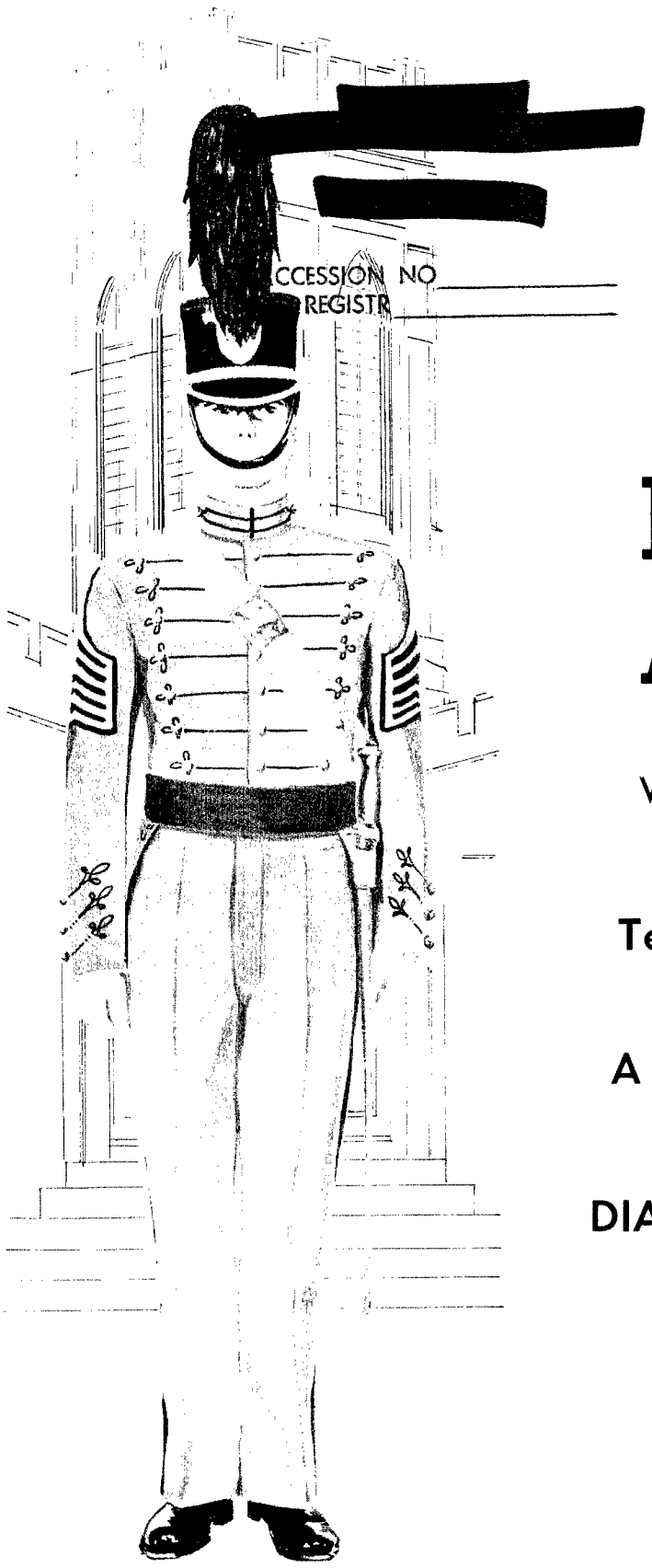
[redacted]






# UNITED STATES MILITARY ACADEMY

WEST POINT · NEW YORK

Technical Report Elec-67-3

## A PRAGMATIC FIRST LOOK AT DIAGNOSTIC PROGRAMMING FOR DIGITAL COMPUTERS

DEPARTMENT OF ELECTRICITY

May 1967

**DUTY-HONOR-COUNTRY**

# A PRAGMATIC FIRST LOOK
# AT
# DIAGNOSTIC PROGRAMMING
# FOR
# DIGITAL COMPUTERS

by

Philip H. Enslow, Jr.
Major, Signal Corps

Alan B. Salisbury
Major, Signal Corps

May 1967

Technical Report Elec - 67-3

Department of Electricity
United States Military Academy
West Point, New York

## DISCLAIMER

Opinions expressed herein are those of the authors and are not to be construed as official or necessarily reflecting the views of the U.S. Military Academy, the U.S. Army, or the Department of Defense.

## ABSTRACT

A brief survey of diagnostic programming for digital computers. The purposes and organization of the program are covered as well as many of the problems attendant to their preparation. Various techniques that have been used to overcome these are discussed with emphasis on those that are practical (at least from the point of view of effort required) and those that have seen wide use. The primary purpose of this paper is to fully define the nature of the problem and the special considerations involved. Most of the discussion is fully applicable to both commercial and military equipment.

TABLE OF CONTENTS

## ACKNOWLEDGEMENT

The work reported here was supported in part by a grant from the Army Research Office Durham which was used to purchase the test vehicle. The publication of this technical report was supported by the United States Military Academy.

The authors would like to express their appreciation to Mr. Arnold Siegel and the staff of Decision Systems Incorporated of Paramus, New Jersey for many productive discussions. Also our appreciation is extended to the many people who helped gather appropriate literature, and particularly to DSI, Philco, RCA, UNIVAC Division of Sperry Rand, IBM, Honeywell Inc and Sylvania, CDC, and DEC for making available examples of diagnostic program documentation and/or reference manuals.

This report was written while the authors were members of the faculty of the Department of Electricity at the United States Military Academy.

## PREFACE

The various attitudes toward diagnostic programs range from "the answer to the maintenance problem" all the way to "they are not worth the preparation effort". A large amount of this divergence of opinion appears to have been caused by the absence of a clear-cut framework on which to base common discussion, or an accurate and complete description of the nature of the problem and its practical considerations. Satisfying this need is the purpose of this paper.

Much of the material presented here is based on an extensive literature search [1]* augmented by personal discussion and correspondence with workers in the field. Many of the points advanced by these sources have been corroborated by experimental work done by the authors. This experimental work also resulted in the development of several original concepts which are also presented here.

When this study was originally conceived it was decided that it should definitely be oriented toward practical solutions. The scope of the project was to examine the basic principles and philosophies of diagnostic programs, the methods of perparing such programs, and the possibility of using automatic programming aids to assist in their preparation. To meet the requirement for "practical solutions" a test vehicle was needed. Since it was anticipated that several different techniques would be examined, a small computer was desired. Most of the test vehicle requirements were satisfied by the Fabri-Tek BI-TRAN SIX, a small desk-top unit specially designed for use in teaching computer concepts. The availability of an actual machine possessing most of the standard features while still being small was of inestimable value in the project. Although the test vehicle was small, we have been unable to discern any characteristics that detract from the general applicability of our findings.

The results of this project are given in three other reports in addition to this one:

"Diagnostic Programming for Digital Computers: A Bibliography"

"Documentation Techniques for Digital Computers"

"Documentation of the Fabri-Tek BI-TRAN SIX" (In preparation)

As with most papers of this type, we are afraid we have asked more questions than we have answered, but that is acceptable since one of our purposes was to identify the questions that need answers.

*Numbers in brackets refer to items in the list of references at the end of this paper.

## LIST OF ILLUSTRATIONS

# I. INTRODUCTION

## A. WHAT IS A DIAGNOSTIC PROGRAM?

A diagnostic program includes the features found in what are commonly called check-out routines and fault-location routines. The function of a check-out routine is merely to verify the correct operation of the machine; and, in fact, some of them give no indication of the general nature or location of a fault if one is detected. The purpose of the fault-location routine is exactly as its name indicates.

The distinction between "diagnostic program" and "check-out routines" is primarily in their basic philosophy and purpose rather than in their organization and operation. The arguments usually arise when the factor of resolution in the fault-location process is considered. A magnetic tape routine that fails with the gross indication that channel three failed has certainly accomplished something in the area of fault-location although the diagnostic programmer will argue that the information is of little value. The "pure" check-out routine would have to have only two outcomes or indications - "O.K." or "Error". Any additional information such as which part of the check-out routine detected the error does have some fault-location value (and thus diagnostic value). A more workable distinction can be derived from figure 1.

> Check-out routines are those programs whose primary purpose is the detection of faults. If there are pre-planned procedures or steps to aid in fault location specified for each fault detected, then the entire routine is classed as a diagnostic program.

### 1. Organization

The general organization of a complete diagnostic program including the manual procedures necessary at the beginning is given in figure 1. The series of test routines shown down the center of the page is of course the desired route for the program to follow. By themselves these test routines constitute what are normally referred to as check-out routines. When embedded in a complete diagnostic program, they are given the name "go path" by some.*

---

*eg. Decision Systems Incorporated.

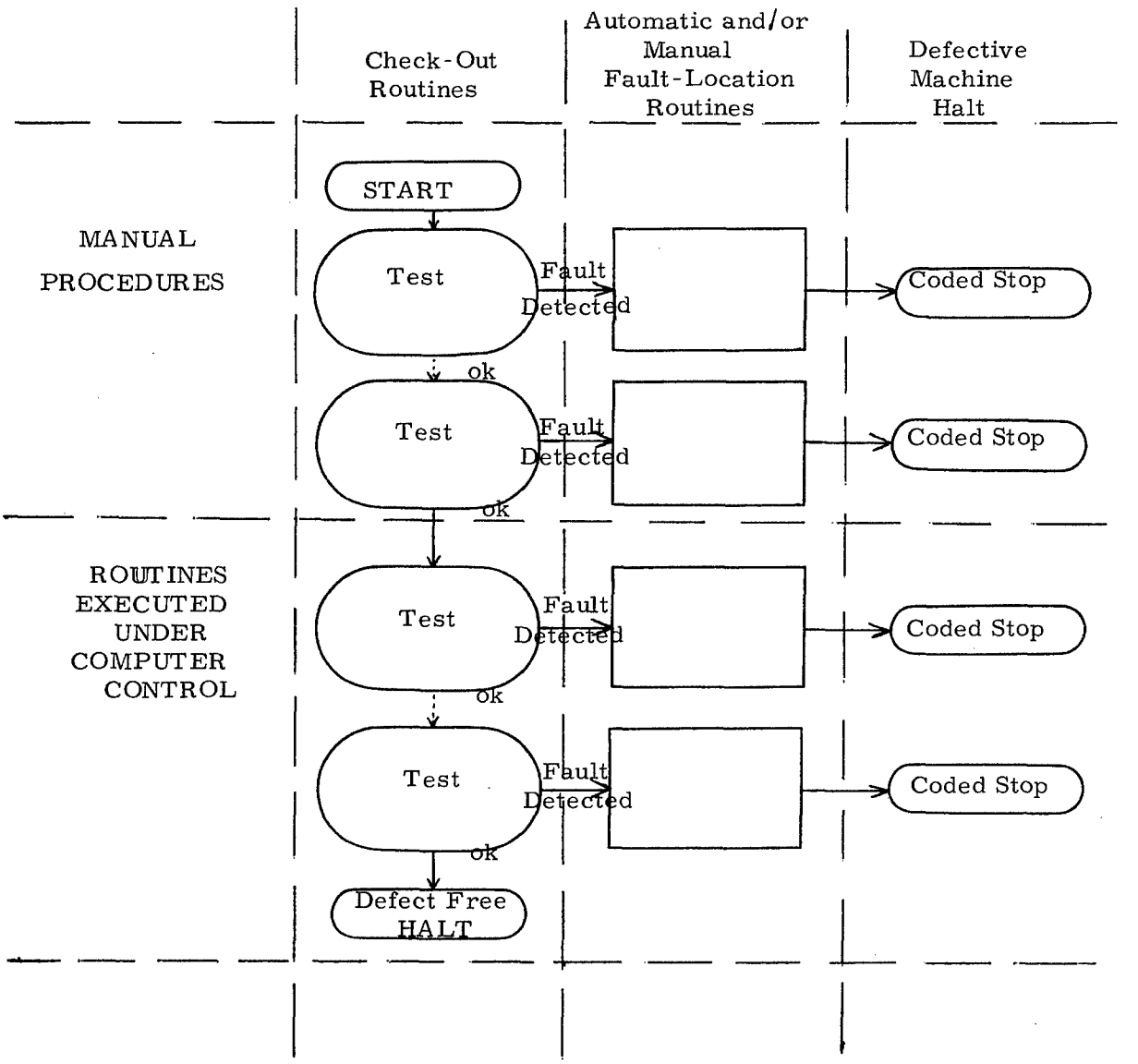



Figure 1: General Organization of Diagnostic Procedures

## 2. Purposes

A good diagnostic program serves several purposes.

(1) It can be used to verify the "fault-free condition" of the system.

(2) If a failure is detected during routine check-out or during production operations, the diagnostic can be used to identify the replaceable element that contains the failure.

(3) After the defective element has been replaced, the diagnostic can be used to verify that the machine is now fault-free.

The expression "fault-free condition" must be used with care in describing the operation of a diagnostic program. The quality of the program is the key factor, for if the machine successfully passes all the tests in the diagnostic, then actually all that can be said is "it performed these tests correctly - not "it is completely fault-free".

Since the diagnostic program serves all the purposes listed above, there is flexibility as to when it is used. To save the expense of preparing separate "check-out" routines, the diagnostic program can be used on a regular basis for machine check-out such as during morning turn-on. Another technique of use recently proposed was in a time-sharing system. The diagnostic would be run on a continuously repetitive basis whenever there were no other demands on the machine [2]. Both of these uses would require some Modification in the diagnostic program, primarily in the initial manual procedures. Finally, the diagnostic is used in its complete form whenever a fault is detected or suspected.

## 3. How They Work in General

As was shown in figure 1, a diagnostic program normally performs a series of tests to check-out various sections of hardware, and based upon the results of those tests either continues in the check-out mode or branches to a fault location routine if an error has been detected, finally arriving at a coded halt which, in association with a dictionary or similar form of documentation, indicates a replaceable unit in which a faulty element is located.

In preparing such a program it is essential to make assumptions as to the number and types of faults which may occur in a given machine. The program itself then consists of a series of first manual and next programmed tests from whose results inferences are drawn as to the conditions of various hardware elements. The tests which are necessary can be identified only after a thorough study of the logical organization of the machine and the details of the hardware implementations of that logic. A complete diagnostic program must execute tests which verify that every signal functions properly - it is either active or inactive as specified by the design logic.

The general method followed during the execution of the check-out routine is that only previously checked-out portions are used to examine the operation of a new section. This means that when a fault is detected it is known to be somewhere in the "new section" and the program transfers to a fault-location routine to resolve its location. As will be discussed in the next section, it is usually assumed that only a single fault exists in the machine so it is then possible to rely on the proper operation of all of the machine except this "new section" to aid in the fault location procedures.

4. Assumptions Normally Made

The most generally accepted assumptions pertain to the state of the machine and the number of faults present during the execution of the different phases of the diagnostic as shown in figure 1. During the manual check-out routines there are normally no assumptions made. The purpose of these routines is to validate the assumption made during the automatic check-out routines that the machine is "breathing", i.e., the input, the memory, and the control unit are operative at least to the degree that a program can be placed in memory and the simplest instruction (perhaps a "do-nothing") can be executed without causing the machine to "hang-up".

After a fault has been detected and fault location is initiated, the basic assumption made in all programs the authors have seen is that there is only one fault or component failure. This assumption is critical to the basic philosophy of the operation of all extant diagnostic programs. It is possible for the diagnostics to function properly if more than one fault is present provided the second one does not cause erroneous results during the fault-location of the first. In such a situation, the coded halt would specify the first fault, and after it was repaired the diagnostic would then proceed past the first halt and stop at the coded halt for the second fault. It is not possible to generalize on the conditions necessary for this to happen - each pair of faults has to be analyzed separately. There has been some work done on the problem of multiple faults by ARIES, Inc., using a simulator, but no results have yet been published. [3].

