SQT

FINAL REPORT

OF RESEARCH CONTRACT NSG-3053

"DEVELOPMENT OF A FAULT-TOLERANT MICROPROCESSOR BASED COMPUTER SYSTEM FOR SPACE FLIGHT"

SUBMITTED TO

THE NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

SUBMITTED BY: DR. V. T. MONTGOMERY

Electrical Engineering Department
Southern University
Baton Rouge, LA 70813
Phone: (504) 771-5357

FINAL REPORT

OF RESEARCH CONTRACT NSG-8053

"DEVELOPMENT OF A FAULT-TOLERANT MICROPROCESSOR BASED COMPUTER SYSTEM FOR SPACE FLIGHT"

SUBMITTED TO

THE NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

SUBMITTED BY: DR. V. T. MONTGOMERY

Electrical Engineering Department
Southern University
Baton Rouge, LA 70813
Phone: (504) 771-5357

ABSTRACT

This report is concern with developing a methodology to be followed when attempting to design a tightly coupled, highly reliable microprocessor based computer system. The concept of Triple Modular Redundancy (TMR) with Sparing is used. The notion of syncronizing by using a single crytal oscillator is is examined. The use of decoders to replace voters is also used. The decoders not only isolate the failed module by also allows error identification to be accomplished. Each module is to have its own RAM memory. The necessary circuitry to select a correct memory and the corresponding DMA controller has been designed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Introduction

The work in this project involves using commercially available microprocessors in an environment which requires a highly reliable microcomputer system. The project specifically addresses synchronization of microcomputer systems and the organization and development of a redundant system with sparing. The Intel 8085 system design kit was used to form the basis of the discussions within this report. However, the concepts projected in this report can be applied to any of the commerically available microprocessor based systems.

It is assummed that commerically available microcomputer systems will meet the specifications as stated by their manufacturer. These conditions are not as strigent as space-flight requirements. Consequently, the failure rate will be significantly higher. We do not attempt to calculate that failure rate. Rather, emphasis has been placed on timely, reliable response to a failed module.

Much attention was given to checking in real time with the use of hardware to perform as many checking functions as possible. By using hardware, the system can run at the designed speed until an error occurs. If or when an error does occur, hardware recovery assures the designers of the fastest and most accurate recovery.

What has been attempted here is the development of a very tightly coupled Triple Modular Rundundancy (TMR) with Sparing. A system designed using this philosophy will run exceptionally

fast. Errors will be detected in a minimum amount of time and the recovery time will also be minimized. The result is a system which is not slowed down by the checking features and in most instances microcomputer modules can be swapped out in a real time environment with a minimum effect on real time operations.

Approach

A triple modular redundancy (TMR) with sparing system normally has three basic computer modules which are being compared constantly to detect errors. The errors are detected by voting on the addresses, data and status information. If an error does occur, the module thats in error is replaced by one of the spares.

Our approach involves using three microcomputer modules in the basic TMR configuration and three spares (Figure 1). Addresses and data are checked constantly. Every bus transfer is observed and checked. The data selectors of Figure 1 are responsible for the selection of microcomputer modules in the TMR configuration. The decoders detect error and identify which module is in error. If the system is not in an critical I/O operation when an error occurs, the control network will first store the state of the computer network. The failed computer module is taken out of the network by changing the code on the data selector. Next, the error detector is deactivated to allow the RAM memory to be updated. One of the correct microcomputer modules is selected. It's content is assummed to be correct and is broadcasted to all RAM memory. We note that the state of the system is in RAM and is transferred along with other data. The module counter now points to spare to be activated. This number is transferred to the data selector that handles the module that was in error. Finally, a vector interrupt is forced by the controller causing all microcomputer modules to reload the state and begin to run. The controller will go back to look for another failure.

