)\,|_{T=0} = -\binom{m}{m} \lambda^m q G^m \prod_{i=0}^{m-1} (m-i+qK)$$
$$= -q^{m+1} \lambda^{m+1} \prod_{i=1}^{m} (1 + \frac{i}{q} G)$$

Further derivations may be obtained by writing

$$\frac{d^{m+1}}{dT^{m+1}} R^q_m(T) = (-1)^m \lambda^m \frac{d}{dT} R^q_m(T)$$
$$+ \sum_{j=1}^{m} (-1)^{m-j} \binom{m}{j} \lambda^m G^j (\frac{d}{dT} R^q_{m-j}(T)) \prod_{i=0}^{j-1} (m-i+qK)$$

and differentiating both sides of this expression.

2.9)
$$\frac{d^{m+x}}{dT^{m+1+x}} R^q_m(T) = (-1)^m \lambda^m \frac{d^x}{dT^x} R^q_m(T)$$

$$+ \sum_{j=1}^{m} (-1)^{m-j} \binom{m}{j} \lambda^m G^j \left(\frac{d^x}{dT^x} R^q_{m-j}(T)\right) \prod_{i=0}^{j-1} (m-i+qk);$$

2.10)
$$\frac{d^{m+2}}{dT^{m+2}} R^q_m(T)\Big|_{T=0} = (-1)\binom{m}{m-1} \lambda^m G^{m-1} \left(\frac{d^2}{dT^2} R^q_1(T)\Big|_{T=0}\right) \prod_{i=0}^{m-2} (m-i+qK)$$

$$+ \binom{m}{m} \lambda^m G^m \left(\frac{d^2}{dT^2} R^q_0(T)\pi_{T=1}\right) \prod_{i=0}^{m-1} (m-i+qK)$$

$$= \lambda^{m+2} q^{m+2} \left(1+\frac{m}{q}\right) \prod_{i=1}^{m} \left(1+\frac{i}{q} G\right)$$

Using Mc Claurin's series,

$$R^q_m(T) = \sum_{n=0}^{\infty} \left(\frac{d^2}{dT^2} R^q_m(0)\right) \frac{T^n}{n!}$$

and substituting from equations 2.7) and 2.8),

2.11)
$$R^q_m(T) \approx 1 - \frac{(q\lambda T)^{m+1}}{(m+1)!} \prod_{i=1}^{m} \left(1 + \frac{i}{q} G\right) \quad \text{for } m \geq 1$$

while equation 2.10) shows that the remainder term is positive.

2.2.3 Formulas for ${}_cR^q_m$

If the effect of error coverage, c, is inserted into this formulation, the basic equation becomes

2.12) $${}_cR^q_0(T) = e^{-q\lambda T}$$

and equation 2.2 becomes

2.13) $${}_cR^q_m(T) = {}_cR^q_{m-1}(T) + \int_0^T c\, e^{-\mu t} e^{-\lambda(T-t)q} \frac{d}{dt}[1 - {}_cR^q_{m-1}(t)]dt.$$

The most concise form for stating the general solution is in terms of $R^q_i(T)$:

2.14) $${}_cR^q_m = \sum_{i=0}^{m} \binom{m}{i} c^i(1-c)^{m-i} R^q_i$$

Equation 2.14 is again proved by induction. If m=1, substituting in equation 2.13 and using equation 2.1 gives

$${}_cR^q_1(T) = e^{-q\lambda T} + \int_0^T c\, e^{-\mu t} e^{-q\lambda(T-e)} q\lambda e^{-q\lambda t}\, dt$$

$$= e^{-q\lambda T}[1+cKq(1-e^{-\mu T})]$$

$$= (1-c)e^{-q\lambda T} + c[e^{-q\lambda T} + qK(1-e^{-\mu T})e^{-q\lambda T}]$$

$$= (1-c)\ R^q_0 + c\ \ R^q_1$$

For the induction step, assume formula 2.14 for m=n-1 and differentiate both sides.

2.15) $$\frac{d}{dt}\; {}_cR^q_{n-1}(T) = \sum_{i=0}^{n-1} \binom{n-1}{i} c^i (i-c)^{n-1-i} \frac{d}{dT}\; R^q_i$$

Substituting this in equation 2.13,

$$ {}_cR^q_n(T) = \sum_{i=0}^{n-1} {}_cR^q_{m-1}\quad(T)$$
$$- c \sum_{i=0}^{n-1} \binom{n-1}{i} c^i (1-c)^{n-1-i} \int_0^T \frac{d}{dT} R^q_i\, e^{-\mu t} e^{-\lambda q(T-e)} dt$$

Substituting from equation 2.2), and using the induction step

$$ {}_cR^q_n(T) = \sum_{i=0}^{n-1} \binom{n-1}{i} c^i (1-c)^{n-1-i} \quad R^q_i$$

$$+ c \sum_{i=0}^{n-1} \binom{n-1}{i} c^i (1-c)^{n-1-i} [R^q_{i+1} - R^q_i]$$

and adding the terms together finishes the proof.

To get an approximation for small λT, R^q_i must be expanded in powers of $(\lambda T)^y$ for $i + 1 \leq y \leq m+1$ about $T = 0$.

Now, $R_0^q = e^{-q\lambda T}$, $\frac{d^j}{dT^j} e^{-q\lambda T} = (-q\lambda)^j e^{-q\lambda T}$

from equation 9,

$$\frac{d^n}{dT^2} R_1^q(T) = -\lambda \frac{d^{n-1}}{dT^{n-1}} R_1^q(T) + \lambda G(1+qK) \frac{d^{n-1}}{dT^{n-1}} R_0^q(T) \text{ for } n > 1$$

and so

$$\frac{d^2}{dT^2} R_1^q(0) = -q^2\lambda^2(1+\frac{1}{q} G)$$

$$\frac{d^3}{dT^3} R_1^q(0) = q^3\lambda^3(1+\frac{G}{q})(1+\frac{1}{q})$$

$$\frac{d^4}{dT^4} R_1^q(0) = -q^4\lambda^4(1+\frac{1}{q}G)(1+\frac{1}{q}+\frac{1}{q^2})$$

$$\frac{d^n}{dT^n} R_1^q(0) = (-1)^{n-1}q^n\lambda^n(1+\frac{1}{q}G)(1+\frac{1}{q}+\frac{1}{q^2}+\dots+\frac{1}{q^{n-2}})$$

Using equation 2.9 again,

$$\frac{d^n}{dT^n} R_2^q = \lambda^2 \frac{d^{n-2}}{dT^{n-2}} R_2^q(T) - 2\lambda^2 G(2+qK) \frac{d^{n-2}}{dT^{n-2}} R_1^q(T)$$

$$+ \lambda^2 G^2(2+qK)(1+qK) \frac{d^{n-2}}{dT^{n-2}} R_0^q(T) \text{ for } n > 2 .$$

By direct differentiation

$$\frac{d^3}{dT^3} R_2^q(0) \; -q^3\lambda^3 \prod_{i=1}^{2} (1+\frac{i}{q} G)$$

$$\frac{d^4}{dT^4} R_2^q(0) = q^4\lambda^4 (\prod_{i=1}^{2} (1+\frac{i}{q} G))(1+\frac{2}{q})$$

$$\frac{d^5}{dT^5} R_2^q(0) = -q^5 \lambda^5 (\prod_{i=1}^{2} (1+\frac{i}{q} G))(1+\frac{2}{q} + \frac{3}{q^2})$$

$$\frac{d^6}{dT^6} R_2^q(0) = q^6\lambda^6 (\prod_{i=1}^{2} (1+\frac{i}{q} G))(1+\frac{2}{q} + \frac{3}{q^2} + \frac{4}{q^3})$$

Using equation 2.9 again,

$$\frac{d^n}{dT^n} R_3^q = -\lambda^3 \frac{d^{n-3}}{dT^{n-3}} R_3^q(T) + 3\lambda^3 G(3+qK) \frac{d^{n-3}}{dT^{n-3}} R_2^q(T)$$

$$-3\lambda^3 G^2 (3+qK)(2+qK) \frac{d^{n-3}}{dT^{n-3}} R_1^q(T)$$

$$+ \lambda^3 G^3 (3+qK)(2+qK)(1+qK) \frac{d^{n-3}}{dT^{n-3}} R_0^q(T)$$

for $n > 3$.

$$\frac{d^4}{dt^4} R_3^q(0) = -q^4\lambda^4 \prod_{i=1}^{3} (1+\frac{i}{q}G)$$

$$\frac{d^5}{dT^5} R_3^q(0) = q^5\lambda^5 (\prod_{i=1}^{3} (1+\frac{i}{q}G))(1+\frac{3}{q})$$

$$\frac{d^6}{dT^6} R_3^q(0) = -q^6\lambda^6 \prod_{i=1}^{3} (1+\frac{i}{q}G)(1+\frac{3}{q}+\frac{6}{q^2})$$

Using these derivations, equation 2.14 may be expanded for different small orders of m.

2.16) $$ {}_cR_1^q = 1 - (1-c)q\lambda T + (1-c-c(1+\frac{1}{q}G))\frac{(q\lambda T)^2}{2} - (1-c-c(1+\frac{1}{q}G)(1+\frac{1}{q}))\frac{(q\lambda T)^3}{6} + \dots$$
$$= 1 - b_{11}\, q\lambda T - b_{12}(q\lambda T)^2 + b_{13}(q\lambda T)^3 + \dots$$

2.17) $$ {}_cR_2^q = 1 - (1-c)^2 q\lambda T + (1-c)(1-c-2c(1+\frac{1}{q}G))\frac{(q\lambda T)^2}{2}$$
$$- [(1-c)^2 - 2c(1-c)(1+\frac{1}{q}G)(1+\frac{1}{q}) + c^2(1+\frac{1}{q}G)(1+\frac{2}{q}G)]\frac{(q\lambda T)^3}{6}$$
$$+ [(1-c)^2 - 2c(1-c)(1+\frac{1}{q}G)(1+\frac{1}{q}+\frac{1}{q^2})$$
$$+ c^2(1+\frac{1}{q}G)(1+\frac{2}{q}G)(1+\frac{2}{q})]\frac{(q\lambda T)^4}{24} + \dots$$
$$= 1 - b_{21}\, q\lambda T - b_{22}(q\lambda T)^2 - b_{23}(q\lambda T)^3 + b_{24}(q\lambda T)^4 + \dots$$

$$2.18)\quad {}_{c}R_{3}^{q} = 1 - (1-c)^{3}\, q\lambda T + (1-c)^{2}(1-c-3c(1+\frac{1}{q}G))\, \frac{q\lambda T)^{2}}{2}$$

$$-(1-c)[(1-c)^{2} - 3c(1-c)(1+\frac{1}{q}G)(1+\frac{1}{q}) + 3c^{2}(1+\frac{1}{q}G)(1+\frac{2}{q}G)\frac{(q\lambda T)^{3}}{6}$$

$$+\ [(1-c)^{3} - 3(1-c)^{2}(1+\frac{1}{q}G)(1+\frac{1}{q}+\frac{1}{q^{2}}) + 3c^{2}(1-c)(1+\frac{1}{q}G)(1+\frac{2}{q}G)(1+\frac{2}{q})$$

$$-\ c^{3}(1+\frac{1}{q}G)(1+\frac{2}{q}G)(1+\frac{3}{q}G)\,]\frac{(q\lambda T)^{4}}{24}$$

$$-\ [(1-c)^{3} - 3c(1-c)^{2}(1+\frac{1}{q}G)(1+\frac{1}{q}+\frac{1}{q^{2}}+\frac{1}{q^{3}})$$

$$+\ 3c^{2}(1-c)(1+\frac{1}{q}G)(1+\frac{2}{q}G)(1+\frac{3}{q}G)(1+\frac{2}{q}+\frac{3}{q^{2}})$$

$$-\ c^{3}(1+\frac{1}{q}G)(1+\frac{2}{q}G)(1+\frac{3}{q}G)\quad]\frac{(q\lambda T)^{5}}{5!} \quad + \ldots..$$

$$= 1 - l_{31}\, q\lambda T - l_{32}(q\lambda T)^{2} - b_{33}(q\lambda T)^{3} - b_{34}(q\lambda T)^{4} + b_{35}(q\lambda T)^{5} + \ldots$$