The single fault that is present is also assumed to be a solid one; that is, not an intermittent. Of course, if the fault is intermittent, there is always the probability that it will not be present when the tests are run, or it might be present during the check-out stage which would cause a transfer to the fault-location routine but at that time there may be "no" fault to locate. The only approach to the problem of intermittent faults appears to be some form of repetitive testing, but the complications posed by all of the probabilities involved are obvious. ARIES, Inc., is also doing some sork in this area [3].

An area in which a great number of different assumptions have been made is in the nature of the fault or component failures. The assumptions discussed above appear to be quite reasonable and do not detract very much from the diagnostic program but those made in regard to the nature of the possible faults greatly limit its value and usefulness - as well as have a great effect on the effort required to produce it. Failure data and the probabilities involved certainly justify the assumption that diodes rarely short-out as opposed to becoming open. It is usually much more difficult to "diagnose" a shorted diode and for this reason they are often just "assumed away". However, the real power of a diagnostic lies in its ability to locate such failures which would otherwise be left to the operator or maintenance man to find. Often, it is possible to obtain a good appraisal of the value or quality of the diagnostic by how many and which types of failures are "assumed away".

A final assumption that is frequently made, although it is certainly not as restrictive as those just mentioned, is that the machine has been assembled properly in accordance with the documentation on which the diagnostic was based. The diagnostic may be able to detect and locate wiring errors on the back panel or other assembly errors, but it is usually not written with this use in mind and may give erroneous indications in such a situation. Even if the diagnostic results in a "fault-free" output, the machine may still contain assembly errors.

5. Signals are Tested - Not the Elements Themselves

All of this discussion about checking-out or verifying the failure condition of the machine and locating the failure might give the impression that the actual components or at least the logic elements are tested in some manner akin to using a voltmeter to check a voltage, an ohmmeter to check a resistor or diode, etc. This is certainly not done if for no other reason than that the time required would be prohibitive. What is done is that the ability of a computer to compare the value of a signal obtained during a test run against a control value permits inferences to be drawn as to the condition of the hardware that produced the signal. During check-out, the control value would be that obtained with a machine known to be defect free. During the execution of fault-location routines the control values may be those associated with the different failures possible. The important point is that the signals in the computer are checked-not the hardware itself. Since this is the situation it is usually not possible to determine whether the failure occurred in signal generation, signal transmission, or signal reception without making manual tests with test instruments or using substitutional procedures.

6. Diagnostic Program vs Redundancy Checking

The entire concept of using a diagnostic program to both check-out the machine and then, if necessary, locate the faults present should be contrasted with the technique of using two systems to obtain fault detection by redundancy. It may be that almost the entire system is duplicated as in some of the SAGE configurations, or just certain critical portions of it are redundant. In this situation the duplicate equipments can solve the same problem and then compare the results. This may be done on the macro scale, i.e. only for the overall result, on the micro scale, i.e. at each logic element output, or at some intermediate level. The additional expense of such a system is usually not warranted solely to replace the diagnostic procedure. If the two systems do not agree at some point, a fault has been detected, but its location is not known. In fact, without some other test means, it is not even possible to say which system contains the fault; and, unless the comparisons are made close to the ultimate micro level, little is known as to which part of the system contains the fault. Although redundant systems can not be justified to completely replace the need met by a good diagnostic program, they do provide the capability of rapid restoration of service.

## B. ROLE OF DIAGNOSTICS IN MAINTENANCE PLAN

### 1. Distribution of Maintenance Times

In a normal commercial environment the distribution of service call durations will be of the form shown in figure 2a [4]. This figure does not reflect the use of any maintenance aids such as a diagnostic program. The fixed increment of time at the origin of the curve is attributed to the time required for the maintenance man to arrive at the computer site, and it is this increment of time that often forms the major portion of the "down time" of the machine.

In any situation the goals desired of a maintenance system are:

(1) Drastically reduce the maximum length of down time.

(2) Substantially reduce the median value of down time.

(3) As a result, the mean down time will also be reduced.

It should definitely be emphasized that these efforts must be made within a group of constraints:

(1) Cost.

(2) Availability of maintenance personnel and their abilities.

(3) Availability of replacement equipment.

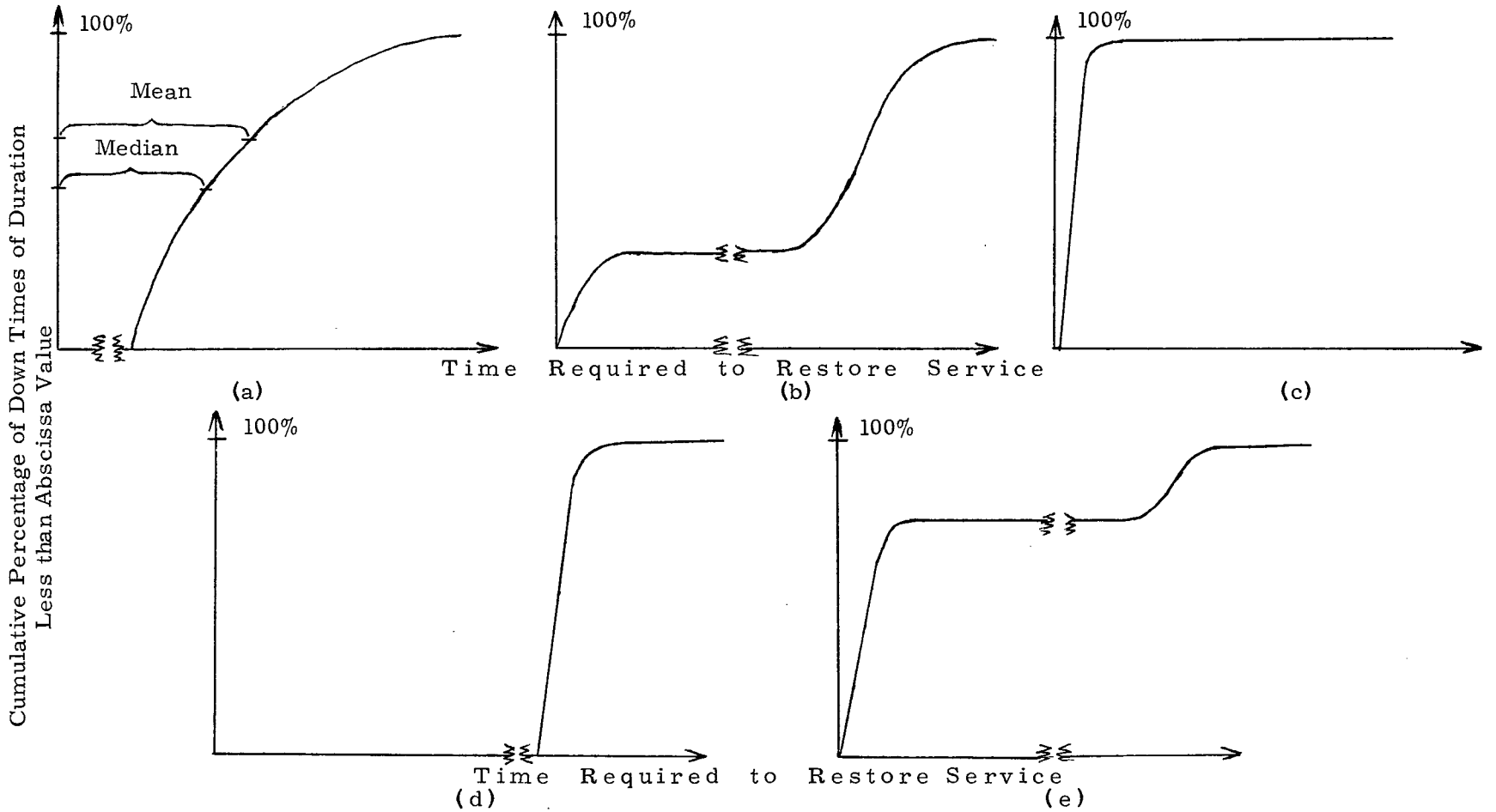



Fig. 2: Distribution of Maintenance Times for Various Maintenance Schemes
(a) Normal Commercial Environment
(b) Effect of Operator as Maintenance Man
(c) Effect of 100% Maintenance Float at Each Operating Level
(d) Effect of Maintenance Float at Shop
(e) Effect of Availability of Diagnostic Program and Maintenance Float at Shop

2. Methods of Decreasing Down-Time

a. Train Operator as Maintenance Man

The first logical step would be to train the operators in the basics of maintenance, but the limitations on time and funds would limit this training, and the operator would be able to correct only the simplest defects. (This still assumes no diagnostic program available.) The distribution of times necessary to restore service would now be something like that shown in figure b. The operator is able to commence his maintenance efforts immediately, but his talents are quite limited, and the long horizontal step in curve is caused by the travel time required by the maintenance man.

b. Use of Maintenance Float

One technique used to drastically reduce the length of down time is the provision for a maintenance float for portable or mobile installations. A maintenance float is differentiated from maintenance parts in that the "float" is a complete assemblage that is used to replace the defective system or sub-system with no attempt being made at repair. If it were economically feasible to do so, the availability of 100% maintenance float at each operating position would certainly improve the down time picture. This procedure might even be able to produce a distribution such as shown in figure c, but a complete evaluation of this approach to the problem would have to consider the following:

(1) Cost of 100% duplication of equipment.

(2) Mean time until failure of the float item both in use and in standby.

(3) The time and costs involved in replacing float items put into use.

The cost picture can be greatly improved by retaining the float equipment at the maintenance shop, but then the time performance will suffer as shown in figure d, because of travel times involved.

c. Diagnostic Programs

A primary effect of making available a good diagnostic program to the operator of the computer will be to greatly increase the percentage of troubles that he is able to remedy. If the failure is one that the operator is not able to locate with his diagnostic program, then it will probably require a large amount of time by the maintenance man, so the diagnostic program should be backed up with a maintenance float at the maintenance shop level to obtain a down time distribution such as shown in figure e. Some of the aspects of this procedure that will have to be examined are:

(1) Is it possible to develop a diagnostic program that will locate a substantial percentage of the failures?

(2) Cost of the preparation of the diagnostic program.

(3) Additional training required for the operator in order to be able to use the diagnostics.

(4) Requirements for special test equipment and maintenance items at the operating location.

(5) Provisions for repair or replacement of maintenance items put into service.

(6) Requirements for float equipment at maintenance shop level.

(7) Quantity, disposition, and staffing of maintenance shops.

## II. DIAGNOSTIC PROGRAM PREPARATION

### A. GENERAL CONSIDERATIONS

The approach to be taken in analyzing the computer and preparing its diagnostic programs should be selected only after considering many factors, some of which are given below:

(1) What is the basic organization of the machine?

(2) What type of logic is used (AND-OR-NOT, NAND, etc.)?

(3) Is it synchronous or asynchronous?

(4) How is the logic implemented?

(5) What are the mechanical features, size, number of elements, etc., of the replaceable units?

(6) What is the availability of spare units for use in substitutional testing, etc?

(7) What are the resolution requirements?

(8) What is the availability of test equipment either built-in or external?

(9) Is there any special hardware that will assist in the diagnostic process? (e.g. a comparator)

(10) What are the programming capabilities available (instruction list, stored-logic, etc.)?

(11) How much manual intervention is acceptable for both start-up and fault-location?

(12) Are there restrictions on the permissable length of the diagnostic (memory limitations, input problem, etc)?

It has been observed that an experienced team will greatly change the approach followed for two different computers because of variations in the factors listed above.

Of all of these considerations, the one that appears to have the greatest effect on the general philosophy of the diagnostic program is the size of the replaceable unit. The Honeywell ALERT is a small airborne computer which uses a rather simple diagnostic to tell which unit is bad (power supply, central processor, input/output, or memory). The decision here was that complete self-diagnosis was not practical [5]. The effect of integrated circuits will also be quite great. One example is the Litton L-304 which has 400 integrated circuits per plug-in board [5].

Of course, the ultimate goal of all of the various techniques employed is to check all of the machine. The differences lie in what is used as the starting point and the method used to select the next area to be tested. All of the methods described below are

not necessarily practical although they all appear to work in theory. References are given to several projects using some of the systems.

## B. ORDER OF TESTING

As pointed out earlier, the first step in the diagnostic procedure is to verify the assumptions made for the automatic check-out routines that the computer is "breathing", i.e., that a program can be placed into memory and that the control unit can operate through a complete instruction cycle. This condition is essential before the program continues.

If there are not any tight restrictions on manual operations, one technique that can be used is a step by step manual check using the control console or maintenance panel. This would allow a check on the controls as well as the loading of the program, the timing generator, and the cycling.

If manual procedures are undesirable, it is also possible to make a gross check using an automatic or programmed check. The problem here is that if the automatic check fails, then it will usually be necessary to now perform the extensive manual procedures to locate the fault, whereas when the manual check procedures are used the exact point of failure gives a large amount of fault location information.

After the breathing machine has been verified it is necessary to move on through the check-out routines in some systematic manner. There has been some work done on the selection of test sequences to locate failures in complex equipment [6]. These works, which make use of the probability of failure, usually apply to situations in which it is possible to apply a known stimulus to a component and observe the results. Because the components or assemblies to be tested in this situation are embedded within the computer, it is not possible to be sure that the correct stimulus is applied or that the response obtained is correctly transmitted to the sensing device unless certain other tests are made first. Primarily for this reason, it is usually not possible to make use of such studies; although, if the failure probabilities were known it should be possible to reduce the <u>average</u> test time required.

Some more useful methods for determining the order of testing are given below.

## C. CHECKOUT PHILOSOPHY

### 1. <u>Command Check-Out</u>

Probably the oldest procedure employed to determine the test sequence was a series of tests to verify the correct operation of each command. Here consideration is not given

to the logic circuits involved, but merely to the output results obtained with given data inputs. Of course, the only way to have the computer do anything is to give it commands to execute, but here the emphasis is on checking at the command level and the order of testing is usually by functional groups of commands -- arithmetic, input, logic and control, output, etc. In these tests the normal technique is to repeat each test several times which is of no value with a solid fault and of questionable value when it is an intermittent. The test itself consists of executing an operation such as ADD and then comparing the results obtained with the known answer entered as part of the input data. Also used is a test that generates random numbers on operands and then checks the results obtained by an inverse operation - ADD and SUBTRACT.

One real weakness in this approach is the quantity of testing required to make a thorough check. Consider for example the test of a parallel adder with a word size of only 10 bits. Since the analysis is confined to the command level, it is necessary to use all $2^{10}$ or 1,024 bit patterns as operands since it is possible that just one of these will cause the error present to appear. The use of random operands will not insure getting all possible patterns, and for large word sizes the problem becomes acute. Command oriented tests using exhaustive testing have not been seen.

Another weakness of this approach is the small amount of information obtained about the location of a fault after an error has been detected since each command usually makes use of a large proportion of the machine. One refinement proposed was a use of the command check-outs coupled with a dictionary of fault effects as follows [7]:

(1) Run all tests noting instructions that failed and contents of registers after failure.

(2) Find the possible fault(s) that could effect all of the various instructions that failed.

(3) Check the pattern of register contents to determine which fault it was.

It is usually necessary to use a command level check-out for the "breathing" machine tests, but these checks only verify that the command is operative enough to produce the "breathing" machine -- i.e., no gross malfunctions. There may still be undetected faults in the circuits associated with these commands that passed the limited tests given. Some combinations of commands that have been used for this start-up purpose are:

| RCA MICRORAC [8] | MOBIDIC [7] | 465-L [9] |
|---|---|---|
| Clear and Add | Clear and Add | Clear and Add |
| Transfer Unconditionally | Subtract | Exclusive OR |
| Read Alphanumeric | Store | Store |
| Write Octal | Transfer on Negative | Transfer on Plus |
| Write Plphanumeric | Halt | |
| Sense and Set | | |
| Sense and Reset | | |

2. Micro-Instruction or Subcommand Check-Out

The next level below the command or instruction level is the micro-instruction level. A micro-instruction causes the computer to execute a unique operation such as "Transfer Memory Buffer Register to Exchange Register" and the diagnostic program is based on the general premise that each command makes use of a unique set of micro-instructions. A portion of the chart giving the micro-instructions that constitute each command of the BI-TRAN SIX is given in figure 3. (The numbers in the chart give the timing pulse on which the micro-instruction is executed, and the parentheses indicate that the execution is dependent on other conditions.) It can be seen that most commands use several different micro-instructions, the only exception being Complement Accumulator Sign (excluding routine housekeeping functions).