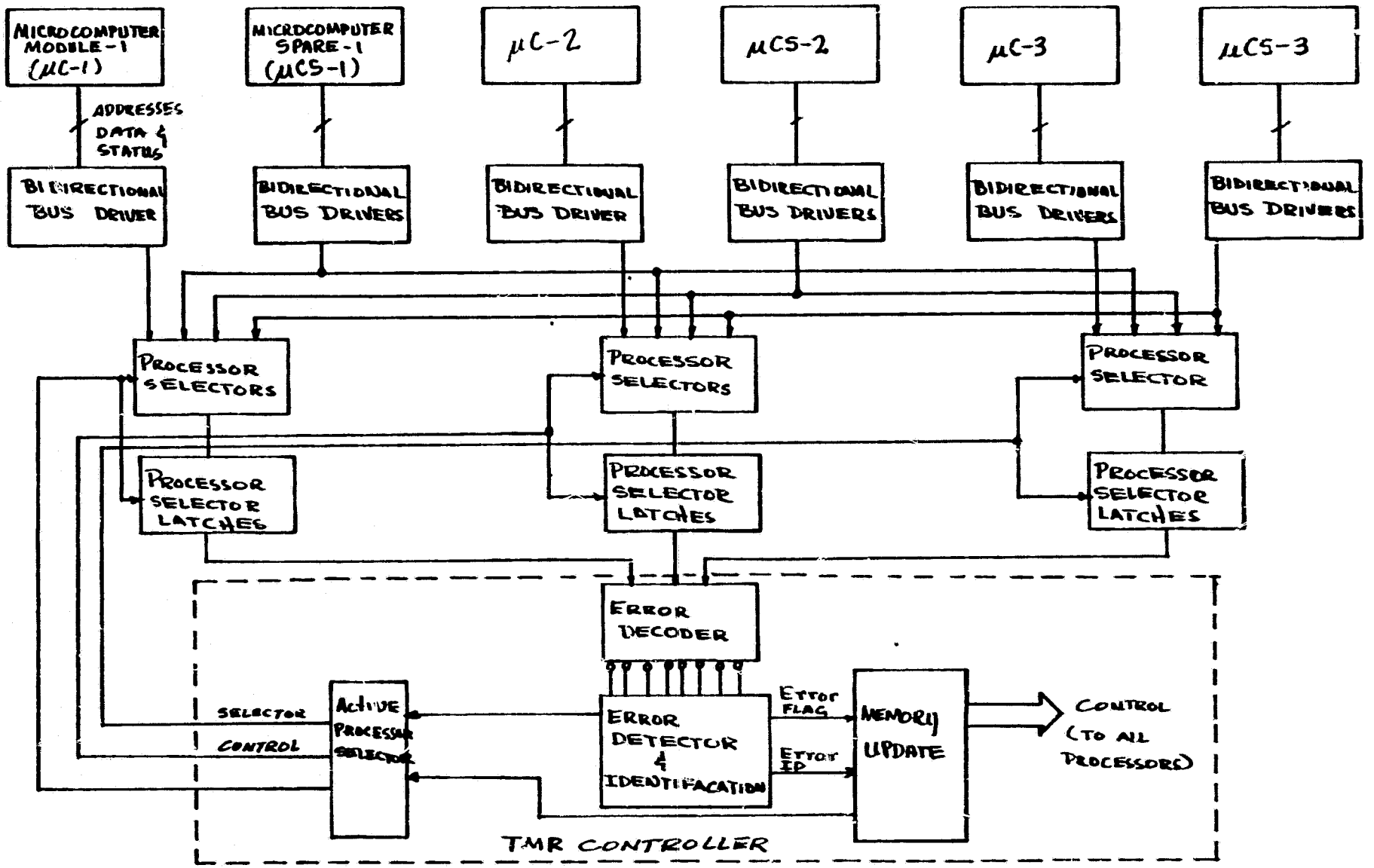

Figure 1. Design Overview

System Partitioning and Configuration

The configuration of the system will determine where and how the TMR and sparing can be invoked into an overall reliable system. The main factors which determine the overall configuration are the number of I/O devices, the amount of code that is written for the system, the amount of connected memory, and the amount of data space required for the system to function properly.

The 8085 is an example of a very large scale integrated (VLSI) chip. A minimum configuration (Figure 2) for basic control functions consists of the processors, an I/O and a read-only memory (ROM) chip, and a random access ;memory (RAM) and 1/O chip. The 8085 processes the information which it obtains from either the ROM or the RAM memory and the I/O chips. The ROM memory contain code which is permanently needed to run the system. The system we refer to has a keyboard and display chip which is used to input control functions and data, and output register and memory values. The RAM & 1/O chip has two functions. The first is to hold the user's program and temporary data. The second purpose is to interface with external devices.