2.2.4 Solution of the Equation $\frac{ds}{dt} - TS$

2.2.4.1 General Form

The problems of determining T and initial conditions will be solved after the form for the solution of the differential equation is determined. This solution will be in a form convenient for computation, based upon reference [CL, 55, P. 75]

Let $J^i_j = \lambda_j E_{r_j} + a Z_{r_j}$ where r_j is the dimension, E_{r_j} is a unit matrix and Z_{r_j} is a matrix with $\delta_{i,i+1} = 1$, other elements 0. It is well known that there exists a non-singular matrix P such that

$$T = P^{-1} J^i_j P$$

where the λ_j's are the characteristic roots of T and r_j depends upon T's invarient factors [v. d. Waerden, 40]
The solution S(t) is given by $PS(t) = e^{tJ} PS(0)$
where

$$e^{tJ_i} = e^{t\lambda_i} \begin{pmatrix} 1 & at & & \cdots & \frac{(at)^{r-1}}{(r-1)!} \\ 0 & 1 & at & \cdots & \frac{(at)^{r-2}}{(r-2)!} \\ 0 & 0 & 1 \;\; at & \cdots & \frac{(at)^{r-3}}{(r-3)!} \\ \cdot & & & \ddots & \vdots \\ \cdot & & & 1 & at \\ \cdot & & & & \\ 0 & \cdots & & & 1 \end{pmatrix}$$

To determine the general form for T, reconsider the definitions made previously. Begin with an assemblage of n modules, each capable of correct operation until $f + 1$ malfunctions occur, with the requirement q modules must be correctly operating for the assemblage to function correctly. In this case, $m = n - q$ modules are spares. If the assemblage has had exactly j errors, and the assemblage will still operate correctly, call this state S_j. Each state may be represented by the 'error list' $(a_1, a_2, \ldots, a_n)$ where the ith entry gives the number of failures existing in the ith module. Since $\mu = 0$, a $-$ indicates that the module is a spare (they never fail). Let F represent the failed state.

Initially there are no malfunctions. Any active unit is equally likely to malfunction, and the probability that a module failure occurs in the time internal h is λh.

From the definition of S_j, it is clear that, in the interval h, S_j may map into S_j, S_{j+1}, or into F. S_j can map into F only if $j \geq (f+1)m+f$. If $\rho = (f+1)m+fq$, states S_j in which the system continues to operate exist until $j = \rho$.

If $1 \leq j \leq \rho+1$, let p_j be the probability that in the event of a malfunction a module in the assemblage in state $S_{\rho+1-j}$ will cease correct operation. The probability of module malfunction is λh, and there are q operating independent modules. Thus the probability that state S_{p+1-j} is mapped into F is $p_j q \lambda h$.

Write the state vector S as

$$S = \begin{pmatrix} F \\ S\rho \\ S\rho-1 \\ S\rho-2 \\ \vdots \\ S_1 \\ S_0 \end{pmatrix}$$

The general form of the matrix T is:

$$T = \begin{pmatrix} 0 & p_1 q\lambda & p_2 q\lambda \cdots\cdots & P_{\rho+1} q\lambda \\ 0 & & & \\ \cdot & & & \\ \cdot & & R & \\ \cdot & & & \\ 0 & & & \end{pmatrix}$$

where $R = -q\lambda E_{p+1} + (1-p_j)q\lambda\,\delta_{j,\,j+1}$

where $2 < j < \rho+1$, and $\delta_{j,\,j+1} = 1$.

Since if there have been ρ malfunctions any additional one will cause the assemblage to fail, $p_1 = 1$.

The smallest number of malfunctions which cause the assemblage to cease correct operation is

$(f+1)(m+1)$. Since $\rho+1-((f+1)(mE1)-1) = f(q-1)+1$, $p_j \neq 0$, when $1 \leq j \leq f(q-1)+1$ and $p_j = 0$ otherwise.

The solution of the differential equations has

$$J = \begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & & & & & \\ \cdot & & & & & \\ \cdot & \cdot & & & & \\ \cdot & -q\lambda E_{\rho+1} + q\lambda Z_{\rho+1} & & & & \\ 0 & & & & & \end{array}$$

The matrix P so that PT = JP can be found by assuming that F has 1's in the first row and otherwise the only non-zero elements are on the main diagonal and these are clearly 1's for $j \geq f(q-1) + 1$. By multiplication and solution of the two equations for each row,

$$P = \left(\begin{array}{cccccccc}
1 & 1 & 1 & 1 & . & . & . & 1 \\
0 & a_1 & 0 & & & & & 0 \\
0 & 0 & a_2 & & & & & \\
. & & & a_{f(q-1)} & & & & \\
. & & & & & 1 & & \\
. & & & & & & & 0 \\
0 & & & & & & 0 & 1
\end{array} \right)$$

Where $a_j = \dfrac{1}{\prod\limits_{i=j}^{f(q-1)} (1 - p_{i+1})}$

and the p_i's are given by T .

Using the previous notation of S' for S transpose,

$$(PS(t))' = \left(F + \sum_{j=0}^{\rho} S_j, \; a_1 S_\rho, a_2 S_{\rho-1}, \ldots, a_j S_{\rho-j+1}, \ldots, S_0\right)$$

The initial condition (assuming no malfunctions) is

$$S'(0) = (0, 0, 0, \ldots, 0, 1)$$

$$(PS(0))' = (1, 0, 0, \ldots, 0, 1)$$

$$e^{tJ} = e^{-q\lambda t} \begin{pmatrix} e^{q\lambda t} & 0 & 0 & & \cdots & 0 \\ 0 & 1 & q\lambda t & & \cdots & \frac{(q\lambda t)^{\rho}}{(\rho)!} \\ 0 & 0 & 1 & q\lambda t & \cdots & \frac{(q\lambda t)^{\rho-1}}{(\rho-1)!} \\ \vdots & & & & & \vdots \\ & & & & 1 & q\lambda t \\ 0 & & \cdots & & 0 & 1 \end{pmatrix}$$

$$[e^{tJ}\, PS(0)]' = e^{-q\lambda t}[e^{q\lambda t}, \frac{(q\lambda t)^{\rho}}{\rho!}, \frac{(q\lambda t)^{\rho-1}}{(\rho-1)!}, \ldots, q\lambda t, 1]$$

So,

$$S_{\rho-j+1} = e^{-q\lambda t}[\prod_{i=j}^{f(q-1)} (1-p_{i+1})] \frac{(q\lambda t)^{f+1-j}}{(\rho+1-j)!} \qquad 1 \leq j \leq f(q-1)$$

$$S_{\rho-j+1} = e^{-q\lambda t} \frac{(q\lambda t)^{\rho+1-j}}{(\rho+1-j)!} \qquad f(q-1)+1 \leq j \leq \rho+1$$

$${}^{f}R_m^q(\lambda, 0) = \sum_{j=1}^{\rho+1} S_{\rho+1-j} = e^{-q\lambda t}[\sum_{i=0}^{(f+1)m+f} \frac{(q\lambda t)^i}{i!}$$

$$+ \sum_{i=(f+1)(m+1)}^{\rho} [\prod_{k=\rho+1-i}^{f(q-1)} (1-p_{k+1})] \frac{(q\lambda t)^i}{i!}]$$

This may be written as

$$ {}^{f}R_{m}^{q}(\lambda, 0) = 1 - \frac{(\lambda qt)^{(f+1)(m+1)}}{[(f+1)(m+1)]!} \sum_{j=0}^{\infty} \frac{(f+1)(m+1)}{(f+1)(m+1)+j} \frac{(-q\lambda t)^j}{j!} $$

$$ + e^{-q\lambda t} \sum_{j=(f+1)(m+1)}^{\rho} \left(\prod_{k=\rho+1-j}^{f(q-1)} (1-p_{k+1}) \right) \frac{(q\lambda t)^j}{j!} $$

Since $e^{-q\lambda t} < 1$, if this term is not expanded, then the series representing ${}^{f}R_{m}^{q}(\lambda, 0)$ has alternating signs, so

$$ A < {}^{f}R_{m}^{q}(\lambda, 0) < B $$

$$ A = 1 - \frac{(q\lambda t)^{(f+1)(m+1)}}{((f+1)(m+1))!} [1 - e^{-q\lambda t}(1-p_{f(q-1)+1})] $$

$$ B = A + \frac{(q\lambda t)^{(f+1)(m+1)+1}}{((f+1)(m+1))!} \left[\frac{(f+1)(m+1)}{(f+1)(m+1)+1} + \frac{(1-p_{f(2-1)+1})(1-p_{f(q-1)})e^{-q\lambda t}}{(f+1)(m+1)+1} \right] $$

For small λt,

$$ {}^{f}R_{m}^{q}(\lambda, 0) \approx 1 - p_{f(q-1)+1} \frac{(q\lambda t)^{(f+1)(m+1)}}{[(f+1)(m+1)]!} > A $$

2.2.4.2 Determination of the Probabilities p_k.

2.2.4.2.1 Introduction

The probabilities p_k depend upon the state $S_{\rho-j}$, the 'error lists' or configurations for the state $S_{\rho-j}$, and the number of paths by which $S_{\rho-j}$ may be reached from the initial configuration.

There are two feasible sets of formulas which derived. The first is to set $f = 1$ and derive formulas for small values of q, say 1, 2, 3, 4, and 5, and arbitrary values of m. The second is to fix m at small values, 1, 2, and 3, and derive approximations for small values of f(say 1 or 2) and arbitrary values of q. To derive these formulas, general forms will be derived for the p_j in the matrix T. To determine these coefficients, in the case of fixed q and arbitrary values of m, difference equations will be derived to determine the number of paths to each configurations in each state S_j until configuration F may be reached. The probabilities of transfer to F can then be determined. These probabilities determine the coefficients in the expression for ${}^{f}R_m^q(\lambda, 0)$. In the case of arbitrary q and fixed values of m, the values of p_j for small j will be determined by beginning with the states F and S_ρ and progressing through the states $S_{\rho j}$.