Since the hardware involved in generating and executing a micro-instruction is a smaller unit than that associated with the complete command the size of the "new area" under check-out is reduced. It should be noted that even at this level the hardware associated with each micro-instruction is not necessarily unique just as with the complete commands. Of course, the smaller size of the check-out units provides more information at the start of the fault-location process.

The number of micro-instructions in the machine is usually quite reasonable since a versatile command repertoire can be developed with just a few. The very small size BI-TRAN SIX has 62 micro-instructions while the stored Program Element of the Air Force 465- L complex had only 65 not counting in the latter case, those that caused gross errors and a non-breathing machine [9].

The discussion in this paper would tend to make it appear that the only order in which to check-out the commands would be from the least complex (considering the number of micro-instructions) to the most. Although this does appear to be the most useful method, there is nothing to prevent using a different order if the situation warrants.

## BI-TRAN SIX MICROINSTRUCTIONS AND TIMES AT WHICH EXECUTED

| | | | Notes | Acq | SBJ 10 | UNJ 12 | NAJ 14 | NZJ 16 | MNI 20 | MNO 22 | EXI 24 | EXO 26 | LDA 40 | LDN 42 | ADD 44 | SUB 46 | RAU 50 | RSU 52 | MPY 54 | DIV 56 | STA 60 | STQ 62 | SAR 64 | SAL 66 | LDC 70 | LCT 72 | CAS 74 | STP 76 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Shift contents of AM and QM regs left 1 | AQL | | | | | | | | | | | | | | | | | | 07 | | | | 14 | | | | |
| | Shift contents of AM and QM regs right 1 | AQR | | | | | | | | | | | | | | | | | 15 | | | | 15 | | | | | |
| | Complement AE | CAE | | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| | Complement each bit of A reg less sign | CAM | | | | | | | | | | | (06) | (06) | (06) | (06) | | (06) | | (04)(06) (12)(15) | | | | | | | | |
| | Complement A sign | CAS | | | | | | | | | | | (06) | (06) | (06) | (06) | | (06) | | | | | | | | | (06) | |
| | Clear A reg | CLA | | | | | | | | | | | 00 (07) | 00 (07) | (07) | (07) | 00 | 00 (07) | 12 | | | | | | | | | |
| | Clear $A_0$ | CLAØ | | | | | | | | | | | | | | | | | | | | | | (02) | | | | |
| | Clear $C_6$ | CLC6 | | | | | | | | | | | | | | | | | | | | | 02 | 02 | | | | |
| | Clear D reg | CLD | | | | | | | 15 | 15 | 15 | 15 | | | | | | | | | | | | | | | | |
| | Clear External Light | CLEL | #2 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| | Clear I reg | CLI | | 02 | | | | | | | | | | | | | | | | | | | | | | | | |
| | Clear Instruction Error | CLIE | | 02 | | | | | | | | | | | | | | | | | | | | | | | | |
| | Clear M reg | CLM | | 00 17 | 00 11 | | | | | | | | | | | | | | | | | | | | | | | |
| | Clear $M_0$ | CLMØ | | | | | | | | | | | | | | | 12 | | | | | | | | | | | |
| | Clear Overflow | CLOV | #2 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| | Clear P reg | CLP | | | 05 | 05 | (05) | (05) | | | | | | | | | | | | | | | | | | | | 05 |
| | Clear Q reg | CLQ | | | | | | | | | | | | | | | | | 00 | | | | (00) | | | | | |
| | Clear $Q_0$ | CLQØ | | | | | | | | | | | | | | | | | | 00 | | | | | | | | |
| | Clear Sign | CLSI | #2 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| | Clear X reg | CLX | | 01 13 | 07 | | | | 01 | 01 | 01 | 01 | 01 | 01 | 01 | 01 | 01,07 13 | 01 07 | 01 07 | 01 | 01 | 01 | | | 01 | | | |
| | Clear Logic Zero | CLZ | #2,5 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 06 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| | Complement each bit of X reg less sign | CXM | | | | | | | | | | | (04) | (04) | (04) | (04) | | (04) | | 02 | | | | | | | | |
| | Complement X sign | CXS | | | | | | | | | | | | 02 | | 02 | | | | | | | | | | | | |
| | Decrease count in C reg by 1 | DEC | | | | | | | 13 | 13 | 13 | 13 | | | | | | | 13 | 13 | | | 13 | 13 | | | | |
| | Initiate Add cycle | INA | | | | | | | | | | | 05 | 05 | 05 | 05 | 05 | 05 | (14) | 03,05 10,(14) | | | | | | | | |
| | Increase contents of A reg by 1 | INCA | | | | | | | | | | | (07) | (07) | (07) | (07) | | (07) | | (14) | | | | | | | | |



Figure 3: Micro-Instruction Chart, Fabri-Tek BI-TRAN SIX

3. Logic Term Check-Out

The next step below the micro-instructions would be the logic terms -- those term used in the logic equations defining the operation of the machine. Although the micro-instructions usually are logic terms appearing as both operands and outputs, the total list of logic terms is much longer and more detailed. Again using the very small BI-TRAN SIX as an example, there are 24 commands, 62 micro-instructions, and about 750 logic elements exclusive of memory drivers, sense amplifiers, etc. The test procedure here is based on the simple premise that if all of the logic terms function properly, then the machine is fault-free.

Two problems face the programmer using this approach. The first of these is that a logic equation is usually a combination of both AND and OR operations and therefore, requires more than one level of logic element hardware to implement it. The other problem is that the logic equation itself is easily used to "look back" to determine what input terms are used to develope the term under test, but it is quite difficult to "look forward" to determine how a failure in the term under test will effect other logic terms. Both the relation-to-hardware problem and the look-forward problem can be alleviated by special documentation techniques described elsewhere [10].

4. Hardware Component Check-Out

The approach employed to produce some of the best diagnostics that the authors have seen is based on the lowest level possible - each hardware component. The first step using this technique is to determine what tests will be sufficient to verify the proper operation of the smallest hardware assemblage that can be examined, since every element is not diagnosible.

The authors define a "diagnosible signal" as one which is used as an input to two or more elements which belong to at least two different diagnosible trees. A "diagnosible tree" is then a collection of interconnected logic elements as shown in figure 4, whose final output is a diagnosible signal and each branch of which terminates when it reaches an element for which all inputs are diagnosible signals. Without considering in detail the nature of the logic elements in the tree, it is possible with fairly simple tests to determine that some element within the tree is faulty, assuming that only single component failures exist. Such tests would consist of applying "known" diagnosible signals to the inputs of each branch and comparing the output diagnosible signal with the "correct" value. By definition, these diagnosible signals may be independently verified.

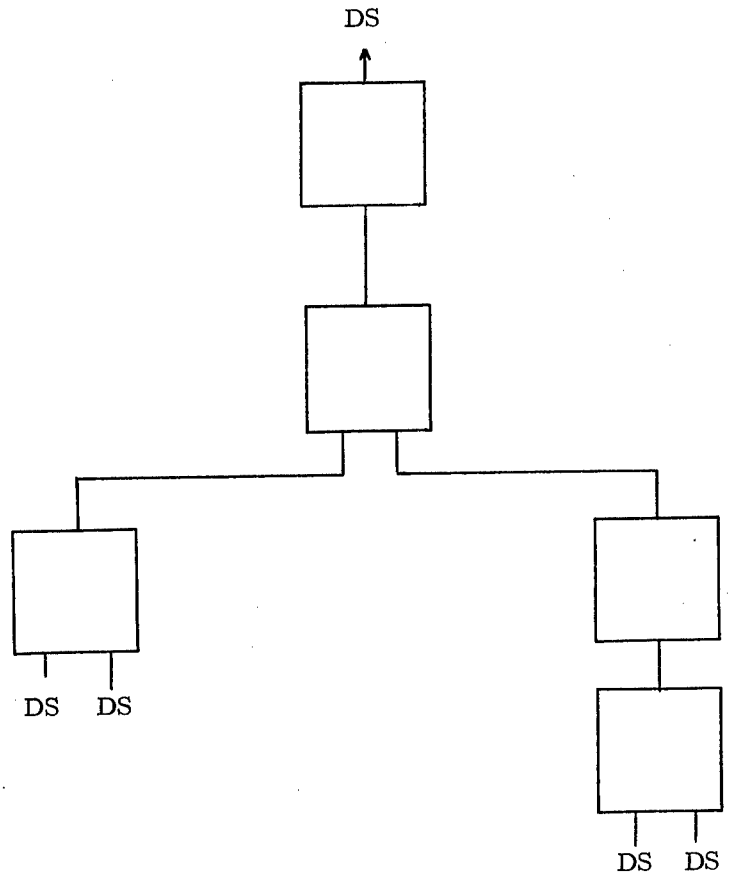



Figure 4: A "Diagnosible Tree"
(DS indicates Diagnosible Signals)

Resolution can be improved if we further define a "diagnosible assembly" as a subset of elements within a diagnosible tree, the output of which is either a diagnosible signal or an OR gate with two or more inputs (in the latter case the input section of the OR gate is included) and each branch of which is terminated by an element whose inputs are all diagnosible signals, or an OR gate with two or more inputs (OR gate output section included). An example of a diagnosible tree subdivided into diagnosible assemblies is shown in figure 5. Since the function performed by a logic gate is dependent upon whether positive or negative logic is used, it may be possible to subdivide the same diagnosible tree into two different diagnosible assemblies and hence gain a further improvement in resolution by treating the tree as both negative and positive logic elements separately.

The concepts advanced here may not be essential to the functions of a pure check-out routine, but they are the limiting factors of fault location resolution. Normally it will only be possible to localize the fault to being located within a diagnosible assembly.

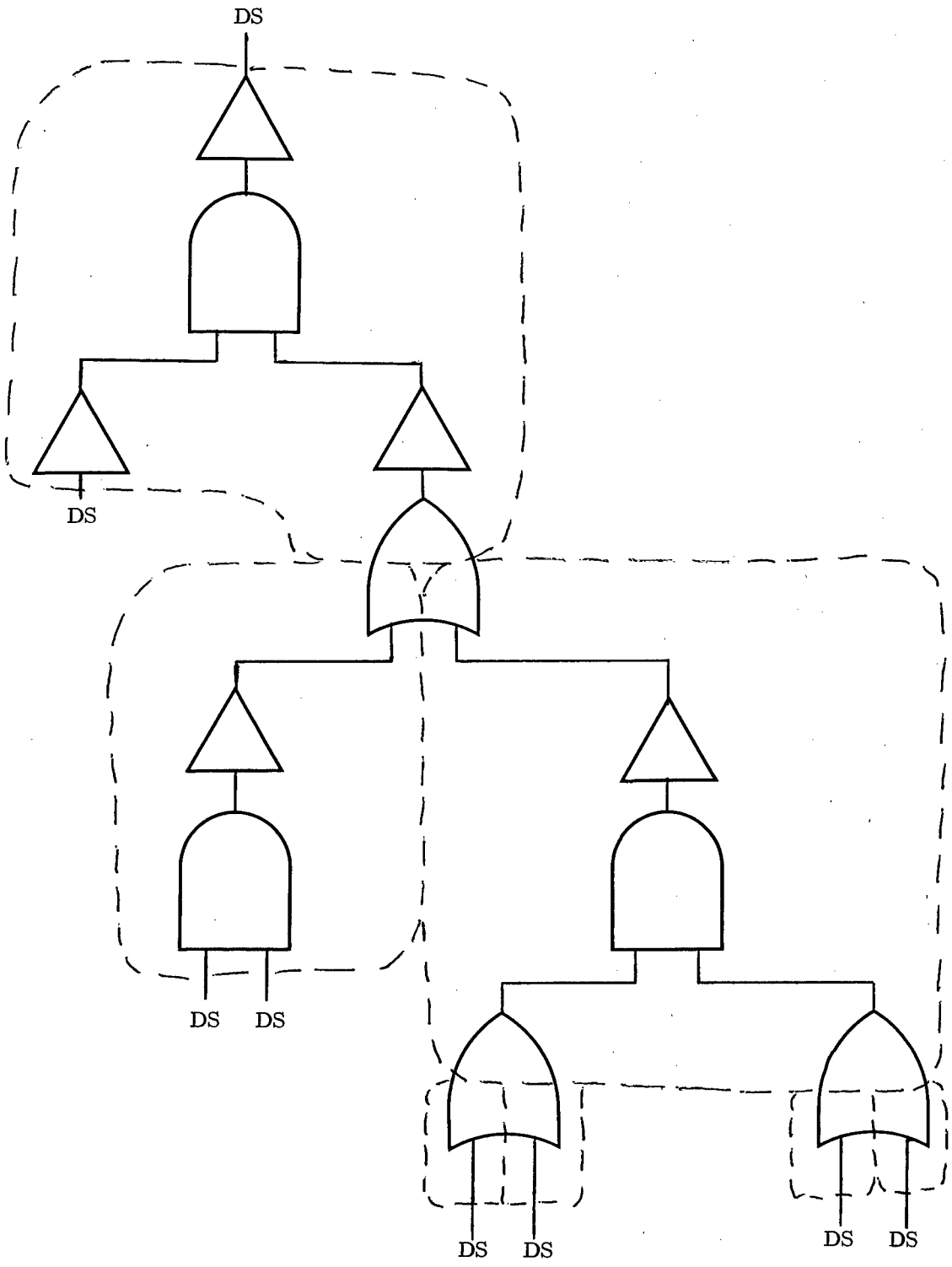



Figure 5: A Diagnosible Tree Subdivided Into Seven Diagnosible Assemblies

After the check-out routines are written, the fault-location procedure must be developed. Each component is now examined to determine the effect of a catastrophic failure such as open diode, shorted transistor, etc. This component is part of a logical element such as a gate or a storage element such as a flip-flop, and the next step is to determine a program for use in fault-location that will detect the error caused by the assumed failure. The very small size of the "new area" under test at each check-out stage provides a great amount of fault-location information, but this added benefit is balanced in part by the complex analysis required to determine the effects of feed-back when faults actually do exist. (A "simple" example of this type is discussed in the appendix.)

When the check-out routine for a given tree is written it may often be found to be equally applicable to another. The same situation may also occur for many of the individual fault conditions. These possibilities coupled with the large number of test and fault-location routines required * present a problem of their own in bookkeeping.

5. Timing Oriented

Although it does not appear to show much promise, it would be possible to use an approach based on checking out the proper execution of all of the micro-instructions that should occur on a given timing pulse. Again, the test would sequence through each timing pulse until all had been shown to be error-free. For the conventional machine with a normal command structure, this does not appear to be very practical, but with special computers such as a stored-logic machine it might prove to be quite appropriate.

D. OTHER APPROACHES

1. "Diagnosing" or "Gedanken Experiment"

When studying this subject the reader will often be enticed by titles such as "The Diagnosis of Circuits", but when the article is examined it is found to apply to the area

* A three-input diode logic AND gate will require four sets of test conditions to completely verify its operation, and there will be eight different fault conditions to be analyzed and used as the basis for the preparation of fault-location routines.

of sequential switching theory known as machine identification experiments or the Gedanken Experiment. The basic idea of the theory is that, in response to a sequence of stimuli, a sequential machine will change from one unique state to some other one in predictable manner. For the fault-free machine, this sequence of states is known and it is possible, in theory, to determine what it will be for the "new" machine created by each component failure. The practical problem of such an approach is primarily the enormous number of "new machines" that have to be analyzed completely as well as the requirement that often applies to such experiments that it be possible to place the machine in a known "reset state" to start the test sequence. The necessity to examine the entire sequence of states and to compare that sequence with all of the possible ones would probably present an insurmountable data processing problem. And finally, there is the problem that it may be necessary to use different sequences of input stimuli to distinguish between certain faulty machines.