The total memory for a microcomputer system consist of both ROM and RAM memory. The program and data size will determine how the TMR microcomputer will be designed. There are several choices. The first choice is to have a separate memory for each microprocessor based system. If we configure the system in this manner, it will be necessary to change memories when a processor malfunctions. But the new memory will have to be updated so far as

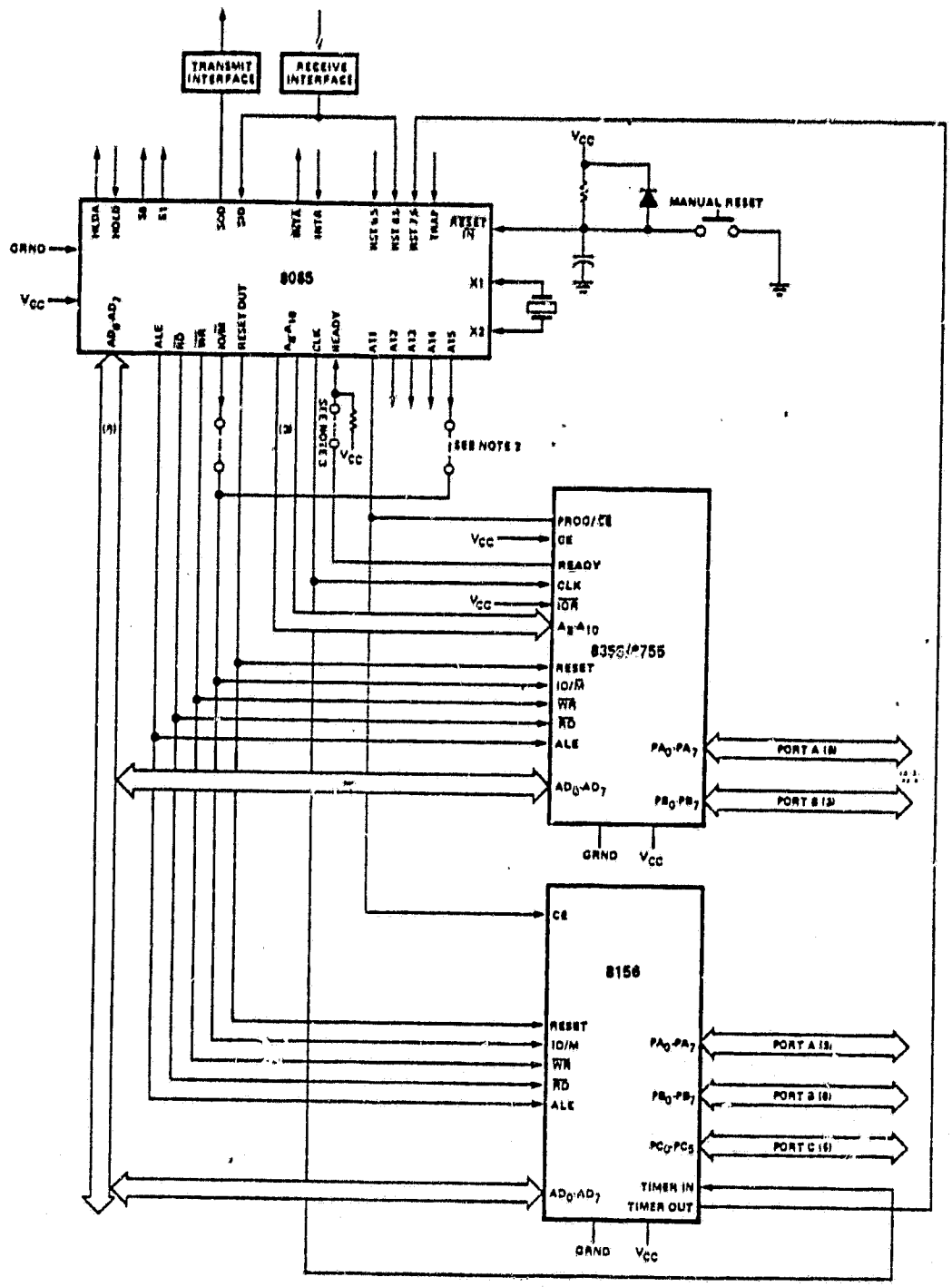

Figure 2. Minimum System Configuration (Intel)

data is concerned. It is obvious that we want to keep the ratio of ROM to RAM as high as possible so that recovery time is as short as possible.

A variation on this scheme is to allow data to be written to all RAM memories whether that RAM is currently a part of the basic TMR configuration or a part of one of the spares. By writing to all memories we can decrease the recovery time because updating would be eliminated. The disadvantage of this method is all RAM has been running for the same time and thus is just as likely to be faulted as the system that it is replacing.

Another choice of configurations is to have three RAMs triplicated such that we vote on the correct output when data is read from these memories. If an error is encountered, it will be masked by the values of the two correct memories.

Checking I/O operations presents a special problem because the I/O chips are connected to the bus on one side and to the I/O devices on the other. Once data is transmitted to the I/O chip, we assume that the I/O chip is performing properly. Data transferred into the processor can be checked by the error detecting network because this data is on the main general purpose bus. We can triplicate the output of the I/O chips. This would ensure that the I/O devices are receiving the correct data.