2.2.4.2.2 Example 1

Begin with fixed q as stated above, the p_j are determined by a consideration of the configurations of active, spare and discarded units in the state $S_{\rho-j}$. A simple but non-trivial example will show the problem.

Let $f = 1$, $q = 3$, m be arbitrary. To get the feel, do a few special cases.

m = 1	State	Error List*	Number of Paths
	S_0	0, 0, 0, -	1
	S_1	1, 0, 0, -	3
	S_2	1, 1, 0, -	6
		2, 0, 0, 0	3
	S_3	1, 1, 1, -	6
		2, 1, 0, 0	21
	S_4	2, 1, 1, 0	60
		F	21
	S_5	2, 1, 1, 1	60
		F	120

* Again the $i^{\underline{th}}$ entry in the list gives the total number of failures in the $i^{\underline{th}}$ module and <u>members</u> means the number of distinct paths or ways to get to the given S_j from the initial state S_0.

m = 2	State	Error List	Number of Paths
	S_0	0, 0, 0, -, -	1
	S_1	1, 0, 0, -, -	3
	S_2	1, 1, 0, -, -	6
		2, 0, 0, 0, -	3
	S_3	1, 1, 1, -, -	6
		2, 1, 0, 0, -	21
	S_4	2, 1, 1, 0, -	60
		2, 2, 0, 0, 0	21
	S_5	2, 1, 1, 1, -	60
		2, 2, 1, 0, 0	183
	S_6	2, 2, 1, 1, 0	546
		F	183
	S_7	2, 2, 1, 1, 1	546
		F	1092
	F		

The structure of the error list is now fairly clear. As m increases S_0-S_5 remain unchanged.

m = 3	State	Error List	Number of Paths
	S_6	2, 2, 2, 0, 0, 0	183
		2, 2, 1, 1, 0, -	546
	S_7	2, 2, 2, 1, 0, 0	1641
		2, 2, 1, 1, 1, -	546

State	Error List	Number of Paths
S_8	2, 2, 2, 1, 1, 0	4920
	F	1641
S_9	2, 2, 2, 1, 1, 1	4920

The possible last list endings for S_{2m+1} are 1, 0, 0 and 1, 1, 1, - . If G(n) is the number of lists ending 1, 0, 0 then H(n) = 3n - G(n) is the number of lists ending with 1, 1, 1, - .

The probabilities of next state transfer are:

$$S_j \begin{array}{l} \xrightarrow{1 - 3\lambda h} S_j \\ \\ \xrightarrow{3\lambda h} S_{j+1} \end{array} \qquad 0 \le j \le 2m$$

$$S_{2m+1} \begin{array}{l} \xrightarrow{1 - 3\lambda h} S_{2m+1} \\ \\ \xrightarrow{\frac{2G(2m+1) + 3H(2m+1)}{3^{2m+2}} \cdot 3\lambda h} S_{2m+2} \\ \\ \xrightarrow{\frac{G(2m+1)}{3^{2m+2}} \cdot 3\lambda h} F \end{array}$$

S_{2m+1} has error lists ending in 1, 1, 0. The probabilities of next state transfer are:

$$\begin{tikzcd} & S_{2m+2} \\ S_{2m+2} \arrow[ru, "1 - 3\lambda h"] \arrow[r, "\lambda h"] \arrow[rd, "2\lambda h"'] & S_{2m+3} \\ & F \end{tikzcd}$$

- - - - - - - - - - - - - - -

$$\begin{tikzcd} & S_{2m+3} \\ S_{2m+3} \arrow[ru, "1 - 3\lambda h"] \arrow[rd, "3\lambda h"'] & \\ & F \end{tikzcd}$$

The largest number of members in each state configuration after $n = 1$ can be seen to satisfy the recurrence relation

$$G(n+2) = 2G(n+1) + 3\,G(n)$$

This corresponds to the difference equation

$$(E^2 - 2\,E - 3)\,G(n) = 0$$

if $G(n+1) = E\,G(n)$. The equation has solution

$$G(n) = 3/4\,[3^n - (-1)^n] \quad (n \neq 0)$$

$$\frac{G(2m+1)}{3^{2m+2}} \,.\, 3\lambda h = 3/4\,(1 + \frac{1}{3^{2m+1}})\,\lambda h$$

$$\frac{2G(2m+1) + 3H(2m+1)}{3^{2m+2}} \,.\, 3\lambda h = 9/4\,(1 - \frac{1}{3^{2m+2}})\,\lambda h$$

$$T = \begin{pmatrix} 0 & 3\lambda & 2\lambda & 3/4(1+\frac{1}{3^{2m+1}}) & 0\ldots & 0 \\ 0 & -3\lambda & \lambda & 0 & 0\ldots & 0 \\ 0 & 0 & -3\lambda & 9/4(1-\frac{1}{3^{2m+2}}) & 0\ldots & 0 \\ 0 & 0 & 0 & -3\lambda & 3\lambda & \cdot \\ \cdot & \cdot & \cdot & & & \cdot \\ \cdot & \cdot & \cdot & & -3\lambda & 3\lambda \\ 0 & \cdot & \cdot\ \cdot & & 0 & -3\lambda \end{pmatrix}$$

Enough examples have been given to motivate a more general approach to handling error lists. Error lists can be divided into three parts - the part called 'retired' which corresponds to inoperable units, the q active units, and the as yet unused spares. Methods of handling the number of paths will be derived.

2.2.4.2.2 <u>Case 1</u>

$f = 1$, $q = 2q'$ (i.e. q even) no error lists <u>mapping into F</u>.

State	Error List				Number of Paths
	Number Retired (NR)	Active Configurations (AC)		Number Spare (NS)	(NP)
		# 1's	# 0's		
S_{2i+1}	i	1	$q-1$	$m-i$	N_0
	$i-1$	3	$q-3$	$m-i+1$	N_1
	$i-2$	5	$q-5$	$m-i+2$	N_2
	$i-j$	$2j+1$	$q-2j+1$	$m-i+j$	N_j
	$i-q'+1$	$q-1$	1	$m-i+q'-1$	$N_{q'-1}$

State	NR	#1's	#0's	NS	NP
S_{2i+2}	$i+1$	0	q	$m-i-1$	$N_0 = N^1_{-1}$
	i	2	$q-2$	$m-i$	$(q-1)\ N_0 + 3N_1 = N^1_0$
	$i-1$	4	$q-4$	$m-1+1$	$(q-3)\ N_1 + 5N_2 = N^1_1$
		. . .			
	$i-j$	$2j+2$	$q-2j-2$	$m-i+j$	$(q-2j-1)N_j + (2j+3)N_{j+1} = N'_j \ldots 0$
		. . .			
	$i-q'+1$	q	0	$m-i+q'-1$	$N_{q'-1} = N'_{q'-1}$
S_{2i+3}	$i+1$	1	$q-1$	$m-i-1$	$qN'_{-1} + 2N'_0$
	i	3	$q-3$	$m-i$	$(q-2)\ N'_0 + 4N'_1$
	$i-1$	5	$q-5$	$m-i+1$	$(q-4)N'_1 + 6N'_2$
		. . .			
	$i-j+1$	$2j+1$	$q-2j-1$	$m-i+j-1$	$(q-2j)\ N'_{j-1} + (2j+2)N'_j$
		. . .			
	$i-q'+2$	$q-1$	1	$m-i+q'-2$	$2N'_{q'-2} + q\ N'_{q'-1}$

Since the error lists in S_{2i+1} and S_{2i+3} are the same, using the next state operator E recurrence relations can be set up in the numbers N_0 to $N_{q'-1}$.

If $N_i\,(2j+3) = E\,N_i(2j+1)$, then

$$0 = \begin{pmatrix} -E+3q-2 & 2\times 3 & 0 & 0 & 0 \\ (q-1)(q-2) & -E+7q-18 & 4\times 5 & 0 & 0 \\ 0 & (q-3)(q-4) & -E+11q-50 & 6\times 7 & \vdots \\ & & \cdots & & \\ 0 & & & 3\times 2 & -E+3q-2 \end{pmatrix} \begin{pmatrix} N_0 \\ N_1 \\ \vdots \\ N_{q'-1} \end{pmatrix}$$

where the jth now is .

$$0\ldots0 \qquad (q-2j)(q-2j+1) \quad -E+(4j+3)\,q-2(2j+1)^2 \quad (2j+2)(2j+3) \quad 0\ldots0$$

The determinant must be zero, giving a difference equation for the N_i which can be solved as before.

2.2.4.2.3

<u>Case 2</u> <u>$f=1$, $q=2q'+1$ (i.e. q odd) no error lists mapping into F.</u>

State	NR	#1's	#0's	NS	NP
S_{2i+1}	i	1	$q-1$	$m-1$	N_0
	$i-1$	3	$q-3$	$m-i+1$	N_1
			$\cdots$		
	$i-j$	$2j+1$	$q-2j-1$	$m-i+j$	N_j
			$\cdots$		
	$i-q'$	q	0	$m-i+q$	$N_{q'}$
S_{2i+2}	$i+1$	0	q	$m-i-1$	$N_o=N'_{-1}$
	i	2	$q-2$	$m-i$	$N'_1(q-1)N_o+3N_1$
	$i-j$	2_j+2	$q-2_j-2$	$m-i+j$	$N'_j=(q-2j-1)N_j$ $+(2j+3)N_{j+1}$
	$i-q'+1$	$q-1$	1	$m-i+q'$	$N'_{q'-1}=2N_{q'-1}+qN'_{q'}$
S_{2i+3}	$i+1$	1	$q-1$	$m-i-1$	$qN''_{-1}+2N''_0$
	i	3	$q-3$	$m-i$	$(q-2)N''_0+4N''_1$
			$\cdots$		
	$i-j$	2_j+1	$q-2_j-1$	$m-i-j-1$	$(q-2^1_j)N''_{j-1}+(2_j+2)N''_j$
	$i-q'+1$	$q-1$	1	$m-i+q'-1$	$N''_{q''-1}$

As before,

$$\begin{pmatrix} -E+3q-2 & 2\times 3 & 0 & 0 & 0 \\ (q-1)(q-2) & -E+7q-18 & 4\times 5 & 0 & 0 \\ 0 & (q-3)(q-4) & -E+11q-50 & 6\times 7 & 1 \\ \cdot & & \cdots & & \\ \cdot & & & & \\ \cdot & & & & \\ 0 & & & 0 & 2-E+q \end{pmatrix} \begin{pmatrix} N_0 \\ N_1 \\ \cdot \\ \cdot \\ \cdot \\ N_{q'} \end{pmatrix} = 0$$

where the $j^{\underline{th}}$ now is

$$0\ldots 0\ (q-2j)(q-2j+1) \qquad -E+(4j+3)q-2(2j-1)^2\ (2j+2)(2j+3)\ \ 0\ldots 0$$

The determinant must be zero, giving a difference equation for the N_i which can be solved as before.