As a general rule, the work in this particular field is not of too much value to the preparation of practical diagnostic programs.

2. Pattern Recognition

Just as in the use of a Gedanken Experiment, pattern recognition techniques are faced with the problem of an enormous number of "faulty-machine" patterns. Two other problems also are common to the two techniques - how do you sense and record the "pattern" produced and what input stimuli are used? The work done by the Bell Telephone Laboratories on diagnostics for the No. 1 ESS (Electronic Switching System) might be considered to fall into this area [11 and 12].

In the ESS the problem of sensing the pattern is solved by using the operative computer of the duplexed-pair found in the Central Control. These two computers are cross-connected at several intermediate levels which permits using signals other than just the normal output ones to form the pattern. The faulty-machine patterns were determined by physically inducing a fault in a given card and noting the results when the input sequence was applied. There were 50,000 faults simulated. These produced only 10,315 different patterns of which 73% (app 7500) were unique to a single possible fault and 13% (1,341) indicated two possible faults. The limitations of this system might be illustrated by some further data on the ESS tests. The input sequence consisted of approximately 1000 different tests. With approximately 20,000 of the simulated malfunctions no tests were failed, and 2800 of the malfunctions caused the central control to "lock-up" and stop cycling. And finally there were inconsistent patterns produced by some malfunctions. These appeared

to be caused by seemingly inconsequential factors such as the condition of input-output lines although they are not used, the starting condition of the machine, etc. (Note the comment above about the requirement for a known starting state.)

Some further work on pattern recognition was done using the ESS test patterns [12] in a study based on approximately 11,000 "valid" patterns which examined the possibility of a geometric interpretation of the patterns. When the patterns were plotted in 6 dimensions using appropriate weighting based on the discriminator power of that test, cluster groupings did occur.

## E. FAULT TYPES AND FAILURE ANALYSIS

The basic nature of the signals present in digital equipment is that they are quantized into just two values - "binary signals". These two states are referred to by several different sets of names - "high-low","0-1", "true-false", "active-inactive". Although the exact value of the actual signals present may vary, a threshold level is always present that divides the two states. Some of the work in this field has assumed that each failure caused a fault classed as either "stuck-at-one" or "stuck-at-zero". This is definitely not the situation as is discussed below.

### 1. Nature of Faults:*

The two fault conditions that may occur are:

(1) a signal is not active when it should be, or

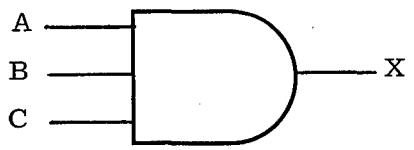
(2) a signal is active when it should not be

Consider the AND gate shown in figure 6. If all input signals, A, B, and C, are active (true) then the output X should be active. For the OR gate, Y should be active if either D, E, or F is active. If these conditions do not occur, i.e.the signal is not active when it should be, the fault is called an "Activate-Fault" or an "A-Fault".

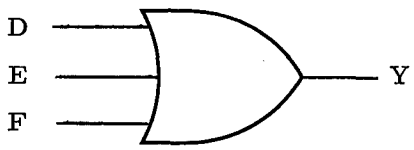
The output of an AND gate should be inhibited (prevented from being active) when any input is inactive. The OR gate is inhibited only when all inputs are inactive. If these conditions are not met, then signals will be active when they should not be, and the fault present is called an "Inhibit-Fault" or "I-Fault".

---

* Names follow those of Eldred [13]




a) AND Gate




b) OR Gate

Figure 6: Basic Logic Gates

2. Relative Frequency and Importance of Activate-and Inhibit-Faults:

One analysis of failure modes and their effects [14] went to great lengths to determine the probability of failure of each component to determine the relative frequency of activate and inhibit type faults. An examination of the Appendix will show that if the design of the AND and OR gates is symmetric, then the occurence of A-Faults and I-Faults in a single gate is symmetric also. The normal situation is for a computer to have many more AND gates than OR's so that the I-faults might be the predominate type. The percentage of A-Faults vs I-Faults expected does not appear to have any real significance. What is important is the effect of the two different types of faults.

Activation-Faults, i.e. the absence of a required signal, are usually detected and located without any great difficulty. This characteristic stems from the fact that the check-out routine normally activates only a small "new" or "unchecked-out" portion of the hardware on each test. The number of possible A-Faults (as well as I-Faults) in the position under test is usually small and their effects are predictable. The check-out routine is designed so as to verify all the activation and inhibit actions within the "new position". On the other hand, an I-Fault in a portion of the machine to be tested later can create quite a problem.

Even if an Inhibit-Fault is "solid and steady" it may not be detected during several of the check-out steps, and when it is finally noticed its effect may be to completely "confuse" the test program. An example of this would be a fault in the adder control section that generated a permanent "Initiate Add" signal. This might cause an add action

on every clock pulse for all operation codes, not just during the desired pulse during an "ADD" operation. As long as the contents of the accumulator were not examined the fault would probably remain undetected, but then even when it was examined, the results would be so far from anything predictable that the problem of fault-location would be extremely difficult.

Another condition that can occur perhaps more than the one just described, is when the I-Fault completely "clobbers" the operation of the machine. Here the number of examples is endless, but a few specific ones are:

(1) The timing pulse generator produces more than one timing pulse during one clock pulse (See figure 7. An inhibit failure of gate X-output stuck at its active level would cause the following timing pulse sequence to be generated: $\emptyset 0$ and $\emptyset 3$, $\emptyset 1$ and $\emptyset 3$, $\emptyset 2$ and $\emptyset 3$, $\emptyset 3$, $\emptyset 4$ and $\emptyset 7$, $\emptyset 5$ and $\emptyset 7$, etc.)

(2) A "Halt-Overflow Error" signal is continuously generated.

3. Failure Analysis

It is only through an examination of the specific circuits used in the equipment that it can be determined whether a given component failure causes an A-Fault or an I-Fault. At first, this may appear to be a simple task to perform; however, the interactions between various assemblies through feed-back paths greatly complicates it and necessitates experimental verification of any results. An example of fault analysis is given in the Appendix. It normally is not this simple.

The following factors can be used to describe a failure situation: [15]

(1) Duration, Frequency } Solid or intermittent

(2) Nature: logical or timing

(3) Number: single or multiple

(4) Functional position: control signal, data path, register

(5) Unit Position: Processor, memory, I/O, etc.

## F. PREPARATION TECHNIQUES

### 1. Manual Analysis

Of course, the most commonly used preparation technique employs "simple" manual analysis of the design of the machine. This technique is usable with any of the approaches or testing levels just discussed. There has been some use of automated aids such as file management routines for documentation; however, the cost of developing automatic programming aids can be prohibitive because of the special considerations required for each

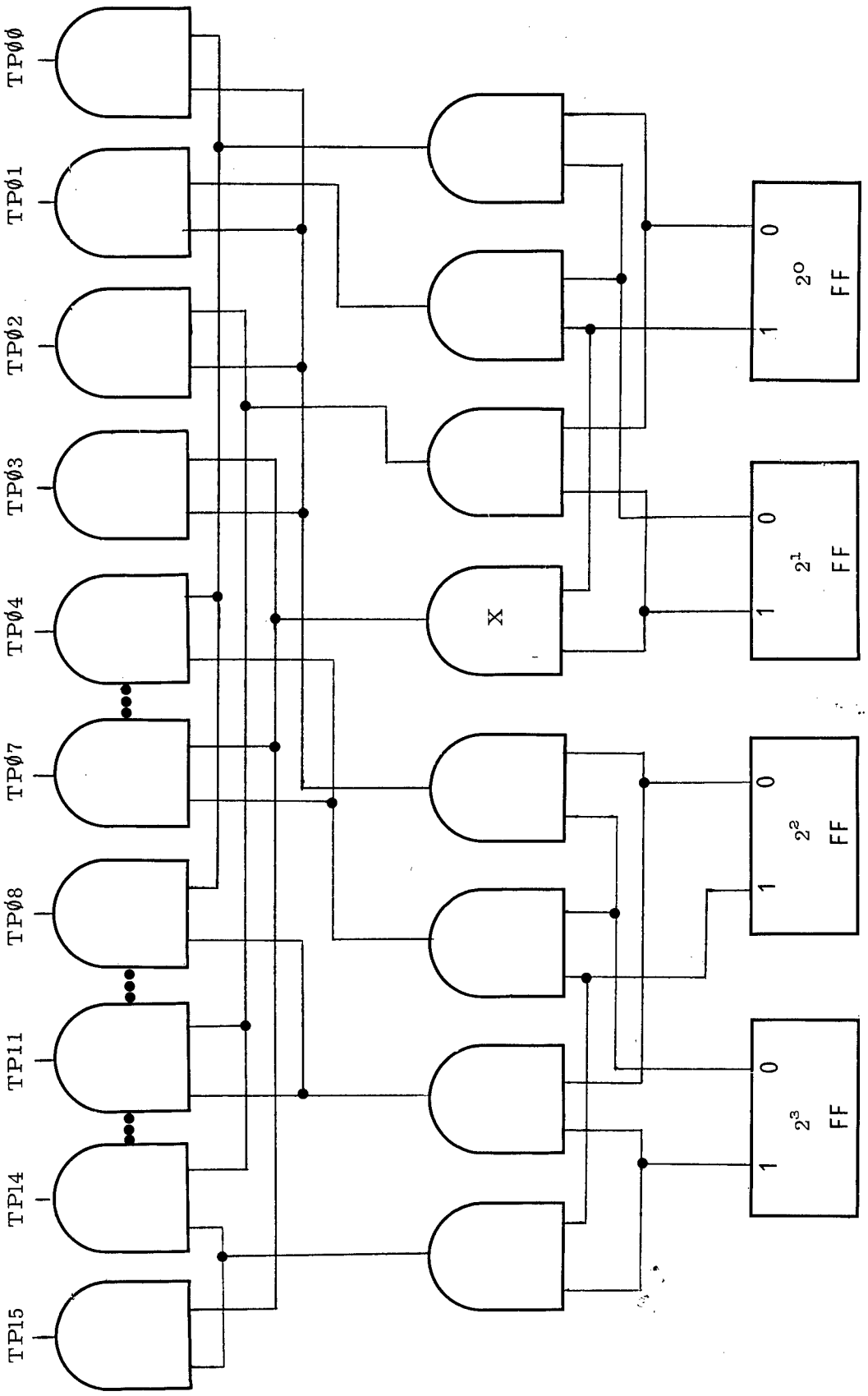



Figure 7: Timing Pulse Generator

type of logic implementation. The field of diagnostic programming does appear to be maturing now to the level at which automatic programming aids of general applicability would be feasible.

2. Physically Introducing Fault

The technique of actually introducing the desired fault into the machine was mentioned above under pattern recognition when the work by the Bell Telephone Laboratories on the No. 1 ESS was described [11]. The usefulness of this technique of diagnostic program preparation appears to be limited to approaches employing a pattern recognition after a standard set of tests.

3. Simulation

A simulator technique was used by ARIES, Inc. to prepare the diagnostic program for the AD/ECS-37 NASA computer. A Computer Logic Simulator (CLS) operating at the logic element level was developed. The Failure Generator (FAGEN) was then used to introduce the faults desired. This system checked for open diodes only. The CLS was then used to observe the effect of the failure. ARIES also expanded the CLS for use in analyzing the UNIVAC 422 with the simulator running on the UNIVAC 490.

Computer simulation was also used by UNIVAC for the Nike-X computer to catalog fault symptoms and to produce the dictionary to indicate which chassis was bad [5].

Seshu has used simulation extensively with apparently good results to develop self diagnosis methods for the CSX-1. The first simulator diagnosis program was run on the IBM 7090 and then an improved program capable of handling descriptions of sequential circuits of up to 300 logical elements, with 96 inputs, 96 outputs, 48 feedback loops and 1000 different failures was written for the CDC-1604 [16].

4. Automated

A pre-requisite to a large-scale effort for the automatic generation of diagnostic programs would logically be the opportunity to use the generator on several pieces of equipment. To do this requires that the hardware implementation of the logic be similar for the various equipments. This condition would certainly exist in a family of computers, and it is therefore not surprising to find that IBM has done work in this area on the System 360 series as well as some of the 1400 models [4].

The "fault locating tests" (FLT's) are produced by a series of programs operating directly on the Design Automation Logic Master Tape. "This program complex (a) produces the FLT data to test directly most System/360 CPU components in the larger models; (b) order these data so that the results of testing can be easily analyzed; (c) keeps these

data up to date with engineering changes in the hardware logic; and (d) produces up-to-date documentation, which in itself is sufficient to reduce the duration of many maintenance calls." [4]

Several organizations currently use some form of automated design system for their computers; a profitable extension to these systems would provide for the generation of diagnostic information as the design is formulated.

5. Problems in Preparation

Some of the more obvious problems in the preparation of diagnostic programs include:

a. Segmentation. The magnitude of the typical diagnostic program is such that it is always a team effort. This dictates a considerable task in breaking down the computer into sub units for which separate personnel will write programs, and an even greater task in coordinating between the teams. Decision Systems Incorporated normally subdivides by mechanical units and assigns two-man teams to each unit.

b. Bookkeeping. It is essential to keep detailed track of which elements have been diagnosed and what failure modes checked for as the work progresses. This is of particular importance when more than one man is working on the same unit.

c. Documentation. No computer program is worth more than the documentation supporting it and this is even more true in the case of diagnostic programs. Documentation cannot be effected after-the-fact but must be carefully considered at the onset of the project. Both the segmentation and bookkeeping problems already mentioned will complicate the documentation problems. The quality of the documentation will be one of the major factors in determining the overall quality of the diagnostic program.

## G. MAGNITUDE OF EFFORT

It is difficult to appreciate figures on the time required to develop a diagnostic program without being familiar with the capabilities of the program, the method of using it and the machine itself; but perhaps these examples will be of interest.

The AN/BRN-3 and AN/BRN-3A Navigation System used the AN/UYK-1 stored logic computer and another data processor of almost equal size as well as analog interfaces and input devices. There were approximately 220 cards with about 40 diodes per card. The design objective of the diagnostic was resolution to a single card which required the development of manual fault-location routines to supplement the automatic ones. The length of the program was 25,000 instructions, and it required 18-20 man years for preparation. Another project by the same company on a medium sized computer required only 6-8 man

years of effort. One big reason for this great reduction was that the resolution specification was greatly relaxed.

The program for the RCA MICRORAC included over 7500 individual diagnostic tests and 22,000 words of instructions and data [8]. During the preparation of this program, which required approximately 25 man-years, it was found that a reasonable time allowance for analyzing, determining the symptoms, diagnosing and documenting the location of a single failure is one man-hour. [14]

## III. MACHINE DESIGN AND DIAGNOSTIC PROGRAMMING

### A. IMPORTANCE OF SYSTEM APPROACH

The requirements for serviceability and maintainability for any complex electronic system dictate that these factors be considered throughout the design process. It is usually quite difficult and often impossible to add service and maintenance features to a completed machine, but all too often this is attempted.

In a digital computer the catastrophic nature of failures greatly increases the maintenance problem. Usually failures occur with no warning, and after they do occur the machine may be completely useless. As mentioned earlier, the service restoration problem is further aggravated by the limited maintenance capability of the operator. Some possible solutions to this problem were also discussed earlier. The point to be emphasized here is the effect of a decision to prepare diagnostic programs.

The development of complete diagnostics is a major project that will certainly require almost as much effort as the design of the machine itself if they are prepared later by a new team. Concurrent preparation of the diagnostic programs can result in considerable savings, even though the final testing, etc., must await the availability of a prototype. Not only will the team already be intimately familiar with the design of the machine, but the feedback from the diagnostic programmer to the designer at this early stage may well yield a system with greatly improved maintainability.

Of course, the number of these machines to be produced and the nature of their use must also be considered. Another factor that will greatly affect the cost aspects of the problem is the effect of the availability of a diagnostic program on the requirements for maintenance assemblies and spare parts.

### B. EFFECTS OF HARDWARE DESIGN, IMPLEMENTATION AND ORGANIZATION

#### 1. "Hard-Core" or "Hang-Up" Problem

One of the basic assumptions made in the preparation of all diagnostic programs is that the machine is "breathing". That is, there must be no gross errors present that prevent the computer from getting the diagnostic program into memory and executing a complete instruction. If this cannot be accomplished, then the machine is said to "hang-up".

When the design of the machine is analyzed it is usually found that a very large proportion of the hardware is involved in performing those functions essential to the breathing machine. This "essential" hardware is referred to as the "hard-core", and the hard-core

has been variously estimated as 50% - 91% of the total hardware [17]. These figures are quite misleading and are sometimes based on erroneous reasoning. The most common fault appears to be classifying an entire logic element as essential even though it may be necessary for the element to operate properly in only mode, not in all possible ways. For instance, it may be impossible to clear an "essential" flip-flop, but if it correctly stays in the set condition during the breating machine tests, then the entire flip-flop should not be included in the hard-core.

Even though the computer has already executed several tests successfully, it is still possible for it to "hang-up". An example of this would be an activate fault on the control signal that terminates the add cycles during a multiply instruction. If this occured the machine would hang-up in the execute cycle. This type of problem can become of such importance that it is desirable to have a special signal generated by an internal clock to force an access cycle if one does not occur after some predetermined interval. This feature could be disabled by a maintenance switch if desired.

2. Input-Output Problems

A very bad problem area in the preparation of diagnostic programs is that of the input-output peripherals. This comment applies to both the check-out as well as the fault-location routines. This problem is the result of the length of the diagnosible trees associated with the input-output. Input/output device tests are usually not diagnostic in nature.

Consider the simple computer shown in figure 8. To perform an automatic check of the input-output devices, paper tape reader and paper tape punch, the procedure would probably have to be to punch an output tape, then place the tape in the reader to be checked for accuracy. Even if an error is detected, it is easy to see that it might have occurred at any point along a rather long signal path. Unfortunately this is just about the best that can be done.

For the larger computer, the problem is basically the same; however, it is aggravated now since the number of devices is usually much higher. Consider the system shown in figure 9. The "loop" to a magnetic tape unit can be closed quite conveniently by the tape itself since the unit is both an input and an output device. The input/output multiplexer permits "Simultaneous" operations through both I/O channels. Then a provision for a connection such as "a" will greatly facilitate the checking of the paper tape channels; however, such a procedure will not check the electromechanical operation of punching and reading. Devices such as the line printer, displays, keyboards, etc., will always require manual actuation and verification for testing. Provisions for closing the real time channels may

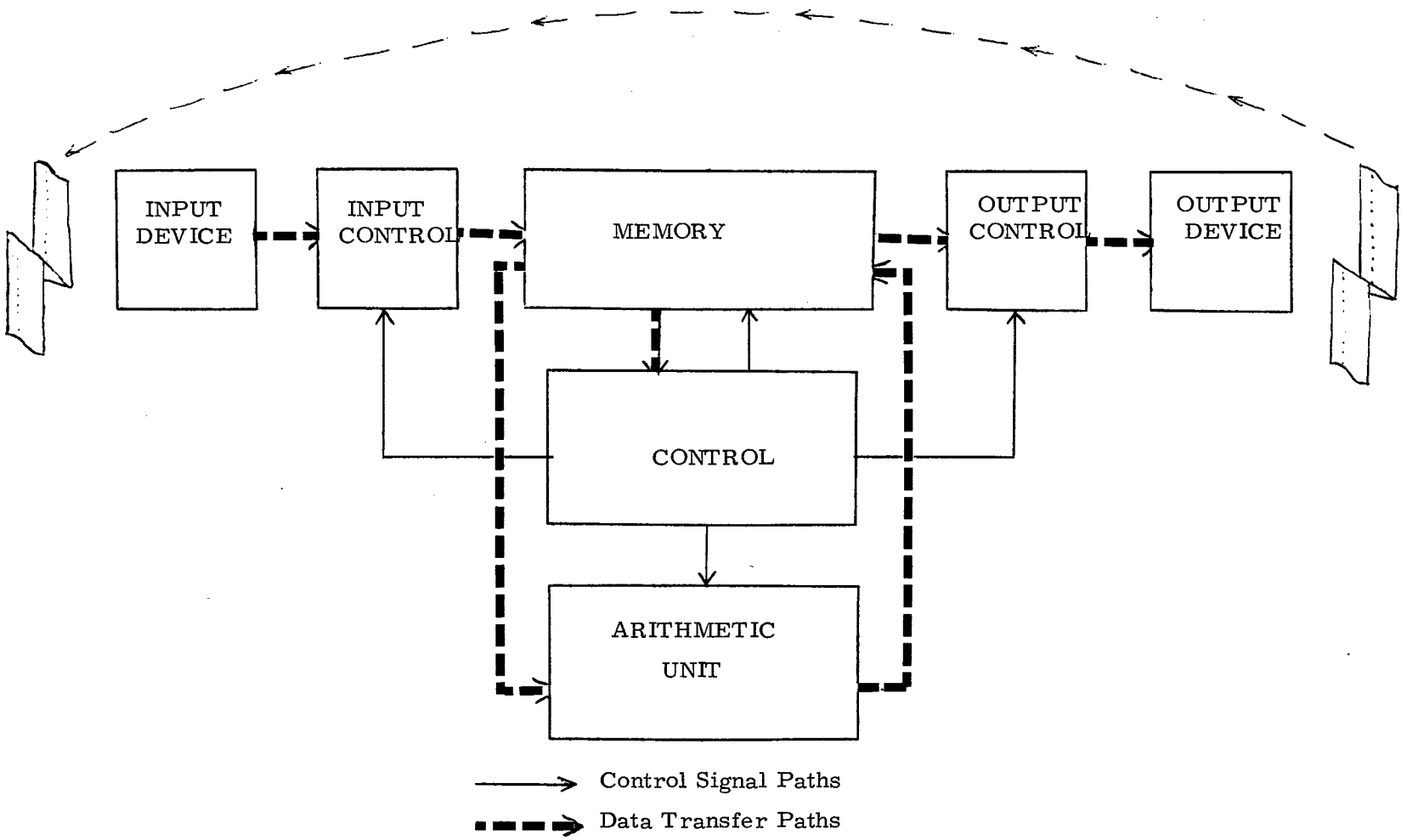



Figure 8: Computer Block Diagram Indicating Input/Output Loop

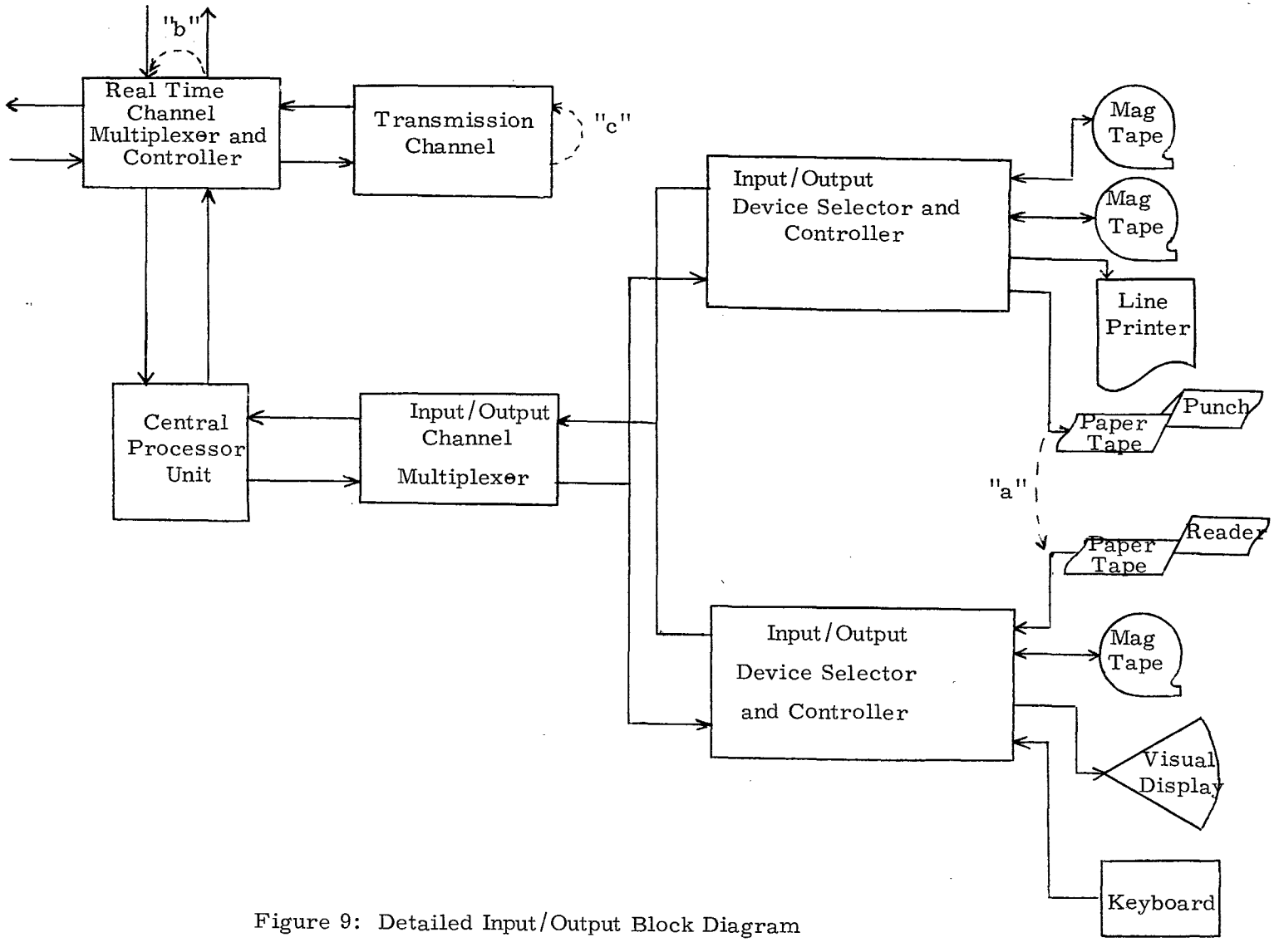



Figure 9: Detailed Input/Output Block Diagram

be at the near-end such as connection "b" or at the distant end connection "c", which will allow checking of the transmission channel and its associated equipment also.

When the entire system is considered, it may seem that it is necessary to specifically design the communication channels so that they may be connected "back-to-back" [8] or that special logic should be included to aid in performing the input/output tests [5].

3. Circuit Packaging - Vertical vs Horizontal Organization

Since their origin, digital computers have been assembled utilizing plug-in component assemblies. In some of the earliest models these were single vacuum tubes with their associated circuitry. Very soon the number of tubes per plug-in unit increased so that each assembly represented several logic or storage elements. The commonest practice was to have each unit contain several of the same type of elements such as AND gates, inverters, flip-flops, etc. This practice has been continued with the application of semi-conductor and integrated circuit devices. Although this technique may be quite advantageous for the design and manufacturing of the plug-in units it usually has quite a detrimental effect on the resolution capability of a diagnostic program. There have been many studies made on the problem of packaging with a goal of optimization with respect to one or more of the following: cost (repair vs throw away), complexity and usefulness, size, production yield, etc.; but none of these studies have considered the question of fault location during the maintenance process.

Consider the diagram shown in figure 10. Noting the predominance of groupings of four, which would undoubtedly be repeated many more times, the simplest packaging plan would be to make some cards with four AND gates, some with four OR gates, some with four flip-flops, etc. This is what has been named "horizontal organization". Since the only diagnosible signals in this example are the outputs of the flip-flops in the registers, it can be seen that this diagram contains four diagnosible trees. If an error is located in a specific bit of the X-Register and the corresponding bits of A and B both function properly, then the fault may lie in the X-Register flip-flop, in the corresponding OR gate or in one of the two AND gates. It may not be possible to refine this location any better and the repair instructions would specify the replacement of all four cards that might contain the fault. If the plug-in card had been "organized vertically" - one flip-flop, one OR gate, and two AND gates - then the location of the fault has already been resolved to one card.

The use of vertical organization will usually result in a larger number of different card types and this will often fail to comply with the contract specifications. The rationale of such specifications is to limit the variety of items in the supply systems, but an equally

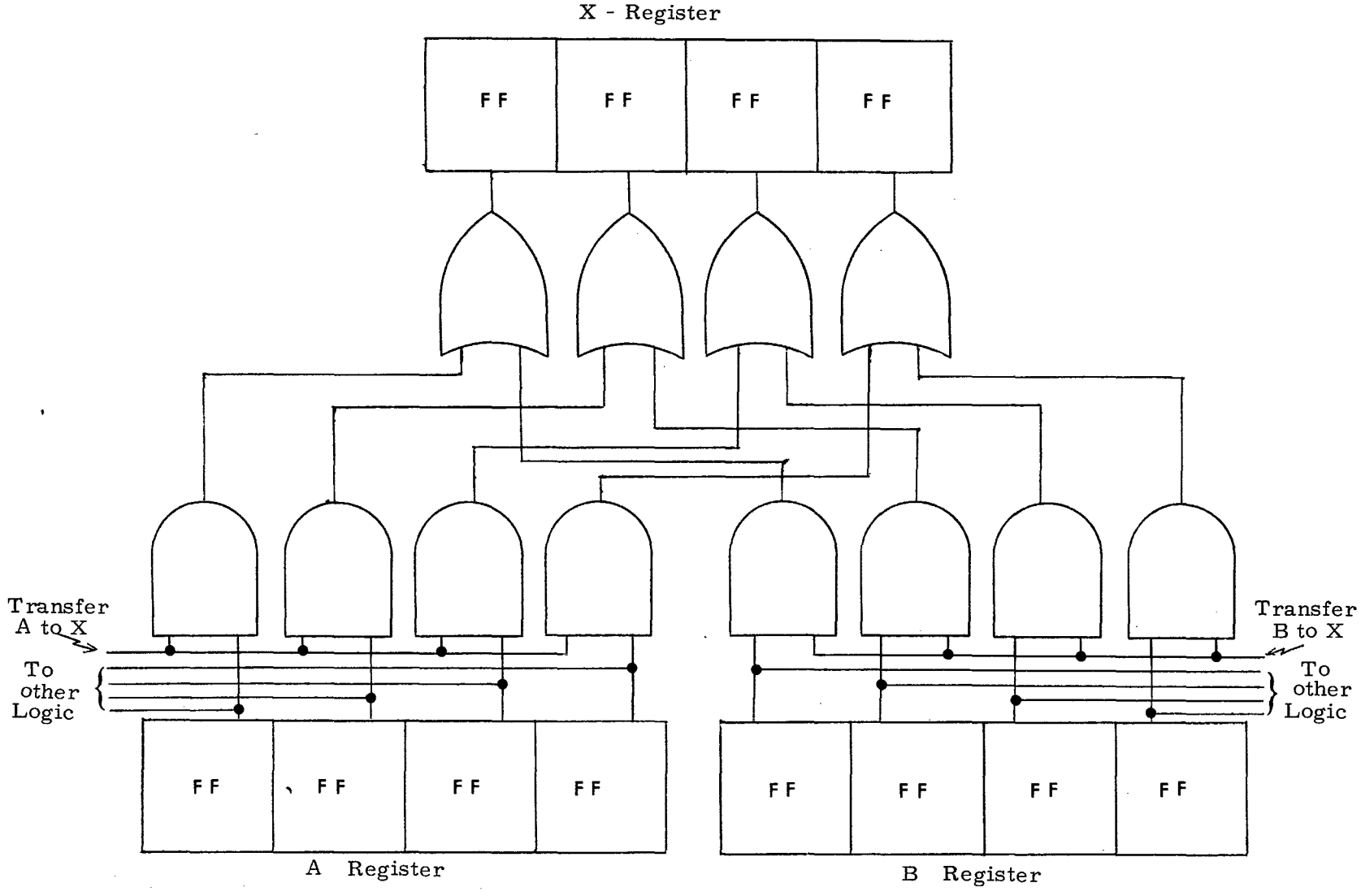



Figure 10: A Four Bit Register with Transfer Gates

important aspect is the quantities of each involved. If the computer is to be repaired by a low echelon operator-maintenance man he may not have a card-repair-testing capability. With horizontal organization he would have four "suspects" to be evaluated for repair. (He might be able to use a process of elimination to narrow the choice, but this would definitely cause a drastic increase in the down-time or mean-time-to-repair.) With vertical organization there would be just one card for evaluation. When overall mission requirements are considered, vertical organization might well result in a smaller quantity of required operating spares. The value of this approach has been recognized in a computer designed emphasizing ease of maintenance [18].