2.2.4.2.4 Now p_K can be determined, if the mappings into F are derived.

Case 3 $f=1$, An error list maps into F.

For states S_j, $2M+1 \leq j \leq 2m+q$, one error list will map into F. The number retained in successive steps will be best found by proceeding backwards from S_{2m+q} to S_{2m+1}. No general formula is easily available, so the steps will be carried out in special cases.

State	NS	#1's	#0's	NR	NP	Number Transfers (to next step) NT
S_{2m+q}	0	q	0	m	$N_o(2m+q)$	$N_0(2m+q)$
S_{2m+q-1}	0	q-1	1	m	$N_0(2m+q-1)$	$(q-1)N_0(2m+q-1)$
S_{2m+q-2}	0	q-2	2	m	$N_0(2m+q-2)$	$(q-2)N_0(2m+q-2)$
	1	q	0	m-1	$N_1(2m+q-2)$	
S_{2m+q-3}	0	q-3	3	m	$N_o(2m+q-3)$	$(q-3)N_0(2m+q-3)$
	1	q-1	1	m-1	$N_1(2_m+q-3)$	
S_{2m+q-4}	0	q-4	4	m	$N_0(2m+q-4)$	$(q-4)N_0(2m+q-4)$
	1	q-2	2	m-1	$N_1(2m+q-4)$	
	2	q	0	m-2	$N_2(2m+q-4)$	
			. . .			
$S_{2m+q-2i}$	0	q-2i	2i	m	$N_0=(2i+1)N_0'$ $+(q-2i+1)N_i$	$(q-2i)N_0$
	1	q-2i+2	2i-2	m-1	$N_1=(2i-1)N_1'$ $+(q-2i+3)N_2'$	
			. . .			
	j	q-2i+2j	2i-2j	m-j	$N_j=(2i-2j+1)N_j'$ $+(q-2i+2j+1)N_{j+1}'$	
			. . .			
	i	q	0	m-i	$N_i = N_i'$	

State	NS	#1's	#0's	NR	NP	Number Transfers (to next step) NT
$S_{2m+q-2i-1}$	0	q-2i-1	2i+1	m	$N'_0=(2i+2)N''_0 +(q-2i)N''_1$	$(q-2i-1)\,N'_0$
	1	q-2i+1	2i-1	m-1	$N'_1 = 2iN''_1 +(q-2i+2)N''_2$	
			. . .			
	j	q-2i+2j-1	2i-2j+1	m-j	$N'_j=(2i-2j+2)N''_j +(q-2i+2j)N''_{j+1}$	
			. . .			
	i	q-1	1	m-i	$N'_i = 2N''_i + q\,N''_{i+1}$	
$S_{2m+q-2i-2}$	0	q-2i-2	2i+2	m	N''_0	$(q-2i-2)N''_0$
	1	q-2i	2i	m-1	N''_1	
			. . .			
	j	q-2i+2j-2	2i-2j+2	m-j	N''_j	
			. . .			
	i+1	q	0	m-i-1	N''_{i+1}	

State	NS	#1's	#0's	NR	NP	Number Transfers (to next step) NT
S_{2m+1}	0	1	q-1	m	N_0^*	N_0^*
	1	3	q-3	m-1	N_1^*	
	2	5	q-5	m-2	N_2^*	
			. . .			
	i	2i+1	q-2i-1	m-i	N_i^*	
			. . .			

The exact configuration in the last step depends upon whether q is odd or even.

2.2.5 Specific Solutions of $d\mathcal{S}/dt = TS$

2.2.5.1 Specific Examples, q small, m arbitrary, f=1.

First the previous example will be redone, using the general solution methods, then more special cases will be solved.

Clearly, S_i is transformed into S_{j+1} or S_i, with probabilities λh and $1 - \lambda h$ respectively. So

$$p_1 = 1 \quad \text{and} \quad {}^1R_m^1(\lambda, 0) = e^{-\lambda T} \sum_{i=0}^{2m+1} \frac{(\lambda T)^i}{i!}$$

Example 2 q = 2.

$$q' = 1 \quad (-E+4)\, N_0(2i+1) = 0$$

$$N_0(2i+1) = a \bullet 4^i$$

when $i = 0$, $N_0(1) = 2 = a$, so $N_0(2i+1) = 2^{2i+1}$

For the transformations to F,

	NS	#1's	#0's	NR	NP	N #
S_{2m+2}	0	2	0	m	$2 \cdot 4^{2m}$	$2 \cdot 4^{2m}$
S_{2m+1}	0	1	1	m	2^{2m+1}	2^{2m+1}

$$p_2 = \frac{2^{2m+1}}{2^{2m+2}} = \frac{1}{2}$$

$${}^1R_m^2(\lambda, 0) = e^{-2\lambda T} \left[\sum_{i=0}^{2m+1} \frac{(2\lambda T)^i}{i!} + \frac{1}{2} \frac{(2\lambda T)^{2m+2}}{(2m+2)!} \right]$$

Example 3. <u>q = 3</u>

<u>q = 1</u> $\begin{pmatrix} -E+7 & 6 \\ 2 & -E+3 \end{pmatrix} \begin{pmatrix} N_0 \\ N_1 \end{pmatrix} = 0$

$$(E^2-10E+9)\,N_i(2j+1) = 0$$

$$N_i(2j+1) = a_i\,9^j + b_i$$

The initial conditions are

$$\begin{array}{lcl} 6 = 9a_1 + b_1 & \qquad & 21 = 9a_0 + b_0 \\ 60 = 81a_1 + b_1 & & 183 = 81a_0 + b_0 \\ \hline a_1 = 3/4 \quad b_1 = -3/4 & & a_0 = 9/4 \quad b_0 = 3/4 \end{array}$$

$$N_1(2j+1) = 3/4(9^j - 1)$$

$$N_0(2j+1) = 3/4(3\cdot 9^j + 1)$$

For the transformations to F,

State	NS	#1's	#0's	NR	NP	NT
S_{2m+3}	0	3	0	m	$3/4(9^{m+1}-1)$	$3/4(9^{m+1}-1)$
S_{2m+2}	0	2	1	m	$3/4(9^{m+1}-1)$	$3/2(9^{m+1}-1)$
S_{2m+1}	0	1	2	m	$3/4(3\cdot 9^{m}+1)$	$3/4(3^{2m+1}+1)$
	1	3	0	m-1	$3/4(9^{m}-1)$	

$$p_2 = \frac{3/2(9^{m+1}-1)}{(3/4)3(9^{m+1}-1)} = 2/3$$

$$p_3 = \frac{3/4\,(3\cdot 9^{m}+1)}{9^{m+1}} = 1/4\,(1+\frac{1}{3^{2m+1}})$$

$${}^{1}R^{3}_{m}(\lambda, 0) = e^{-3\lambda T} \sum_{i=0}^{2m+1} \frac{(3\lambda T)^{i}}{i!} + 3/4(1-\frac{1}{3^{2m+2}})\,\frac{(3\lambda T)^{2m+2}}{(2m+2)!}\,(1+\frac{\lambda T}{2m+3})$$

Example 4. $\underline{q=4}$

$\underline{q=2}$

The difference equation to be solved is

$$\begin{pmatrix} -E+10 & 6 \\ 6 & -E+10 \end{pmatrix} \begin{pmatrix} N_0 \\ N_1 \end{pmatrix} = 0$$

with solution

$$N_i(2i+1) = a_i \cdot 4^{i} + b_i\, 16^{i}$$

$$N_0(3) = 40,\ N_0(5) = 544,\ N_1(3) = 24,\ N_1(5) = 480$$

$$N_0(2i+1) = 2\cdot 4^{i}(4^{i}+1)$$

$$N_1(2i+1) = 2\cdot 4^{i}(4^{i}-1)$$

For the transformations to F

State	NS	#1's	#0's	NR	NP	NT to F
S_{2m+5}	0	5	0	m	$\frac{1}{16}[5 \cdot 25^{m+12} - 15 \cdot 9^{m+2} + 10]$	$\frac{1}{16}[5 \cdot 25^{m+2} - 15 \cdot 9^{m+2} + 10]$
S_{2m+4}	0	4	1	m	$\frac{1}{16}[5 \cdot 25^{m+2} - 15 \cdot 9^{m+2} + 10]$	$\frac{1}{4}[5 \cdot 25^{m+2} - 15 \cdot 9^{m+2} + 10]$
S_{2m+3}	0	3	2	m	$\frac{1}{8}[25^{m+2} - 15 \cdot 9^{m+1} - 10]$	$\frac{3}{8}[25^{m+2} - 15 \cdot 9^{m+1} - 10]$
	1	5	0	m	$\frac{5}{16}[25^{m+1} - 3 \cdot 9^{m+1} + 2]$	
S_{2m+2}	0	2	3	m	$\frac{1}{8}[5 \cdot 25^{m+1} + 5 \cdot 9^{m+1} - 10]$	$\frac{1}{4}[5 \cdot 25^{m+1} + 5 \cdot 9^{m+1} - 10]$
	1	4	1	m-1	$\frac{5}{16}[25^{m+1} - 3 \cdot 9^{m+1} + 2]$	
S_{2m+1}	0	1	4	m	$\frac{1}{16}[25^{m+1} + 5 \cdot 9^{m+1} + 10]$	$\frac{1}{16}[25^{m+1} + 5 \cdot 9^{m+1} + 10]$
	1	3	2	m-1	$\frac{1}{8}[25^{m+1} - 15 \cdot 9^{m+1} - 10]$	
	2	5	0	m-2	$\frac{5}{16}[25^{m} - 3 \cdot 9^{m} + 2]$	