4. Multiple Use of Elements

The multiple use of a given element such as a logic chain or a register has both good and bad features. If the operation of the element can be checked under totally different operating conditions, the diagnostics capability may be enhanced. However, such multiple use will usually have very bad effects on the "hard-core" problem.

## C. HARDWARE AIDS TO DIAGNOSIS

1. Special Maintenance Commands

The ability to execute certain commands is found to be of great value in the preparation of diagnostics. Some of the commands which might be included primarily for maintenance uses are:

(a) Capability to test (sense) the state of all flip-flops [19]

(b) Capability to generate parity errors to exercise parity checking circuitry [19]

(c) Capability of direct register-to-register transfers for all registers. (Not necessarily all-to-all, but at least to one other under direct program control.)

2. Maintenance Panels and Test Points

The maintenance panels associated with many computers today, if they have one, represent almost a complete absence of engineering from the maintenance point of view. The operator's control console on the other hand is usually well-designed with great emphasis often given to human-factor aspects such as ease and speed of operation. Very often the maintenance man is forced to use the operator's console since that is all there is, but it is usually quite inadequate for his purposes. The operator is particularly interested in being able to sense the state of the machine; quickly detect error conditions such as overflow, parity error, etc.; and take the actions necessary to control the computer such as

clear error, start, etc. On the medium and smaller machines, there is also the requirement to be able to examine the contents of various registers from the console for "console debugging". The cost of time on a large scale machine usually prohibits this kind of activity although, luckily for the maintenance men, the capability is often included.

It is possible to select several desirable features for the maintenance panel from a purely maintenance point of view. Some of these might be a single-clock-pulse push button, a repeat mode control, and the ability to directly sense and set the state of various control flip-flops. One extreme here is a panel such as the System Tester for the Autonetics RECOMP II which was connected to the central processor when necessary. The System Tester had 180 indicator lights and pushbuttons - one for every flip-flop and other controls such as timing pulse generator and marginal voltage controls. The main console on the AN/FSQ-32 (solid state SAGE computer) is almost this complete, but such exhaustive coverage is usually not provided. The signals to be displayed are usually selected by the hardware design group deciding what it would be "nice to have" in addition to the operator's console display. If a diagnostic programming effort is proceeding concurrently with the design work, then a much better choice can be made as to what to display.

The only two real limitations on the ability of the diagnostic program are:

(1) It will not operate at all unless a certain portion of the computer (usually quite large) is fault free.

(2) Its resolution is limited by its ability to sense signals.

A proper choice of displayed signals can greatly assist in overcoming both of these. An example falling into the first category would possibly be the timing chain. If it is not functioning properly usually nothing will run, but if it is displayed its operation can be quickly checked. In the second class would be signals such as flip-flops imbedded in a large diagnostic tree . An example of this might be a buffer register in the input/output multiplexer or select/controller. In fact the I/O has so many signals falling in both of these categories that separate maintenance panels are often associated with the I/O units.

Even with special effort it is usually not possible to design the machine so that the input/output circuitry associated with FILL control can be diagnosed. For that reason, this portion of hardware would be liberally displayed; but other I/O channels if present, can usually be automatically diagnosed. Similar comments can also be made in regard to the primary and other memory modules.

The important point is the role that the diagnostic programmer can play in the selection of controls and displays needed for the maintenance panel.

3. Added Hardware - Internal

It is often possible to greatly assist the execution of the diagnostic by adding a relatively small portion of hardware to the machine. An example of this is a comparator added to check the contents of one register against another [5]. Also falling into this category are simple test devices used to aid the fault-location process.

One internal feature that is often added is the capability to set marginal voltages during maintenance tests. This does not assist in the location of a fault that has already appeared during normal operation, but it does often cause a potential failure to appear during maintenance when it can be detected without loss of production time.

4. Added Hardware-External

Externally added hardware usually refers to items such as card testers for fault-location, but other equipment also falls into this category. One example is the previously mentioned System Tester for the RECOMP II. This device, which was connected directly to the central processor by several multi-pair cables, was primarily an extensive maintenance console which displayed the state of every flip-flop and provided controls for setting and clearing them and for controlling the clocking of the central processor.

Another good example of external hardware that greatly aids the diagnostic process is the FALT (Fadac Automatic Logic Tester) for the U.S. Army's small tactical computer, the FADAC. The FALT is connected to the FADAC by multi-pair cables and then the FALT "exercises" the logic of the FADAC to test it and identify any failures detected.

## IV. QUALITY FACTORS OF DIAGNOSTIC PROGRAMS

The quality of a diagnostic program cannot be assessed by examining only the program itself. A valid assessment requires that the entire maintenance procedure (both diagnosis, repair, and final check-out) be evaluated. It is quite possible to make some of the features below appear to be very good at the expense of others. The quality factors that have been identified are:

(1) Time required to execute the diagnostic program.

(2) The skill required to run the diagnostic, interpret the results, and take the necessary further action.

(3) The documentation required to specify how to run the diagnostic, how to interpret the results, and how to take the necessary further action.

(4) Proportion of the hardware that can be checked-out.

(5) Proportion of the hardware that can be completely diagnosed.

(6) Resolution of the defective element identification.

(7) Additional manual operations necessary to complete the identification process.

(8) Manual intervention during check-out routines.

(9) Nature of failures considered.

(10) Reliability of the program.

### A. CAPABILITIES

#### 1. Nature of Failures Considered:

The case where failures of several types have been "assumed away" has already been mentioned. The results of some Navy tests are interesting in regard to this quality factor. During independent testing of a diagnostic program it was found that it would not run at all for approximately 65% of the faults inserted. Consultation with the developer of the program determined that they were inserting mostly "illegal" faults and even further consultation was necessary to have the programmer point out that the test team was now "inserting them improperly".

#### 2. Proportion of Hardware Checked-out and/or Diagnosible

There does not appear to be any reason that should prevent a 100% check-out of all of the computer, i.e., if the check-out routine runs correctly or the diagnostic follows the go-path to its normal end, then the machine can be considered 100% operable. Any deviations from this specification should be questioned. The only problem area is the complete checking of peripheral equipment. Here, there may well be deviations from the 100% specification.

As pointed out earlier simple check-out is quite different from complete diagnosis which includes fault location as well as check-out. The primary problem is the proportion of the machine that must be failure-free in order to even load the diagnostic and execute the first simple test. In various machines the proportion necessary to load and start the diagnostic has been quoted to be as high as 85% of the total hardware. The implication is then made that "if the diagnostic program loads and executes the first instruction correctly 85% of the machine is failure-free leaving only 15% to be diagnosed automatically. If it does not load properly, then manual procedures must be followed to locate the fault in the remaining 85% of the machine." The analysis of the project test vehicle did show that a large percentage of the machine was required to be operative to obtain a "breathing" machine but this figure must be critically examined. Consider only an example of a simple 3-input diode logic gate. It may have as many as eight failure modes (not all of which will have unique effects), but even under one of these failure conditions the gate might still allow the basic "breathing" condition to exist. In fact, many of the failure modes may fall in this category.*

In attempting to raise the proportion of the machine that is diagnosible, it is necessary to simplify the loading logic as much as possible and to minimize the hardware used to implement it. When this is done, however, the results often are in direct opposition to the efforts to simplify the overall machine and minimize the total hardware required. This aspect of the problem was discussed earlier when the effects of logical design were considered.

### 3. Resolution

The goal of the complete diagnostic procedure is positive identification of the defective element that must be replaced. The "replaceable element" referred to here is the smallest hardware component that may be removed and replaced by the individual performing the diagnostic (probably the operator), so it will have to be a plug-in assembly. Normally, the logical organization will restrict the resolution of the fault-location procedure to a group of logical elements - gates, flip-flops, etc. If these logic elements are not all on the same replaceable element, then the best that the diagnostic program can do is to specify a group of plug-in cards as "suspect". Since it is assumed that the operator has no card testing equipment, his only alternative to replacing all suspect cards is to attempt

*Consider the second example in the Appendix: If the breathing condition requires G=STH, then there are only two failure modes that would prohibit further operations.

to locate the specific card that failed by a substitution and re-check procedure. The time required to do this will greatly affect the first quality factor discussed above as well as the one discussed below.

If the "average resolution" capability of the fault-location routines is given, the technique used to obtain this figure should be questioned. A simple approach to the problem would be to just take the raw average -

| Number of Suspect Defective Units | Number of Coded Stops Having This Many |
|---|---|
| 2 | 20 |
| 3 | 18 |
| 4 | 22 |
| 5 | 5 |
| 6 | 17 |
| 7 | 18 |

Average Resolution: 4.35 units

But, in an effort to make the figures look better, some have argued that the median (middle-most value) would be a better statistic:

Median Resolution: 4 units

A somewhat pessimistic figure is obtained by making a simple assumption as to the frequency of occurrence of each group of coded stops - a coded stop that applies to 4 suspect units will occur twice as often as one that applies to 2.

Simple weighted average resolution: 5.33 units

And yet another method would be to obtain a weighting for the probability of occurrence of each coded stop based on an analysis of the probability of failure of the components causing that stop. This would certainly be a mamouth task and it is not clear that the results would be of any value.

The same warning applies here as in any encounter with statistical data - find out the basic technique used to calculate the statistic quoted.

Some figures based on actual tests of a completed diagnostic are given below [8].

| | |
|---|---|
| Average resolution | 8 cards |
| Median resolution | 5 cards |
| Maximum resolution | 38 cards |

## B. EXECUTION REQUIREMENTS

### 1. Running Time

The time required to run the diagnostic program depends on whether or not a fault is detected. The two conditions then that must be examined are:

(1) Time required to follow the complete go-path.

(2) Time required to follow the go-path until the fault is detected and then the time required to locate the fault (worst case condition).

In some instances it is highly desirable to have a short check-out routine that verifies the condition of the machine; but in a tactical military situation, as in many real time environments, when a failure does occur the critical aspect of the situation is the maximum length of time required to detect it, locate the failure, repair it, and return the machine to use. There are many portions of the go-path, particularly at the beginning, where the order in which various tests are made is dictated by other considerations; however, for the most part, the order of tests can be varied and the criterion of maximum time required should be the governing factor.

2. Operator Skill

Since the primary role of the diagnostic program is to assist rapid restoration of service, the diagnostic program and all attendant procedures should be designed so that they may be executed by the operator of the equipment - not a higher echelon maintenance man.

3. Manual Intervention

Because of the specific assumptions made earlier as to the individual using the diagnostic program and the limitations on the equipment he has available, a two-step procedure requiring manual operation with external test equipment is to be avoided. If manual procedures such as substitution and re-check or special tests with internal test equipment are to be performed they must be clearly and explicitly documented and their effect on the time and skill required must be carefully evaluated.

A requirement for the operator to manually intervene during the running of the check-out routines is definitely to be avoided. At the beginning of the diagnostics when the "breathing" machine check is being made, a certain number of specific manual steps are usually required; and the operator can be used to "close the loop" on an input-output test of a device such as a typewriter. A greatly undesirable condition is a requirement for the operator to make checks during the tests to verify the correct operation condition. This may take the form of a display read-out [5] or a print-out. Not only is there a chance for human error, but the running time of the diagnostic will also be considerably increased.

4. Support Documentation

Again the personnel that are to use the diagnostic program and the environment in which it is to be used must be considered when the documentation is prepared. Both the logical organization and the physical form of all operator maintenance documentation are important.

## C. VERIFICATION OF THE DIAGNOSTIC PROGRAM

The verification that the diagnostic program meets its design specification may require almost as large an effort as the development of the program. A decision that must be made at the onset of the preparation of the acceptance or verification test plan is just how extensive the tests must be. The standard procedure of statistical quality control would permit evaluation at various confidence levels by using appropriate sampling plans. Unfortunately there are always those that come forward to say "This system must work in a tactical military environment and I must be absolutely certain that 100% of the faults possible will be detected and located". One would then be inclined toward making the expenditure to verify by testing 100% of the possible faults. Since it is a most unusual test system that in itself is error free, statistical quality control again says that all results will have associated with them a probability less than unity. This is not to say that 100% testing would not be desirable, but only points out that caution must be used in interpreting the results. In any regard, with such a large population of possible faults extensive testing will usually be required.

Testing is facilitated by some device used to physically introduce the fault while the diagnostic is run to locate it. These devices usually take the form of extenders for the printed circuit cards with control switches built into the extender. Using such a device with normally controlled switches, it took 30 days at 3 shifts per day to check the diagnostics for the UNIVAC 1218 by sampling only [5]. Burroughs on the D-84 project used a relay operated extender card where the relays were controlled by a D-825 computer [5]. This permitted automatic verification of the diagnostic programs. No time figures are known.

## D. SPECIFICATIONS FOR PROGRAM

The quality factors discussed in the preceeding sections on Capabilities and Execution Requirements should have a direct influence on the specifications written for diagnostic programs. In addition some predetermined procedure to verify the reliability of the program should be included in the specifications, as discussed in last section.

Certainly no single diagnostic program can meet the ideal standards for each quality factor. Conflicts will no doubt arise between many factors and it is thus essential that the specifications assign relative priorities to the factors considered in order that such conflicts may be satisfactorily resolved. The resulting specifications will then provide for a realistic diagnostic program with an optimum balance of capabilities and execution requirements.

## V. SUMMARY

Diagnostic Programming is at present still an unstructured field. Many different approaches are being pursued with considerable degrees of success.

In this paper we have attempted to provide a common framework in which the various approaches could be discussed and the many related problems pointed out. The results of several organizations have been cited to help assess the state-of-the-art as it is today.

The most important conclusion that can be drawn from the material presented here is that diagnostic programming should be considered throughout the design stages for any computer if the real potential of such a maintenance aid is to be realized. A systems approach of this type should result in a design minimizing the "hard-core" problem and providing an optimum mix of hardware aids and module organization that will immeasurably simplify the task confronting the diagnostic programmer without unduly complicating the logistics problem.

Particularly from the military point of view we would hope that the quality factors presented and the importance of the systems approach to design will both be carefully considered when specifications for future computers are written.

# APPENDIX

## SIMPLE FAULT ANALYSIS

In order to be completely accurate, an experimental fault analysis for operational units such as AND gates, OR gates, etc. should be performed in the environment in which it is to be used. An analysis of the circuit alone, either theoretical or experimental, fails to take into consideration the loading and isolation provided by other units connected to the inputs, and outputs. An analysis of an isolated circuit also makes it difficult to detect feed-back effects of component failures.

As a first approximation to an experimental analysis of actual hardware, a theoretical analysis of an operating circuit will be performed. The basic logic gates and their MIL-STD symbols are shown in figure 11. First a fault analysis of the AND gate will be made. A typical situation is to have "levels" of AND gates and OR gates as shown in figure 12 which includes both the logic and schematic diagrams. The gate under study is AND gate "X". Only catastrophic failures are considered. The ones possible are listed below and their predicted effects are described.

$D_1$ : Open: I-Fault* on X

The AND gate functions just as if A were not connected to it, and X may be active even if A is not.