Example 5. $\underline{q = 5}$

$\underline{q = 2}$

The difference equation to be solved is

$$\begin{pmatrix} -E+13 & 6 & 0 \\ 12 & -E+17 & 20 \\ 0 & 2 & -E+5 \end{pmatrix} \begin{pmatrix} N_0 \\ N_1 \\ N_2 \end{pmatrix} = 0$$

with solution

$$N_0(2i+1) = \frac{1}{16}\left[\, 25^{i+1} + 5\cdot 9^{i+1} + 10 \,\right]$$

$$N_1(2i+1) = \frac{1}{8}\left[\, 25^{i+1} - 15\cdot 9^{i} - 10 \right]$$

$$N_2(2i+1) = \frac{5}{16}\left[25^{i} - 3\cdot 9^{i} + 2 \,\right]$$

For the transformations to F,

State	NS	#1's	#0's	NR	NP	NT to F
S_{2m+4}	0	4	0	m	$2 \cdot 4^{m+1}(4^{m+1}-1)$	$2 \cdot 4^{m+1}(4^{m+1}-1$
S_{2m+3}	0	3	1	m	$2 \cdot 4^{m+1}(4^{m+1}-1)$	$6 \cdot 4^{m+1}(4^{m+1}-1)$
S_{2m+2}	0	2	2	m	$3 \cdot 4^{2m+1}$	$6 \cdot 4^{2m+1}$
	1	4	0	m-1	$2 \cdot 4^{m}(4^{m}-1)$	
S_{2m+1}	0	1	3	m	$2 \cdot 4^{m}(4^{m}+1)$	$2 \cdot 4^{m}(4^{m}+1)$
	1	3	1	m-1	$2 \cdot 4^{m}(4^{m}-1)$	

$$p_2 = \frac{6 \cdot 4^{m+1}[4^{m+1}-1]}{8 \cdot 4^{m+1}[4^{m+1}-1]} = 3/4$$

$$p_3 = \frac{6 \cdot 4^{2m+1}}{12 \cdot 4^{2m+1}} = 1/2$$

$$p_4 = \frac{2 \cdot 4^{m}(4^{m}+1)}{4^{2m+2}} = 1/8(1 + 1/4^{m})$$

$$^1R_m^4(\lambda, 0) = e^{-4\lambda T} \sum_{i=0}^{2m+1} \frac{(4\lambda T)^i}{i!} + 7/8(1 - 2^{\frac{1}{2m+3}}) \frac{(4\lambda T)^{2m+2}}{(2m+2)!}[1 + \frac{2\lambda T}{2m+3} + \frac{(\lambda T)^2}{(2m+3)(2m+2)}$$

$$p_2 = \frac{\frac{1}{4}[5\cdot 25^{m+2}-15\cdot 9^{m+1}+10}{\frac{5}{16}[5\cdot 25^{m+2}-15\cdot 9^{m+1}+10} = \frac{4}{5}$$

$$p_3 = \frac{\frac{3}{8}[25^{m+2}-15\cdot 9^{m+1}-10]}{\frac{3}{8}[25^{m+2}-15\cdot 9^{m+1}-10]+\frac{5}{16}[25^{m+2}-27\cdot 9^{m+1}+2]}$$

$$= \frac{6}{11-12\frac{9^{m+1}-1}{5\cdot 25^{m}-3\cdot 9^{m+1}-2}}$$

$$p_4 = \frac{\frac{1}{4}[5\cdot 25^{m+1}+5\cdot 9^{m+1}-10}{\frac{1}{4}[5\cdot 25^{m+1}+5\cdot 9^{m+1}-10]+\frac{1}{16}[55\cdot 25^{m+1}-45\cdot 9^{m+1}-10]}$$

$$= \frac{4}{15-\frac{12\cdot 9^{m+1}-20}{5\cdot 25^{m+1}+5\cdot 9^{m+1}-10}}$$

$$p_5 = \frac{\frac{1}{16}[25^{m+1}+5\cdot 9^{m+1}+10]}{25^{m+1}}$$

$$= \frac{1}{10}[1+\frac{9^{m+1}+2}{5\cdot 25^{m}}]$$

$${}^{1}R_{m}^{5}(\lambda, 0) = e^{-5\lambda T} \sum_{i=0}^{2m+1} \frac{(5\lambda T)^{i}}{i!}$$

$$+ \frac{15}{16}\left(1 - \frac{9^{m+1}+7}{5 \cdot 25^{m}}\right) \frac{(5\lambda T)^{2m+2}}{(2m+2)!} \left[1+(1-p_4)\frac{5\lambda T}{2m+3}\right.$$

$$\left. + (1-p_4)(1-p_3) \frac{(5\lambda T)^{2}}{(2m+3)(2m+4)} + \frac{(1-p_4)(1-p_3)}{5} \frac{(5\lambda T)^{3}}{(2m+3)(2m+4)(2m+5)}\right]$$

where p_3 and p_4 are given above.

2.2.5.2 Specific Examples, m small, f small, q arbitrary.

For approximations to ${}^{f}R_{m}^{q}$ $(\lambda, 0)$ the probabilities $p_{f(q-1)+1}$ and $p_{f(q-1)}$ must be calculated. The probability $p_{f(q-1)+1}$ is the probability that a member of state $S_{(f+1)(m+1)-1}$ is transferred to F (eg. that the system fails as soon as possible), and $p_{f(q-1)}$ is the probability that a member of state $S_{(f+1)(m+1)}$ is transferred to F. Let $f = 1$, q, m be arbitrary.

The easiest way to do the calculations is to define N_{ij} as the number of configurations with i units with a single failure (q active) during the jth state, and compile a list.

State	NC
S_0	$N_{00} = 1$
S_1	$N_{11} = q$
S_2	$N_{02} = N_{11} = q$
	$N_{22} = (q-1)N_{11} = q(q-1)$
S_3	$N_{13} = q\,N_{11} + 2N_{22} = q(3q-2)$
	$N_{33} = (q-2)N_{22} = q(q-1)(q-2)$
S_4	$N_{04} = N_{13} = q(3q-2)$
	$N_{24} = (q-1)N_{13} + 3(q-2)N_{22} = q(q-1)(6q-8)$
	$N_{44} = \prod_{i=0}^{3} (q-i)$

State	NC
S_5	$N_{15} = qN_{04} + 2N_{24} = q[15q^2 - 30q + 16]$
	$N_{35} = (q-2)N_{24} + 4N_{44} = q(q-1)(q-2)(10q-20)$
	$N_{55} = \overset{4}{\underset{i=0}{\pi}} (q-i)$
S_6	$N_{06} = N_{15} = q(15q^2 - 30q + 16)$
	$N_{26} = (q-1)N_{15} + 3N_{35} = q(q-1)[45q^2 - 150q + 136]$
	$N_{46} = (q-3)N_{35} + 5N_{55}$
	$= q(q-1)(q-2)(q-3)(15q-20)$
	$N_{66} = \overset{5}{\underset{i=0}{\pi}} (q-i)$
S_7	$N_{17} = qN_{06} + 2N_{26} = q[105q^3 - 420q^2 + 588_{q} - 272]$
	$N_{37} = (q-2)N_{26} + 4N_{46} = q(q-1)(q-2)[105q^2 - 410q + 376]$
	$N_{57} = (q-4)N_{46} + 6N_{66} = q(q-1)(q-2)(q-3)(q-4)[21q-50]$
	$N_{77} = \overset{6}{\underset{i=0}{\pi}} (q-i)$
S_8	$N_{08} = N_{17} = q[105q^3 - 420q^2 + 588q - 272]$
	$N_{28} = (q-1)N_{17} + 3N_{37} = q(q-1)[420q^3 - 2280q^2 + 417q - 1024]$
	$N_{48} = (q-3)N_{37} + 5N_{57} = q(q-1)(q-2)(q-3)[210q^2 - 1080q + 1376]$
	$N_{68} = (q-5)N_{57} + 7N_{77}$
	$= q(q-1)(q-2)(q-3)(q-4)(q-5)(28q-92)$

Since $f = 1$, $f(q-1) + 1 = q$, and clearly

$$p_q = \frac{N_{2(m+1)}}{q^{2m+2}}$$

$$p_{q-1} = \frac{2N_{2\ 2(m+1)}}{q^{2m+3} - qN_{0}2(m+1)}$$

<u>m = 1.</u>

$$p_q = \frac{3q-2}{q^3}$$

$$p_{q-1} = \frac{4(3q-4)}{q(q-1)(q+2)}$$

<u>m = 2.</u>

$$p_q = \frac{15(q-1)^2+1}{q^5}$$

$$p_{q-1} = \frac{2[45q^2-150q+136]}{q[q^4+q^3+q^2-14q+16]}$$

<u>m= 3.</u>

$$p_q = \frac{105(q-1)^2(q-2)+63(q-1)+1}{q^7}$$

$$p_{q-1} = \frac{2[420q^3 - 2280q^2 + 4176q - 1024]}{q[q^6+q^5+q^4+q^3-104q^2+316q-267]}$$

If $f = 1$, $\quad {}^{1}R^{q}_{m}(\lambda, \lambda) \approx 1 - C^{m+q}_{m+1} \left[\frac{(\lambda T)^2}{2} \right]^{m+1}$

$${}^{1}R^{q}_{m}(\lambda, 0) \approx 1 - p_q \frac{(q\lambda T)^{2(m+1)}}{[2(m+1)]!}$$

e) m = 1, q arbitrary.

$${}^{1}R^{q}_{1}(\lambda, \lambda) \cong 1 - \frac{q(q+1)}{8} (\lambda T)^4$$

$${}^{1}R^{q}_{1}(\lambda, 0) \cong 1 - \frac{q(3q-2)}{24} (\lambda T)^4$$

f) m = 2, q arbitrary

$${}^{1}R^{q}_{2}(\lambda, \lambda) \cong 1 - \frac{q(q+1)(q+2)}{48} (\lambda T)^6$$

$${}^{1}R^{q}_{2}(\lambda, 0) \cong 1 - \frac{15q(q-1)^2 + q}{(15)(48)} (\lambda T)^6$$

g) m = 3, q arbitrary

$${}^{1}R^{q}_{3}(\lambda, \lambda) \cong 1 - \frac{q(q+1)(q+2)(q+3)}{384} (\lambda T)^8$$

$${}^{1}R^{q}_{3}(\lambda, 0) \cong 1 - \frac{(105)(q-1)^2(q-2)q + 63\, q(q-1) + q}{(105)(384)} (\lambda T)^8$$

h) $f = 2$, $m = 1$, q arbitrary (without derivation here)

$$ {}^{2}R_{1}^{q}\ (\lambda, \lambda) \cong 1 - \frac{q(q+1)}{72} (\lambda T)^6 $$

$$ {}^{2}R_{1}^{q}\ (\lambda, 0) \cong 1 - \frac{q(10q-9)}{720} (\lambda T)^6 $$

2.3 State Variable Transform Models

One method of studying computer system architecture is to identify the set of states of the system with a finite dimensional vector space over a field and the transformations between states with linear transformations. A practical technique using this idea is called System Effectiveness Evaluation, [US, 65] and this technique has been used for planning, design optimization, procurement, and system management in a variety of applications. To quote the originators, "A principal concept underlying the SEE technique is the fact that its evaluation is performed in terms of system states and **their transitions**, not in terms of the subsystems or black boxes. By breaking the system down functionally by system states rather than structurally, the technique obviates any need to assume that subsystems function independently, thereby avoiding many computational problems concerned with interdependence or coupling among subsystems." However, the implementation is by computer program and simulation, and the mappings and their effects are obscured.

A new method, outlined here, is to identify the linear transformations with Boole's displacement operator. The advantage of a displacement operator is that in this case it leads to a set of finite difference equations whose solution is in finite form, is easily expressed and manipulated, and depends upon the characteristic roots of the mapping. This allows a wide body of mathematical knowledge to be used and combined with the usual properties of solutions of linear difference equations (similar to the better known properties of linear differential equations). The general and abstract properties of the system are made clear and available for analysis. Classes of inputs can be considered. These properties are typical of mathematical models and is the justification for the time spent in developing them. They must describe the physical situation accurately, be able to be used for calculations, and allow the effect of parameters to be clearly delineated.

The use of the Boole displacement operator E [Boole, 72] as a formal next state operator for linear transformations over a vector space leads to the following equation:

2.3.1) $$S(n+1) = ES(n) = MS(n)$$

where $S(j)$ is the j^{th} state vector and M is a $m \times m$ matrix. Since M satisfies its characteristic equation, 2.3.1) defines a linear difference equation in $S(j)$. There always exist characteristic vectors V_i which transform M into the third normal form F_3 (as defined by van der Waerden) [v. d. Waerden, 40] and which have the property that

2.3.2) $$MV_i = V_i\lambda_i \quad i = 1, \dots, m.$$

for each characteristic root λ_i. The expression

2.3.3) $$S(n) = \sum_{i=0}^{m-1} \beta_i V_i \lambda_i^n$$

where the β_i are constants to be determined by the initial conditions satisfies 2.3.1) for all n since

2.3.4) $$MS(n) = \sum_{i=0}^{m-1} \beta_i M V_i \lambda_i^{\,n} = \sum_{i=0}^{m-1} \beta_i V_i \lambda_i^{\,n+1} = S(n+1) = ES(n)$$

This vector solution is analogous to the usual scalar solution of a difference equation.

The initial conditions are

2.3.5) $$S(i) = M^i S(0), \quad i = 0, \dots, m-1.$$

If $|M| \neq 0$, solve

2.3.6) $$S(0) = \sum_{i=0}^{m-1} \beta_i V_i = V \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{m-1} \end{pmatrix}, \quad V = (V_0 V_1 \dots V_{m-1}), \quad |V| \neq 0$$

The β_i are unique, depend upon $S(0)$ except in the exceptions proved later and by 2.3.4) satisfy 2.3.1) and the initial conditions.

If equations 2.3.3) and 2.3.6) are used to find a solution, then the characteristic roots and characteristic vectors must be found, and then a set of linear equations solved. An alternative method of calculation is frequently simple and is interesting theoretically. Write

2.3.7) $$S(n) = (S_j(n)) = \sum_{i,j=0}^{m-1} a_{ij} \lambda_i^{\,n} .$$

Using 2.3.5) develop the following set of equations to determine the a_{ij}. For each j, $0 \leq j \leq m-1$,

2.3.8) $$S_j(n) = \sum_{i=0}^{m-1} a_{ij} \lambda_i^{\,n} \qquad 0 \leq n \leq m-1$$

The determinant of the coefficients is a Vandermonde determinant, and all λ_i are roots of the characteristic equation $g(\lambda) = |M - \lambda I| = 0$. In this case a simple formula can be found for the a_{ij}. Divide $g(\lambda)$ by the factor $\lambda - \lambda_i$ to obtain

$$f_j(\lambda) = \sum_{k=0}^{m-2} d_{k_j} \lambda^k .$$

Now, it can be shown that [Carter, 62]

$$2.3.9) \qquad a_{ij} = \frac{(-1)^i \sum_{k=0}^{m-2} d_{k_i} S_j(k)}{g'(\lambda_i)}$$

where $g'(\lambda_i)$ is the derivative of $g(\lambda)$ evaluated at λ_i. This equation allows the characteristic roots and corresponding coefficients to be found or analyzed one at a time.

There are many theorems which allow an estimation of the relative absolute values of the characteristic roots of a matrix or the roots of an arbitrary polynomial. Choosing $\lambda_0 = \operatorname{Max}_i \lambda_i$,

$$2.3.10) S(n) = \lambda_0^n \left(\beta_0 V_0 + \sum_{i=1}^{m-1} \beta_i V_i (\lambda_i / \lambda_0)^n \right) \rightarrow \beta_0 V_0 \lambda_0^n$$

gives a useful approximation.

In the second case,

if $|M| = 0$ and M has rank $m-1$, solve

$$2.3.11) \quad S(1) = \sum_{i=0}^{m-1} \beta_i V_i \lambda_i \quad \text{with solution}$$

2.3.12) $$\begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{m-1} \end{pmatrix} = V^{-1} M\, S(0) = F_3\, V^{-1} S(0)$$

Since F_3 may be chosen to have a first row of zeros when M has rank $m-1$, $\beta_0 = 0$ as well as $\lambda_0 = 0$, and the solution 2.3.11) depends upon $(m-1)$ non zero constants. The solution 2.3.3) with constants determined by 2.3.12) satisfies 2.3.1) and all initial conditions except $S(0)$. However, this is precisely the property of the solution desired in most cases, as will now be shown.

Let M be a column stochastic matrix. Then its greatest positive root is 1, and 1 is greater than the absolute value of all the other roots. Moreover, $(\lambda-1)$ appears only linearly in the list of elementary divisors of $M-\lambda I$. Choose $\lambda_0 = 1$, then

2.3.13) $$\lim_{n \to \infty} S(n) = \beta_0 V_0 = \overline{S}$$

The values of V_0 are proportional to cofactors of $(M-I)$, so V_0 is independent of $S(0)$. Let J be the matrix with all elements 1. Then, since $J = JM$, and $S(n) = (S_j(n))$, $0 \le j \le m-1$,

2.3.14) $$\begin{aligned} \sum_{j=0}^{m-1} S_j(n) &= (1,\ldots,1)S(n) = \frac{1}{m}(1,\ldots,1)\,J\,S(n) \\ &= \frac{1}{m}(1,\ldots,1)\,JM^m S(0) = \frac{1}{m}(1,\ldots,1)J\,S(0) \\ &= \sum_{j=0}^{m-1} S_j(0) \end{aligned}$$

using the definition of a column stochastic matrix, JM = J .

Write the left hand side of 2. 3. 14) as:

$$\sum_{j=0}^{m-1} S_j(n) = \sum_{i=0}^{m-1} \beta_i \lambda_i^n \sum_{j=0}^{m-1} V_{ij}$$

and evaluate for $n = 0, \ldots, m-1$; writing $\sum_{j=0}^{m-1} S_j(0) = S_0$

$$S_0 = \beta_0 \sum_{j=0}^{m-1} V_{0j} + \beta_1 \sum_{j=0}^{m-1} V_{1j} + \ldots + \beta_{m-1} \sum_{j=0}^{m-1} V_{m-1j}$$

$$S_0 = \beta_0 \sum_{j=0}^{m-1} V_{0j} + \lambda_1 \beta_1 \sum_{j=0}^{m-1} V_{1j} + \ldots + \lambda_{m-1} \beta_{m-1} \sum_{j=0}^{m-1} V_{m-1j}$$

$$S_0 = \beta_0 \sum_{j=0}^{m-1} V_{0j} + \lambda_1^{m-1} \beta_1 \sum_{j=0}^{m-1} V_{1j} + \ldots + \lambda_{m1}^{m-1} \beta_{m-1} \sum_{j=0}^{m-1} V_{m-1j}$$

This set of equations in

$$S_0 - \beta_0 \sum_{j=0}^{m-1} V_{0j}, \; \beta_i \sum_{j=0}^{m-1} V_{ij} \qquad i = 1, \ldots, m-1$$

has a Vandermonde determinant, non zero, and thus

2..3. 15) $$\beta_0 \sum_{j=0}^{m-1} V_{0j} = \sum_{j=0}^{m-1} S_j(0) = S_0$$

$$\beta_i \sum_{j=0}^{m-1} V_{ij} = 0 \qquad i = 1, \ldots, m-1 \; .$$

If $\sum_{j=0}^{m-1} S_j(0)$ is a constant, β_0 is independent of the particular initial configuration $S_0(0), \ldots, S_{m-1}(0)$. The limiting solution $\bar{S} = \beta_0 V_0$, given in 2.3.13) thus is independent of the particular initial configuration, and depends only on $\sum_{j=0}^{m-1} S_j(0)$.

Since M is defined to be column stochastic if and only if $JM = J$, the converse is true.

For the important special case of Markov processes or Markov chains, $\sum_{j=0}^{m-1} S_j(0) = 1$. Now the limiting or steady state probability of state occurrence is independent of the initial configuration.

The limiting state may be calculated by finding the characteristic vector corresponding to $\lambda = 1$ and then calculating β_0 using 2.3.9) since the formula holds except for the obvious case of $\lambda_{m-1} = 0$. In this case it was shown that $\beta_{m-1} = 0$.

The solution 2.3.3)of the next state transformation equation 2.3.1)has the advantage that it is in finite form, and is easily expressed and manipulated. All the properties of linear difference (or differential) equations are available–the sum of two solutions is a solution for example. The behavior of the solution with increasing n is frequently obvious and more straightforward than the treatment in terms of powers of the matrix M. The treatment in terms of powers of M has been detailed in the case in which M is stochastic and the process is one of Markov chains. Relative changes with n are easier to follow using 2.3.3). Equation 2.3.3)has another advantage–the coefficients of the powers of characteristic roots are expressed in terms of the initial state and characteristic matrices (equation 2.3.6)or in terms of the characteristic equation and the first m states (equation 2. 3. 9). As is more convenient either case can be used to evaluate the effect on S(n) of changing either M or the initial state.

In most cases of computer design analysis what is desired is not a detailed step by step consideration (simulation is good for this), but a knowledge of the limiting case together with an estimate of the speed of convergence of S(n) to this case. The limiting case was given by 2.3.10) for the general case and 2.3.13) for Markov chains.

Since V_0 corresponds to the characteristic root 1, the steady state solution of 2.3.13) for Markov processes may be obtained by solving the equations

2. 3. 16)
$$(M-I_m)\ V_0 = 0 \qquad\qquad (V_0 = (V_{0i}))$$
$$\beta_0 = \sum_{i=0}^{m-1} V_{0i}$$

The value of β_0 assumes that the sum of the limiting probabilities is one.

Howard [Howard, 60] calls the transpose of the matrix $(M-I_m)$ the 'rate matrix' corresponding to the Markov process defined by M and uses it as a basic tool for analysis.

A convenient method of determining the speed of convergence of S(n) to the limiting case is to consider the sizes of the largest and next to the largest characteristic roots. Much work has been done to determine bounds for these roots by eminent mathematicians. This work can be applied effectively to the analysis of models of computer processes as shown by Dumey. [Dumey, 63]

Determining the speed of convergence is important in practical examples. For Markov processes, if $S_i(n) > \overline{S}_i$ and approaches $\overline{S}_i$ slowly, then if such an $\overline{S}_i$ is used to estimate the capacity of a system, the system will tend to be overloaded. Knowing $S_i(n) > \overline{S}_i$ would allow a design to avoid this overloading.

A final advantage is that in many cases of interest the number of states is related to some outside parameter say k. For example, the number of people using a time-sharing system. Now $\beta_0 V_0, \beta_1 V_1$, and λ_1 will vary with k. For a Markov process $\lambda_0 = 1$, and variations in $\beta_0 V_0$ and λ_1 as k and n both vary can frequently be handled, and prove interesting.