Shorted: I-Fault on X

X will be active if and only if A is positive regardless of states of B and C

I-Faults on B and C

Feedback effects will force B and C to the active state when A is positive regardless of their own inputs. B and C can also go positive under the control of their own inputs.

---

* I-Faults and A-Faults are defined in Section II E, p 20

$D_2$: Open: I-Fault on X

Output X is no longer inhibited by B

Shorted: I-Fault on X

X will be active if and only if B is positive regardless of states of A and C

I-Faults on A and C

Feedback effects will force A and C to the active state when B is positive regardless of their own inputs. A and C can also go positive under the control of their own inputs even if B is inactive. The I-Fault on A will cause an I-Fault on E.

$D_3$: Effects similar to those obtained for $D_2$

$R_1$: Open: A-Fault on X

Output X will also be inhibited by H. ($X = A \cdot B \cdot C \cdot H$)

Shorted: I-Faults on A, B, C, and X

Outputs A, B, C, and X will be held in the positive-active state.

In actual experiment, the results depend on actual component parameters and voltage levels and the source of the initial inputs, as well as threshold levels assigned to each gate. Multi-level diode logic without inverters or transistor amplifiers to restore nominal voltage levels are very difficult to analyze accurately on paper due to the complicated loading effects. A further complication in actual circuits arises from the fact that one shorted diode can result in one or more diodes being burned out (open) depending on the order in which signals are applied. Of course the same logic diagram implemented with different logic gates (for example DTL) will yield quite different results when similar component faults are introduced.

The effects of component failures can also be depicted by a modified Logic Diagram Two of these are given in Figures 13 a and b.

Another method of depicting the effects of failures is to modify the equations describing the logic. Consider the analysis of failures in OR gate "X" in figure 14.

The correct set of logic equations are given below:

(1) $A = O \cdot P$

(2) $B = Q \cdot R$

(3) $C = S \cdot T$

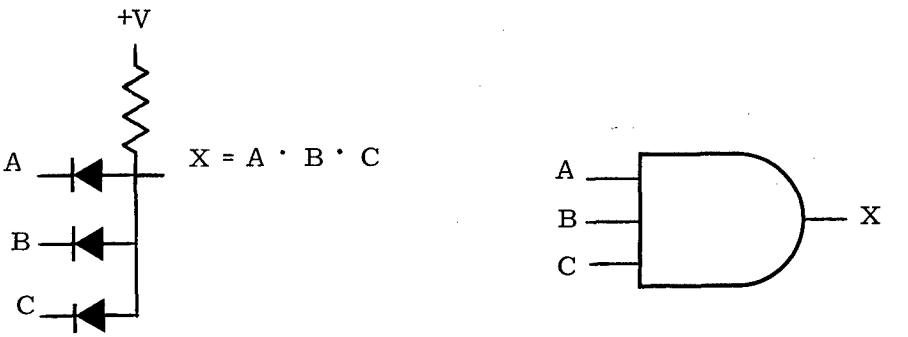
(4) $D = M \cdot N$

(5) $E = D + A = MN + OP$

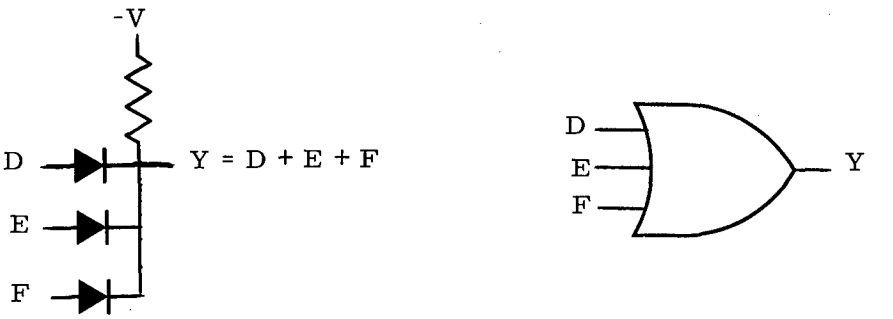
(6) $F = E \cdot X = MNQR + MNST + OP$

(7) $G = X \cdot H = OPH + QRH + STH$

(8) $X = A + B + C = OP + QR + ST$




(a) AND Gate




(b) OR Gate

Figure 11: Diode Switching Circuits
(Positive Logic)

The possible component failures and their effects are given below, ignoring loading effects. Only the equations that are changed are given.

$D_1$ : Open: A-Fault on X

(6) F = MNQR + MNST + OPQR + OPST

(7) G = QRH + STH

(8) X = B + C = QR + ST

Shorted: A-Fault on X, B, and C

(2) B = Q · R · A

(3) C = S · T · A

(6) F = OP

(7) G = OPH

(8) X = A = OP

$D_2$: Open: A-Fault on X

(6) F = MNST + OP

(7) G = OPH + STH

(8) X = A + C = OP + ST

Shorted: A-Fault on X, A, C, and E

(1) A = O · P · B

(3) C = S · T · B

(5) E = D + A = MN + OPB

(8) X = B = QR

$D_3$: Effects similar to $D_2$

$R_1$: Open: I-Fault on X

(8) X = A + B + C + H

Shorted: A-Faults on A, B, C, and X

(1) A = $\emptyset$ (Inactive State)

(2) B = $\emptyset$

(3) C = $\emptyset$

(6) F = $\emptyset$

(7) G = $\emptyset$

(8) X = $\emptyset$

Again, the experimental results will certainly not be completely unambiguous because of the loading effects and the problem of assigning threshold levels to determine whether a given gate is active or inactive under the new fault conditions.

This appendix is not intended as a guide to fault analysis, but merely as an indication of the complexity of this problem area and a few different approaches that may be helpful. It would appear that the best approach is through experimentation on identical circuits to the ones in the computer to be diagnosed, similarly imbedded in other logic.

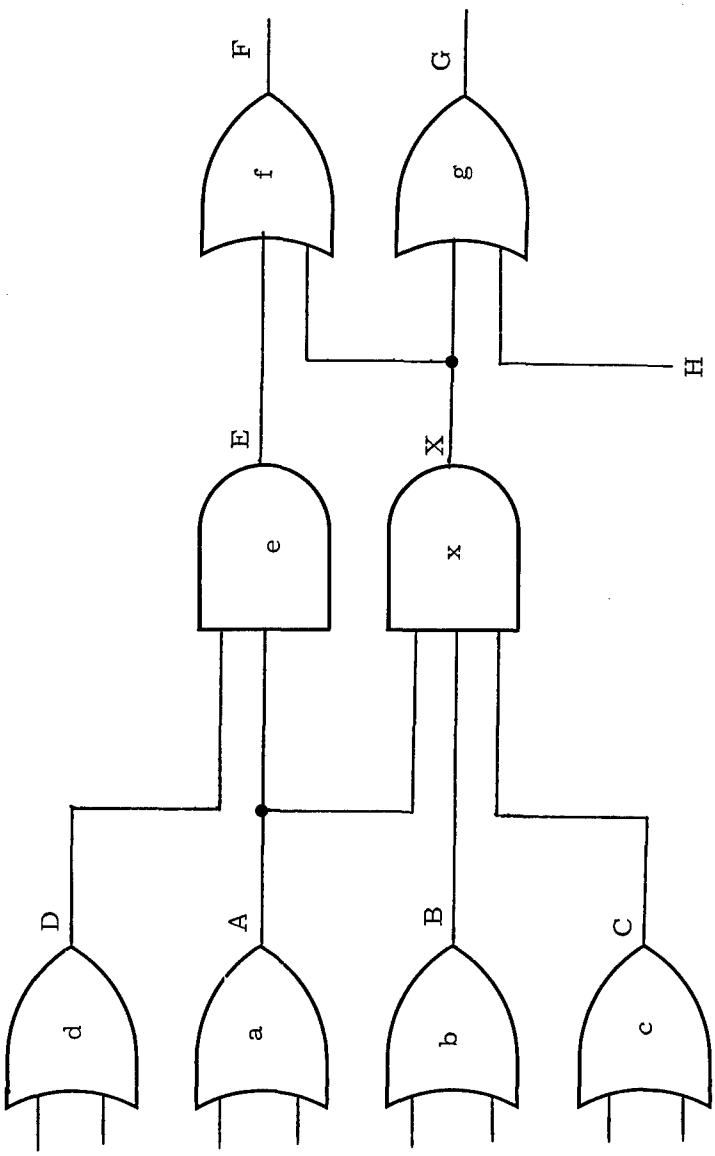



Figure 12(a): Fault Free Logic Diagram I

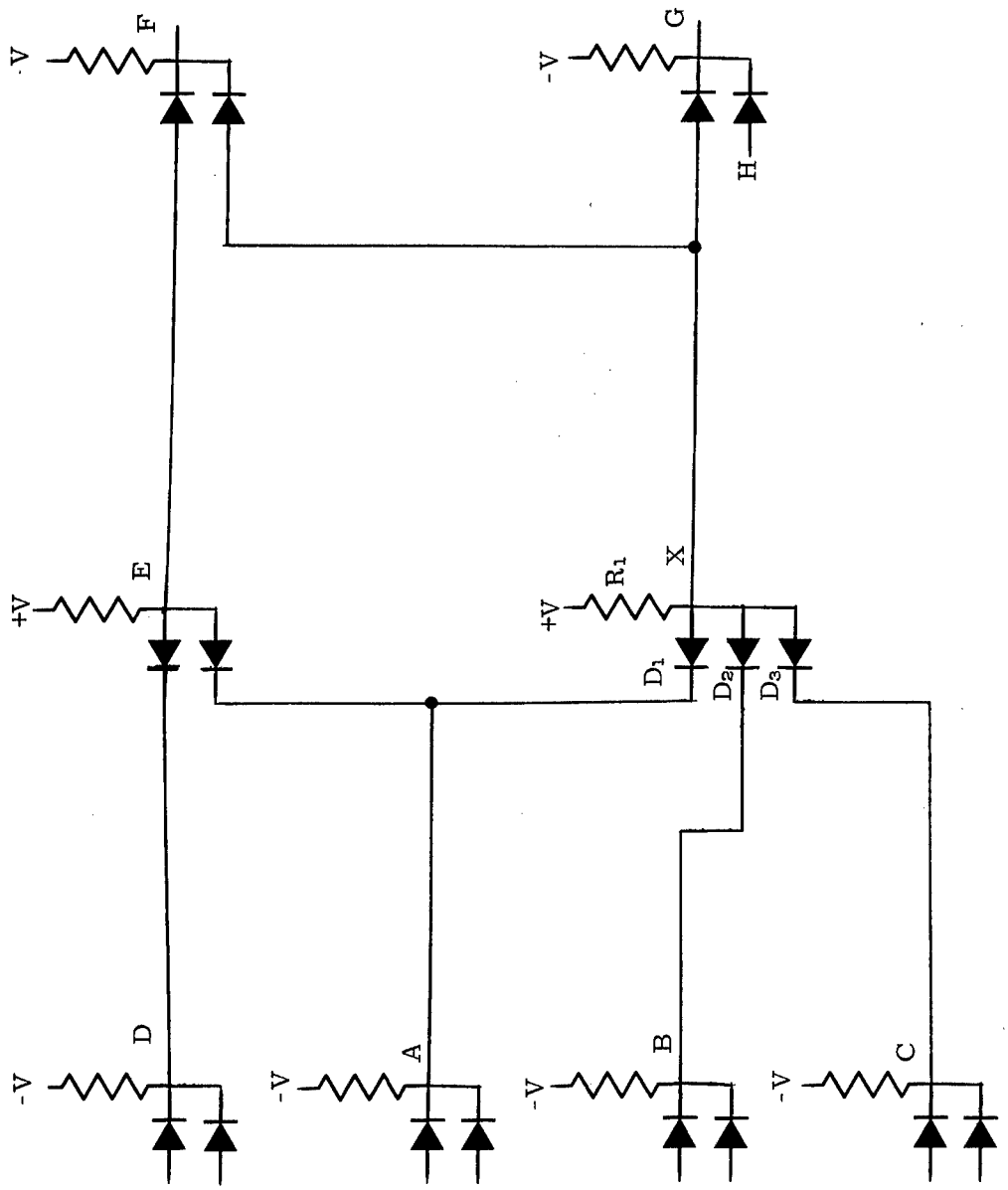



Figure 12(b): Schematic Diagram I

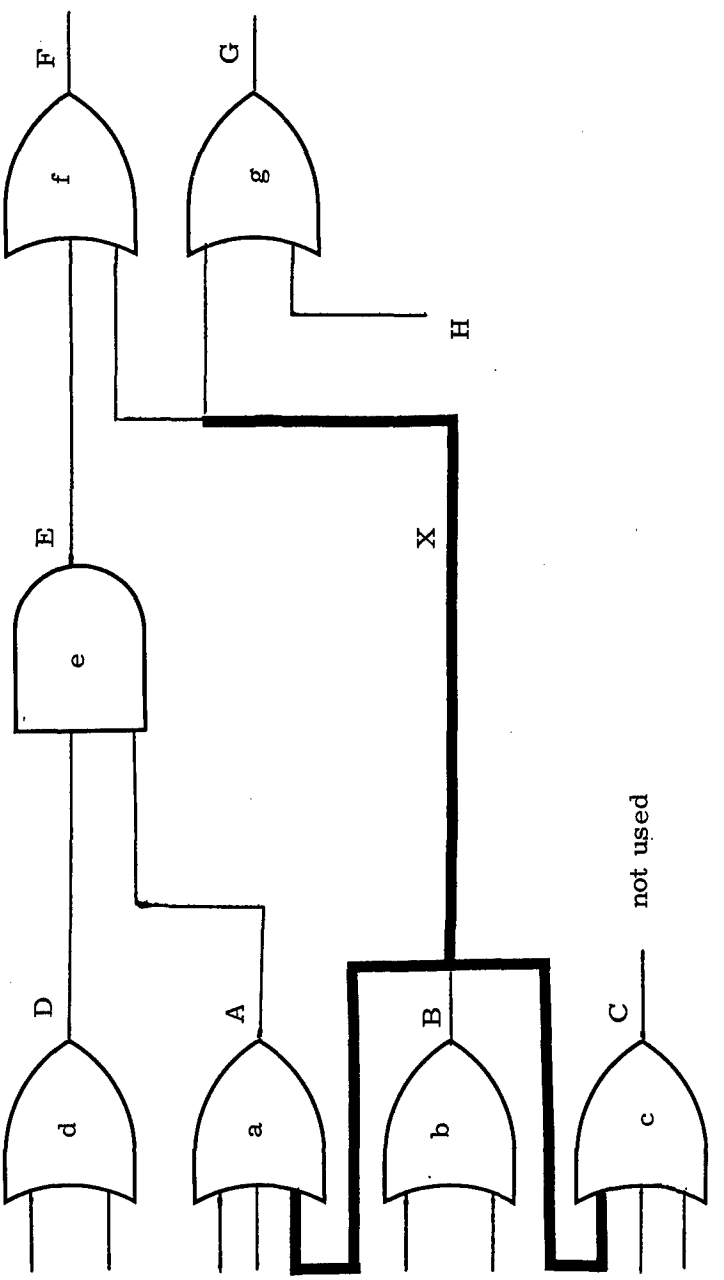



Figure 13(a): Modified Logic Diagram I, $D_2$ Shorted

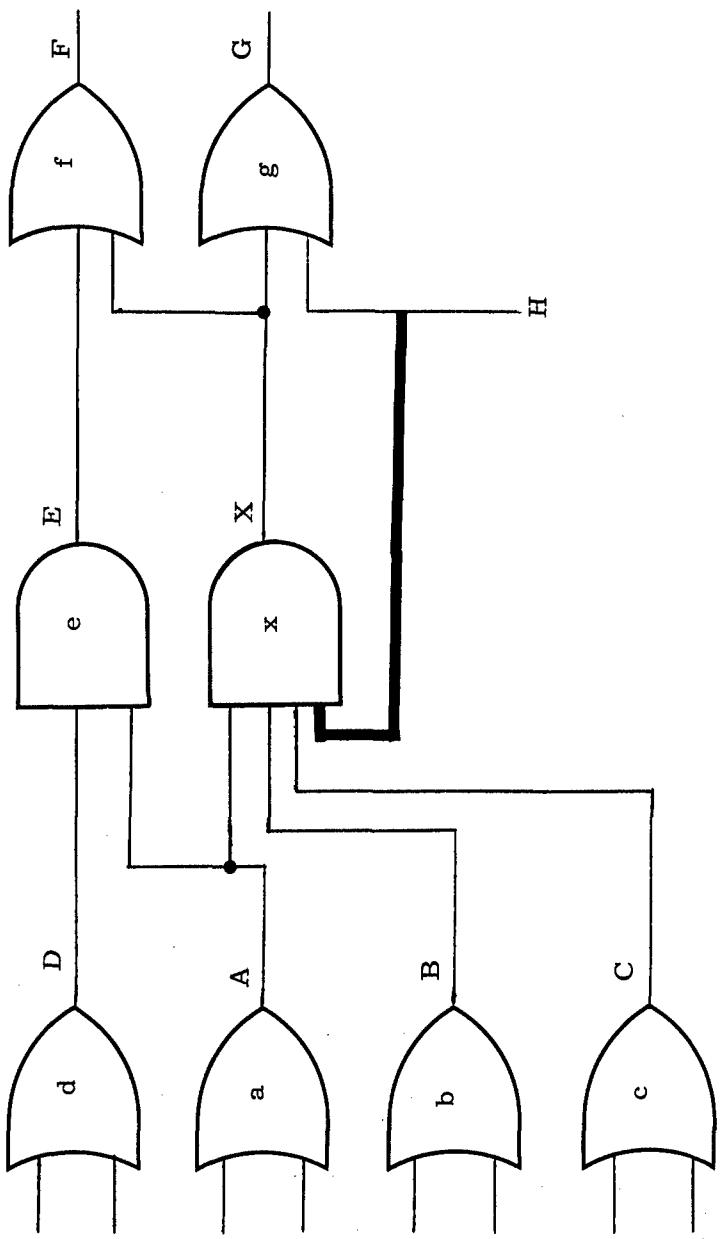



Figure 13(b): Modified Logic Diagram I, $R_1$ Open

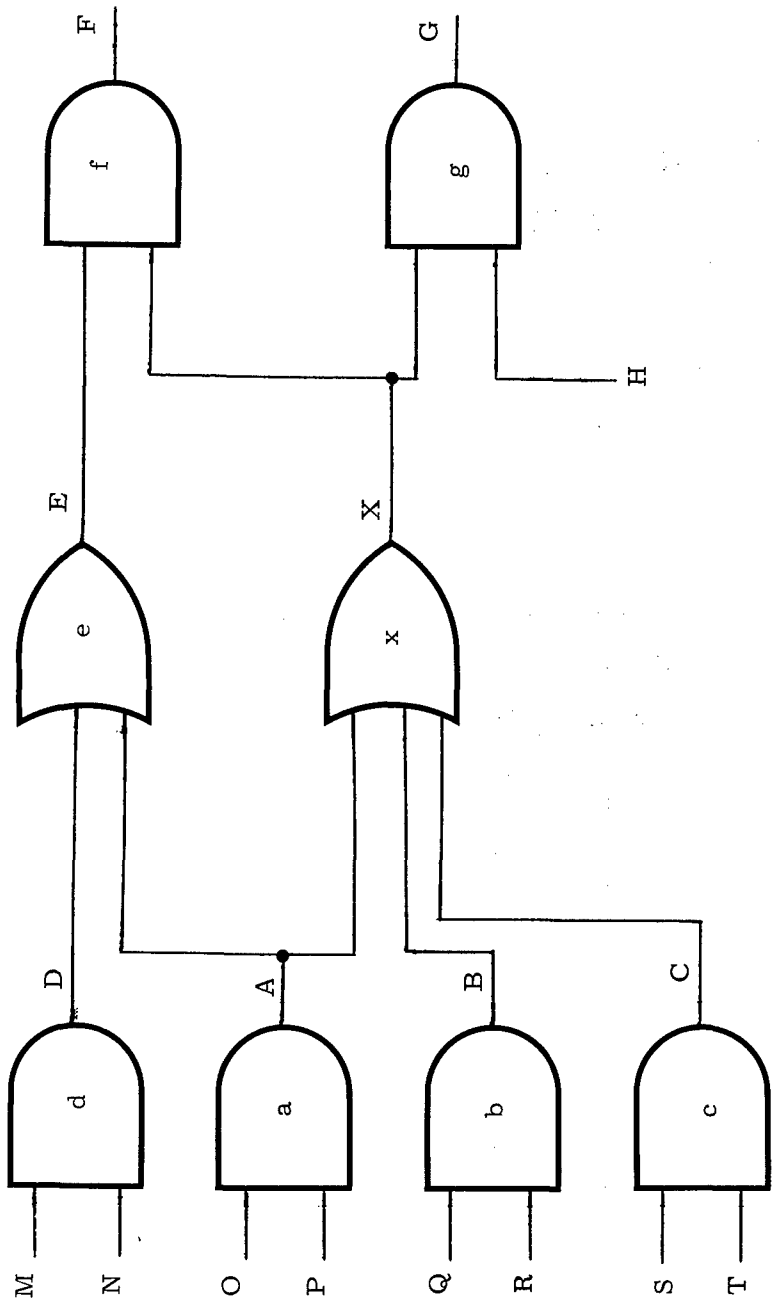



Figure 14(a): Fault Free Logic Diagram II

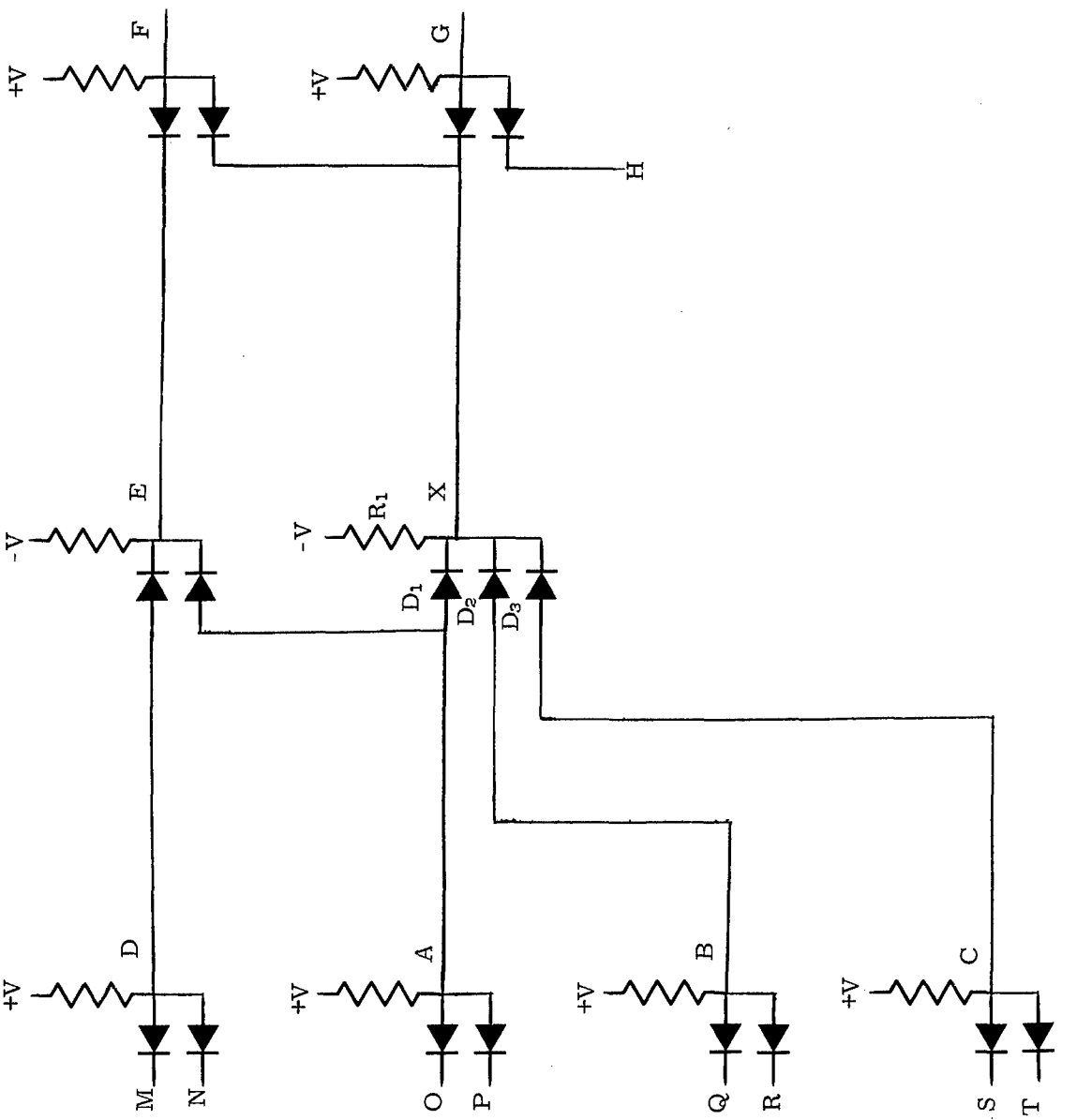



Figure 14(b): Schematic Diagram II

## REFERENCES


1. Salisbury, A.B. and Enslow, P.H. Jr., "Diagnostic Programming for Digital Computers: A Bibliography", Technical Report ELEC-67-1, U.S. Military Academy, West Point, N.Y., April 1967.

2. Quatse, J.T., "Time-Shared Troubleshooter Repairs Computer On-Line", Electronics, Jan 24, 1966, pp 97-101.

3. ARIES Corp., "Computer Diagnostic Study", NASA, Goddard Space Flight Center, Greenbelt, Md., Report on Contract No. NASS-3147 Jan-Jul 1963.

4. Carter, W.C.; Montgomery, H.C.; Preiss, R.J. and Reinheimer, H.J., "Design of Serviceability Features for the IBM System 360", IBM Journal, April 1965, pp 115-126.

5. Buckley, F.J., Assorted documents and Memos-for-Record of the Automatic Data Field Systems Command, Ft. Belvoir, Virginia.

6. Enslow, P.H. Jr., "A Bibliography of Search Theory and Reconnaissance Theory Literature", Technical Report No. 1906-2, Stanford Electronics Laboratories, Stanford University, Stanford, Calif., June 1965.

7. Cohen, J.J., and Whitaker, L.A., "Improved Techniques in Diagnostic Programming", The Sylvania Technologist, Vol XIII, No 3, July 1960, pp 90-96.

8. Radio Corporation of America, "Final Progress Report: Test and Diagnostic Program for Random Access FIELDATA Computer System", RCA Defense Electronic Products, Communications Systems Division, Camden, N.J., September 1, 1965.

9. Bashkow, T.R.; Friets, J.; and Karson, A., "A Programming System for Detection and Diagnosis of Machine Malfunctions", IEEE Transactions on Electronic Computers, Vol 12, No 1, Feb 1963, pp 10-17.

10. Enslow, P.H. Jr., "Documentation Techniques for Digital Computers", Technical Report Elec-67-1, United States Military Academy, West Point, N.Y., January 1967.

11. Tsiang, S.H., and Ulrich, W., "Automatic Trouble Diagnosis of Complex Logic Circuits", The Bell System Technical Journal, Vol XLI, No 4, July 1962, pp 1177-1200.

12. Kruskal, J.B. and Hart, R.E., "A Geometric Interpretation of Diagnostic Data from a Digital Machine: Based on a Study of the Morris, Illinois Electronic Central Office", The Bell System Technical Journal, Vol XLV, No 8, October 1966, pp 1299-1338.

13. Eldred, R.D., "Test Routines Based on Symbolic Logical Statements", Journal of the ACM, Vol 6, No 1, January 1959, pp 33-36.

14. Radio Corporation of America, "Proposal and Technical Analysis for Diagnostic Programming of MICRORAC Computer", Defense Electronics Products, Communications Systems Division, Camden, N.J., August 1963.

15. Maling K.,"Classes of Component Failures and a System Design to Facilitate their Location", TR 00.1029 Rev, IBM Data Systems Division, Development Laboratory, Poughkeepsie, N.Y., June 28, 1963, Rev Mar 10, 1964.

16. Seshu, S., "On an Improved Diagnosis Program", Report R-207, Coordinated Science Laboratory, University of Illinois, Urbana, Ill, May 1964, AD 601156.

17. Buckley, F.J., "ADP Systems Maintenance", Tech Doc Rpt 04-01-01, Automatic Data Field Systems Command, Ft. Belvoir, Va., December 1965.

18. Lewis, T.B., "Techniques for Achieving Operational Reliability and Maintainability in Digital Computers", FSD Space Guidance Center, Owego, N.Y., date unknown.

19. Computronics Inc. (now Decision Systems, Inc.), "MICRORAC Maintenance and Diagnostic Study", Computronics, Inc., Ft. Lee, N.J., February 1963.

## VITAE

Major Philip H. Enslow, Jr. graduated from the United States Military Academy with a degree in military science in 1955. He was awarded the degrees of Master of Science in Electrical Engineering in 1958, Engineer in Electrical Engineering: Administration in 1959, and Doctor of Philosophy in Electrical Engineering in 1965 by Stanford University. Major Enslow has served in Signal battalions in the United States and Korea. He worked on the development of the Army's tactical automatic data processing systems before coming to the Military Academy in 1962 where he directed the digital computer course and laboratory for two years. He is a senior member of the IEEE and a member of the ACM, AFCEA, and Sigma Xi.

Major Alan B. Salisbury graduated from the United States Military Academy with a degree in Military Science in 1958, and received a Master of Science in Electrical Engineering: Administration degree from Stanford University in 1964. Major Salisbury has served in communications assignments in Maryland and Alaska. He has developed and taught courses in digital computers and solid state electronics at the U.S. Military Academy and is a member of the ACM and AFCEA.

Unclassified

Security Classification

**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)*<br>Department of Electricity<br>United States Military Academy<br>West Point, New York 10996 | 2a. REPORT SECURITY CLASSIFICATION<br>Unclassified<br>2b. GROUP<br>N/A |

3. REPORT TITLE

A Pragmatic First Look at Diagnostic Programming for Digital Computers

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Technical Report

5. AUTHOR(S) *(First name, middle initial, last name)*

Philip H. Enslow, Maj , SigC, U.S. Army
Alan B. Salisbury, Maj, SigC, U.S. Army

| 6. REPORT DATE | 7a. TOTAL NO. OF PAGES | 7b. NO. OF REFS |
|---|---|---|
| May 1967 | 56 | 19 |

| | |
|---|---|
| 8a. CONTRACT OR GRANT NO. | 9a. ORIGINATOR'S REPORT NUMBER(S) |
| b. PROJECT NO. | Elec-67-3 |
| c. | 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* |
| d. | None |

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited

| 11. SUPPLEMENTARY NOTES | 12. SPONSORING MILITARY ACTIVITY |
|---|---|
| | U.S. Army Research Office<br>Durham, N.C. |

13. ABSTRACT

A brief survey of diagnostic programming for digital computers. The purposes and organization of the program are covered as well as many of the problems attendant to their preparation. Various techniques that have been used to overcome these are discussed with emphasis on those that are practical (at least from the point of view of effort required) and those that have seen wide use. The primary purpose of this paper is to fully define the nature of the problem and the special considerations involved. Most of the discussion is fully applicable to both commercial and military equipment.

DD FORM 1 NOV 65 1473

Unclassified

Security Classification

| 14. KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| Diagnostic Programming<br>Maintenance<br>Fault Location<br>Digital Computer<br>Check-Out Routine | | | | | | |