Appendix 3 The REL Program

HOW

SYSTEM RELIABILITY CALCULATION

FUNCTIONAL UNIT NAMES ARE AS FOLLOWS: BALU(BYTE ARITHMETIC LOGIC UNIT), PCU(PROGRAM CONTROL UNIT), CCU(CHANNEL CONTROL UNIT), ROS(READ ONLY STORE), MS(MAIN STORE), SS(SECONDARY STORE), DR(DRUM), CL(CLOCK), POW(POWER SUPPLY),AND BPB,BPR, BBR,BPM,BCM,BCS,BCD(ALL BUSSES WITH THE LAST TWO LETTERS BEING THE FIRST LETTERS OF NAMES OF THE TERMINAL UNITS).

PARAMETERS ARE:R[SPEC]=SPECIFIED SYSTEM RELIABILITY, Y=YEAR (FAILURE RATES LAMBDA(LA) AND MU DEPEND ON DATE), K=LA÷MU= RATIO OF POWER ON TO POWER OFF FAILURE RATES, D=DUTY CYCLE, Q=REQUIRED NO. OF UNITS FOR ADEQUATE FUNCTIONING, M=NO. OF SPARE UNITS, S=NO. OF MODULES INTO WHICH THE UNIT IS SEGMENTED, I=INCREMENTAL HARDWARE FACTOR FOR SEGMENTING MODULES, C=COVERAGE OF CHECKERS AND DIAGNOSTICS, F=FAILURE TOLERANCE, SEL=A VECTOR THAT SELECTS PRINTOUT CONTENTS, AND R CONTAINS ALL RELIABILITIES(UNITS,SYSTEM,SPEC). Y,D,K,AND T(MISSION TIME) ARE SCALARS, ALL ELSE ARE VECTORS.

INSTRUCTIONS FOR USE

```
1.EXECUTE FUNCTION 'SETUP' TO INITIALIZE PARAMETERS.
        (DISPLAY SETUP FOR DETAILS)
2.MODIFY PARAMETERS AS DESIRED.
        EXAMPLES:M[BALU,MS]←2 3 ;T←6500 ;R[SPEC]←.98
3.EXECUTE 'RELIABILITY' FUNCTION.
4.AT ANY TIME PRINT ANY QUANTITY.
        EXAMPLES:S[PCU] ;T  ;R  ;Q[BPB] ;LA[DR]  ;K  ;SEL
5.→2
```

THE 'TIME' FUNCTION FINDS A MISSION TIME THAT ACHIEVES THE SPECIFIED SYSTEM RELIABILITY.

THE 'DISPLAY' FUNCTION PRINTS OUT THE FOLLOWING COMPUTED RESULTS DEPENDING ON THE CONTENTS OF SEL:

```
     1∊SEL;CRITICAL SYSTEM PARAMETERS AND RESULTS.
     2∊SEL;NORMALIZED PROBABILITIES OF FAILURE FOR UNITS.
     3∊SEL;R AND 1-R FOR EACH UNIT.
     4∊SEL;THE SYSTEM CONFIGURATION.
     5∊SEL;THE SYSTEM REL. AND NORMALIZED PROB.OF FAILURE.
```

```
      ∇ SETUP
[1]     T←10879
[2]     D←0.25
[3]     Y←1968
[4]     K←5
[5]     INDEX
[6]     SEL←ι3
[7]     EXP←4
[8]     R←(17ρ0),0.997
[9]     M← 1 2 2 2 2 0 2 0 2 3 3 3 3 3 3 3
[10]    Q← 4 1 1 3 1 3 3 0 1 12 95 36 15 36 36 0
[11]    C←16ρ1
[12]    S← 1 3 4 1 2 4 1 1 1 ,7ρ1
[13]    I← 0 0.1 0.1 0 0.1 0.1 ,10ρ0
[14]    F← 1 0 1 1 0 0 1 ,9ρ0
[15]    PRINTOUT←(Q≠0)/ι16
[16]    CHARS←' 0123456789.|QWERTYUIOPASDFGHJKLZXCVBNM[]()×÷+'
[17]    UNAMES←(16 7)ρ' BALU    PCU     ROS    MSTORE  CCU     CLOCK
          SSTORE  DRUM   POWER    BPB     BPR      BPM      BBR      BCM
            BCS     BCD    '
[18]    QQ←'BALU   PCU    ROS   MSTORE CCU   CLOCK SSTORE DRUM   POW
        ER '
[19]    HEAD←QQ,' BPB   BPR    BPM    BBR    BCM    BCS    BCD    '
[20]    WIDTH←(1 7)ρ'WIDTH  '
      ∇

      ∇ INDEX
[1]     POW←1+DR←1+SS←1+CL←1+CCU←1+MS←1+ROS←1+PCU←1+BALU←1
[2]     SWC←1+SWP←1+PAGE←1+CIR←1+BCD←1+BCS←1+BCM←1+BBR←1+BPM←1+
        BPR←1+BPB←10
[3]     SPEC←1+SYS←17
      ∇

      ∇ RELIABILITY
[1]     FAILRATES
[2]     RELCALC
[3]     DISPLAY
      ∇

      ∇ RELCALC
[1]     R[ι17]←0
[2]     R[BALU]←RMQCSF BALU
[3]     R[PCU]←(RMQCSF PCU)×RSW SWP,PCU
[4]     R[ROS]←RROS
[5]     R[MS]←RMQCSF MS
[6]     R[CCU]←(RMQCSF CCU)×RSW SWC,CCU
[7]     R[CL]←RTMR CL
[8]     →(Q[SS]<1)/10
[9]     R[SS]←RMQCSF SS
[10]    →(Q[DR]<1)/12
[11]    R[DR]←RMQC DR
[12]    R[POW]←RMQC POW
[13]    R[BUSS]←RBUSS BUSS
[14]    R[SYS]←×/R[UNITS,BUSS]
      ∇
```

```
     ∇ FAILRATES;YY;F
[1]    INDEX
[2]    YY←Y-1966
[3]    G←1÷K
[4]    F←(2 1)ρ(D+(1-D)×G),G
[5]    L←(2 20)ρ6931.47
[6]    N←M+Q
[7]    UNITS←(Q[⍳9]≠0)/⍳9
[8]    BUSS←,(Q[9+⍳7]≠0)/(9+⍳7)
[9]    SAS←+/((7 2)ρN[PCU,BALU,PCU,ROS,PCU,MS,BALU,ROS,CCU,MS,
       CCU,SS,CCU,DR])[BUSS-9;]
[10]   L[;PAGE]←(5 4.7 3.7 2.6)[YY]×F
[11]   L[;CIR]←L[;PAGE]÷475
[12]   L[;BALU]←L[;PAGE]×1.5
[13]   L[;PCU]←(L[;PAGE]×10×1+I[PCU]×(S[PCU]-1)*
       0.5)÷S[PCU]
[14]   L[;SWP]←L[;CIR]×8×N[PCU]
[15]   L[;ROS]←(((((5 11)÷16)×(20 20 17 14)[YY])+L[;PAGE]×
       3.5)×1+I[ROS]×(S[ROS]-1)*0.5)÷S[ROS]
[16]   L[;MS]←((50 47 42 37.5)[YY]×F)+(
       15 12 12 11.5)[YY]+0.5×L[;,PAGE]
[17]   L[;CCU]←(L[;PAGE]×5×1+I[CCU]×(S[CCU]-1)*
       0.5)÷S[CCU]
[18]   L[;SWC]←L[;CIR]×8×N[CCU]
[19]   L[;CL]←(L[;PAGE]×1+I[CL]×(S[CL]-1)*
       0.5)÷S[CL]
[20]   L[;SS]←((50 47 42 37.5)[YY]×F)+0.5×L[;,PAGE]+
       0.67×(15 12 12 11.5)[YY]
[21]   L[;DR]←(15 10 9 8)[YY]+((5 5 5 4)[YY]×F)+
       0.5×L[;,PAGE]
[22]   L[;POW]←(20 15 13 12)[YY]×F
[23]   L[;BUSS]←(2,ρBUSS)ρ((6×Q[BUSS]),(ρBUSS)ρ10)×(2×ρBUSS)ρ
       SAS×L[1;CIR]
[24]   L←L×1E¯6
[25]   LA←L[1;]
[26]   MU←L[2;]
     ∇

     ∇ DISPLAY;I;LI
[1]    I←0
[2]    LI←(0 4 6 8 10 12)[1+SEL],0
[3]    →LI[I←I+1]
[4]    PRINTSYS
[5]    →LI[I←I+1]
[6]    PRINTNPOF
[7]    →LI[I←I+1]
[8]    PRINTREL
[9]    →LI[I←I+1]
[10]   CONFIG
[11]   →LI[I←I+1]
[12]   PRINTTOT
[13]   →LI[I←I+1]
     ∇
     ∇ R←RMQCSF U
[1]    →(F[U]=⍳2)/ 4 8
[2]    R←RMQCS U
[3]    →0
[4]    R←RMQCF U
[5]    →(S[U]=1)/0
[6]    R←R*S[U]
[7]    →0
[8]    'F>1'
[9]    →R←0
     ∇

     ∇ R←RMQCS U
[1]    R←RMQC U
[2]    →(1=S[U])/0
[3]    R←R*S[U]
     ∇

     ∇ R←RMQCF U;I;J;LAT
[1]    →(C[U]=1)/4
[2]    'C≠1'
[3]    →R←0
[4]    I←0,⍳1+2×M[U]
[5]    J←0,⍳M[U]
[6]    LAT←LA[U]×T
[7]    →(Q[U]=⍳4)/ 10 12 15 18
[8]    'Q≠⍳4'
[9]    →R←0
[10]   R←(*-LAT)×(((1-G)×(+/(LAT*I)÷!I))+G×(1+LAT)×+/(1-(*-LAT
       )×1+LAT)*J)
[11]   →0
[12]   R←((2×LAT)*2+2×M[U])÷2×!2+2×M[U]
[13]   R←(*-2×LAT)×(((1-G)×(+/((2×LAT)*I)÷!I)+R)+G×((1+LAT)*
       2)×+/(J+1)×(1-(*-LAT)×1+LAT)*J)
[14]   →0
[15]   R←(3÷4)×(1-3*-2+2×M[U])×(((3×LAT)*2+2×M[U])÷!2+
       2×M[U])×1+LAT÷3+2×M[U]
[16]   R←(*-3×LAT)×(((1-G)×(+/((3×LAT)*I)÷!I)+R)+G×((1+LAT)*
       3)×+/(2!J+2)×(1-(*-LAT)×1+LAT)*J)
[17]   →0
[18]   R←(7÷8)×(1-2*-3+2×M[U])×(((4×LAT)*2+2×M[U])÷!2+
       2×M[U])×1+LAT×(2÷1+2×M[U])+LAT÷(3+2×M[U])×2+2×M[U]
[19]   R←(*-4×LAT)×(((1-G)×(+/((4×LAT)*I)÷!I)+R)+G×((1+LAT)*
       4)×+/(3!J+3)×(1-(*-LAT)×1+LAT)*J)
     ∇

     ∇ R←RMQC U;I
[1]    →(1=C[U])/5
[2]    I←0,⍳M[U]
[3]    R←+/(I!M[U])×(C[U]*I)×((1-C[U])*M[U]-I)×RMQ U
[4]    →0
[5]    R←(RMQ U)[1+M[U]]
     ∇
```

```
      ∇ R←RMQ U;I
[1]     R←,*-Q[U]×LA[U]×T
[2]     →(0=M[U])/0
[3]     I←⍳M[U]
[4]     R←R×+\1,(×\(Q[U]×LA[U]÷MU[U])÷I-1)×((1-*-MU[U]×T)*I)÷!I
      ∇

      ∇ R←RTMR U
[1]     R←(1-((1-*-LA[U]×T)*N[U]-1)×1+(N[U]-1)×*-LA[U]×T)*S[U]
      ∇

      ∇ R←KROS;L1;L2
[1]     L1←1
[2]     L2←2
[3]     R←(1-(1-(*-L[L1;ROS]×T)×(*-L[L2;ROS]×T)×1+L[L2;ROS]×T)*
        N[ROS])*S[ROS]
      ∇

      ∇ R←RSW V
[1]     R←(1-(1-(*-LA[V[1]]×T))*2)*S[V[2]]-1
      ∇
```

```
      ∇ TIME;PS;T0;R0
[1]     FAILRATES
[2]     PS←1-R[SPEC]
[3]     RELCALC
[4]     T0←T
[5]     R0←R[SYS]
[6]     (8ρ' ';+⌊T←(T≤0)+(T>0)×T)
[7]     T←T×(PS÷1-R[SYS])*1÷EXP
[8]     RELCALC
[9]     EXP←(⍟(1-R0)÷1-R[SYS])÷⍟T0÷T
[10]    →4×⍳(|T0-T)>1
[11]    ('      TIME = ';T←⌊T+0.5;' HOURS')
      ∇
```

```
     ∇ PRINTSYS;N
[1]    ''
[2]    QQ←'        TOT SYS REL.=.'
[3]    (QQ;⌊10000×R[SYS];'    PROB OF FAIL.=';
       0.1×⌊1000×(1-R[SYS])÷1-R[SPEC];'(NORMALIZED TO 100)
                YEAR=';Y)
[4]    QQ←'        SYS REL SPEC=.'
[5]    (QQ;⌊0.5+10000×R[SPEC];'    MISSION DURATION=';⌊T;'
       HOURS    DUTY CYCLE=';D;'    LA÷MU=';K)
     ∇

     ∇ PRINTNPOF;N;Z
[1]    ''
[2]    N←1+(5⍴10)REP,⌊0.5+10000×(1-R[,PRINTOUT])÷1-R[,SPEC]
[3]    N[;1]←N[;1]-Z←N[;1]=1
[4]    Z←Z/⍳(⍴N)[1]
[5]    N[Z;2]←N[Z;2]-N[Z;2]=1
[6]    N←(1 1 1 0 1 1)\N+1
[7]    N[;4]←((⍴N)[1],1)⍴CHARS⍳'.'
[8]    '               ',,((16 6)⍴HEAD)[PRINTOUT;]
[9]    '       NPOF',CHARS[,N]
     ∇

     ∇ PRINTREL;REL;M;N;REM
[1]    REL←,R[PRINTOUT]-1E¯13×R[PRINTOUT]=1
[2]    M←((⍴REL),6)⍴CHARS⍳' ×N×××'
[3]    M[;2]←2+N←((9=(9⍴10)REP,⌊1000000000×REL)IOTA 0)-1
[4]    M[;3+⍳3]←2+((⍴REL),3)⍴(3⍴10)REP⌊REM←1000×(REL×
       10*N)-(10*N)-1
[5]    '       RELS',CHARS[,M]
[6]    M[;3+⍳3]←2+((⍴REL),3)⍴(3⍴10)REP 1000-⌊REM
[7]    M[;3]←CHARS⍳'Z'
[8]    '       POFS',CHARS[,M]
     ∇

     ∇ CONFIG
[1]    INDEX
[2]    ''
[3]    UNITS←(Q[⍳9]≠0)/⍳9
[4]    BUSS←(Q[9+⍳7]≠0)/9+⍳7
[5]    (16⍴' '),'     UNIT        N       M       Q     100×C
          S      100×I     F    '
[6]    (16⍴' '),'       ↓       TOTAL  SPARES   REQ    COVE
         SEG      INC     FAIL '
[7]    (UNAMES[,UNITS;])PRINTCON((7 16)⍴((M+Q),M,Q,(
       100×C),S,(100×I),F))[;,UNITS]
[8]    ''
[9]    →(0=⍴BUSS)/0
[10]   (16⍴' '),'      BUSS      ',,UNAMES[BUSS;]
[11]   WIDTH PRINTCON((+/Q[BUSS]≠0),1)⍴Q[BUSS]
[12]   ''
     ∇
     ∇ HEAD PRINTCON VAL;N;L;I;LI
[1]    N←(ρVAL)[1]
[2]    L←(ρVAL)[2]
[3]    I←(HEAD[;1]=' ')/ι(ρHEAD)[1]
[4]    HEAD[I;]←HEAD[I;(1+ι6),1]
[5]    I←1
[6]    LI←1+(3ρ10)REP VAL[;I]
[7]    LI[; 1 2]←LI[; 1 2]-((LI[;1]=1)∘.∧ 1 0)∨(∧/LI[;
       1 2]=1)∘.∧ 1 1
[8]    LI←,1+(0 0 1 1 1 0 0)\(N,3)ρLI
[9]    (16ρ' '),'        ',HEAD[I;],CHARS[LI]
[10]   →(L≥I←I+1)/6
     ∇

     ∇ PRINTTOT
[1]    ('        R=';1E¯5×⌊100000×R[SYS];'     NPOF=';
       0.001×⌊100000×(1-R[SYS])÷1-R[SPEC])
     ∇

     ∇ V←M IOTA S;I
[1]    →(1<ρρM)/3
[2]    M←(1,ρ,M)ρM
[3]    I←1
[4]    V←ι0
[5]    V←V,M[I;]ιS
[6]    →((ρM)[1]≥I←I+1)/5
     ∇

     ∇ R←V REP M
[1]    →(1<ρρV)/7
[2]    V←,V
[3]    R←ι0
[4]    →((×/(ρM),ρV)≠ρR←R,V⊤(,M)[1+(ρR)÷ρV])/4
[5]    R←((ρM),ρV)ρR
[6]    →0
[7]    HALT
     ∇
```

```
                                   AN EXAMPLE WHICH SHOWS:
SETUP                             ←PARAMETER INITIALIZATION,
Y←1969                            ←CHANGES TO SOME PARAMETERS,
R[SPEC]←.99
TIME                              ←DETERMINATION OF MISSION TIME,AND
  10879
  17103
  18185
  18120
TIME = 18121 HOURS
SEL←4 1 2 3
DISPLAY                           ←↓A DISPLAY OF THE RESULTS OF THE COMPUTATION.

              UNIT     N       M       Q     100×C    S    100×I    F
               ↓     TOTAL  SPARES    REQ    COVER   SEG    INC    FAIL
              BALU     5       1       4      100     1      0      1
              PCU      3       2       1      100     3     10      0
              ROS      3       2       1      100     4     10      1
              MSTORE   5       2       3      100     1      0      1
              CCU      3       2       1      100     2     10      0
              CLOCK    3       0       3      100     4     10      0
              SSTORE   5       2       3      100     1      0      1
              POWER    3       2       1      100     1      0      0

              BUSS    BPB     BPR     BPM     BBR     BCM    BCS
              WIDTH   12      95      36      15      36     36

TOT SYS REL.=.9900    PROB OF FAIL.=99.9(NORMALIZED TO 100)            YEAR=1969
SYS REL SPEC=.9900    MISSION DURATION=18121 HOURS     DUTY CYCLE=0.25     LA÷MU=5

       BALU  PCU    ROS   MSTORE CCU   CLOCK SSTORE POWER  BPB   BPR    BPM    BBR    BCM    BCS
NPOF   0.04 14.25  7.34  35.97  3.71  7.33  7.38   3.76  0.06  15.11  1.78   0.12   1.78   1.78
RELS  5N566 2N857 3N266 2N640 3N628 3N267 3N262  3N623 5N384 2N848  3N822  4N878  3N822  3N822
POFS  5Z434 2Z143 3Z734 2Z360 3Z372 3Z733 3Z738  3Z377 5Z616 2Z152  3Z178  4Z122  3Z178  3Z178
```

Security Classification

3ND PPSO 13152

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) International Business Machines Corporation Thomas J. Watson Research Center Yorktown Heights, New York 10598	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE
PHASE II OF AN ARCHITECTURAL STUDY FOR A SELF-REPAIRING COMPUTER

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Final Report June 1966 - November 1967

5. AUTHOR(S) (Last name, first name, initial)
Roth, John P.; Bouricius, Willard G.; Carter, William C.; and Schneider, Peter R.

6. REPORT DATE November 1967	7a. TOTAL NO. OF PAGES 440	7b. NO. OF REFS 119

8a. CONTRACT OR GRANT NO. AF04 (695) - 1056 b. PROJECT NO. 3176 03, Computer Systems Usability c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) SAMSO TR-67-106 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None

10. AVAILABILITY/LIMITATION NOTICES
This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of SAMSO (SMTRE)

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Hq Space and Missile Systems Orgn (AFSC) AF Unit P O Los Angeles, California 90045

13. ABSTRACT

The architecture and organization of a self-repairing computer which operates correctly even if some of its components malfunction is described. This computer, using currently available components, has a 0.997 probability of correct operation on a 10,000 hour mission with a 25% duty cycle. First the problem is stated, an overview of previous work is given, and the results of the investigation are summarized. Then several current computer component technologies are appraised and the assumption in reliability models of a Poisson failure distribution for these components is justified. Next, the development of several algorithms for the generation and evaluation of diagnostic test patterns for computer circuits is described. These algorithms are illustrated by examples and by detailed APL programs. Next, mathematical reliability evaluation models for a variety of functional computer units are devised, their APL programming implementation given, and their use in the design process illustrated. The effects of failure coverage, duty cycle, number of spares, failure tolerance, lower component failure rate with power off, and other parameters are tabulated and delineated. Last, the architecture and operation of an automatically repaired computer at different levels of organization is described, including failure tolerant storage, ROS, and arithmetic logical organs. The design of reconfiguration switches, status registers, and a fully checked decoder is also given. By employing these techniques, which require about 2.5 times as much hardware as an equivalent simplex computer, it is possible to gain the same benefits as would be produced by a 600 fold decrease in component failure rates. ( )

DD FORM 1 JAN 64 1473 IIA 0101 807 6800

Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Self-Repairing Computer						
Computer Design						
Reliable Computer						
Computer Diagnosis						
Error Detecting						
Reliability Modeling						
Automatically Repaired Computer						
Reconfigurable Computer						
Computer Redundancy						
Highly Available Computer						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through ______________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through ______________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through ______________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.

Security Classification