At this point, we see that we can be certain that our system is performing properly simply by triplicating memories and memory transfers, and by triplicating the output of the I/O chips. We must do the same for the bus which interfaces all these devices.

However, the amount of added logic far exceeds that of the basic system configuration. This implies that if a fault should occur, it is likely to be in the checking portion as opposed to the basic computer system. In an attempt to obtain an optimum configuration, it was decided to check transfers on the main data and address busses only. Each microcomputer module will have its own memory. The memory will be divided into ROM and RAM. The RAM will be as small as possible, so that recovery will be minimal.

The configuration decided upon is shown in Figure 1. Three of the microcomputer modules form the basic TMR configuration. The remaining three are spares. The solid line interconnecting the processors to the data selectors represent the address and data buses. The number of actual lines depend on processor size and the bus configuration. Typically, there are sixteen address lines and eight data lines. In the case of the 8085, there are sixteen total lines because the lower address and data lines are multiplexed on the same pins. If the system is designed such that the address and data lines are demultiplexed between the processor and the remainder of the system, then we have a normal address and data bus. Otherwise, the checking network must be responsible for checking when addresses are valid and when data is valid.

As can be seen from Figure 1, the selection of which microcomputer module is currently in the TMR configuration is done by the data selectors. The number of control lines determine how many spares the system can have. A two-to-four line selector can have three spares. A three-to-eight line selector can have seven spares. Therefore, the number of spares is two to the n

minus one, where n is the number of control lines for the selector.

The output of the selectors is fed into the voters and error detector network. This section is responsible for the identification of the failed microcomputer module and the generation of the data signals. The error recovery network is responsible for the sequencing of control information if an error does occur. This network will interrupt the microcomputer modules and cause them to store current state information.

The Next Available Spare (NAS) counter will be incremented and this information will be directed to the data selectors which is in error. This causes another processor to be selected. The TMR Controller will then cause the memories of the active processor modules to re-write themselves. During this period, the error detector is ignored but the voter is relied upon to bring the new memory up to date.

Finally, the TMR Controller will cause the microcomputer modules in the active processor configuration to restore the state. At this point, the processing can continue and the Error Detector can look for another fault.

Synchronization

One of the first problems to be solved is that of synchronization. It may or may not be a problem. It depends on how the checking is to be done on the microcomputer modules. It will also depend on what is to be checked. Let's look at some possible checking schemes. One approach is to check data after a given number of instructions have been executed. There are several advantages to this scheme; 1) timing is not as critical because we can simply count op-code fetch cycles until we are ready to test data, 2) microcomputer modules can be slightly out of sync during an interval, and 3) the checking will be minimized because we only use a few cycles to check. The disadvantages are; 1) an error may occur during the timing period that will go undetected until a checking cycle is entered, 2) incorrect output data or status can alter handshaking and data flow between I/O devices and the microcomputer module, and 3) to make this scheme efficient, there must be restrictions on the programmer and the software that is written for the system.

Another scheme is to only check when data is being transferred. The philosophy here is "until incorrct data or status is transmitted beyond the system, no fault has occurred." The disadvantage of this philosophy is that the memory may be completely degraded by the time the fault is discovered. One solution to this problem is to this problem is to triplicate the memory so that incorrect data can not degrade the system. Of course this means that more logic has to to be added to the system.

It was decided that minimizing the logic is the best approach.

A parts count was the determining factor that led to this decision. If the number of components is increased beyond that of the system, then the checking network is more likey to be in error than is the original microcomputer module. Consequently, it was decided that the best approach was to check address and data transfers only. More precisly, all addresses and data transfers to and from the processors, memories and I/O devices are checked.

Checking all data transfers implies a tight coupling between all microcomputer modules as well as the checking network. It also means that all microcomputer modules must be executing the same code in perfect sync with one another. Hence, it was necessary to devise a scheme where only one clock is used for the entire TMR system including the sparring modules as well as the voting system.

By observing the timing mechanism of Figure 2, we can see another problem. The clocking circuit for microprocessors has been placed inside the chip with the logic so that the chip count can be reduced. Synchronized timing can be obtained by the intelligent use of external timing control lines such as READY, RESET-IN, or HOLD.

An attempt to use the READY input to synchronize the microprocessors was attempted. The circuit of Figure 3 was used to hold the READY line of all processors until each was in the op-code fetch state. When a processor enters the op-code fetch state the status lines IO/M, S1 and S2 are in state (011). This causes the first one-shot to fire changing the state of the op-code fetch flip-flop. When all other processors have reached the same state, the second one-shot will fire. If the single-step switch is inactive, the RDY signal will

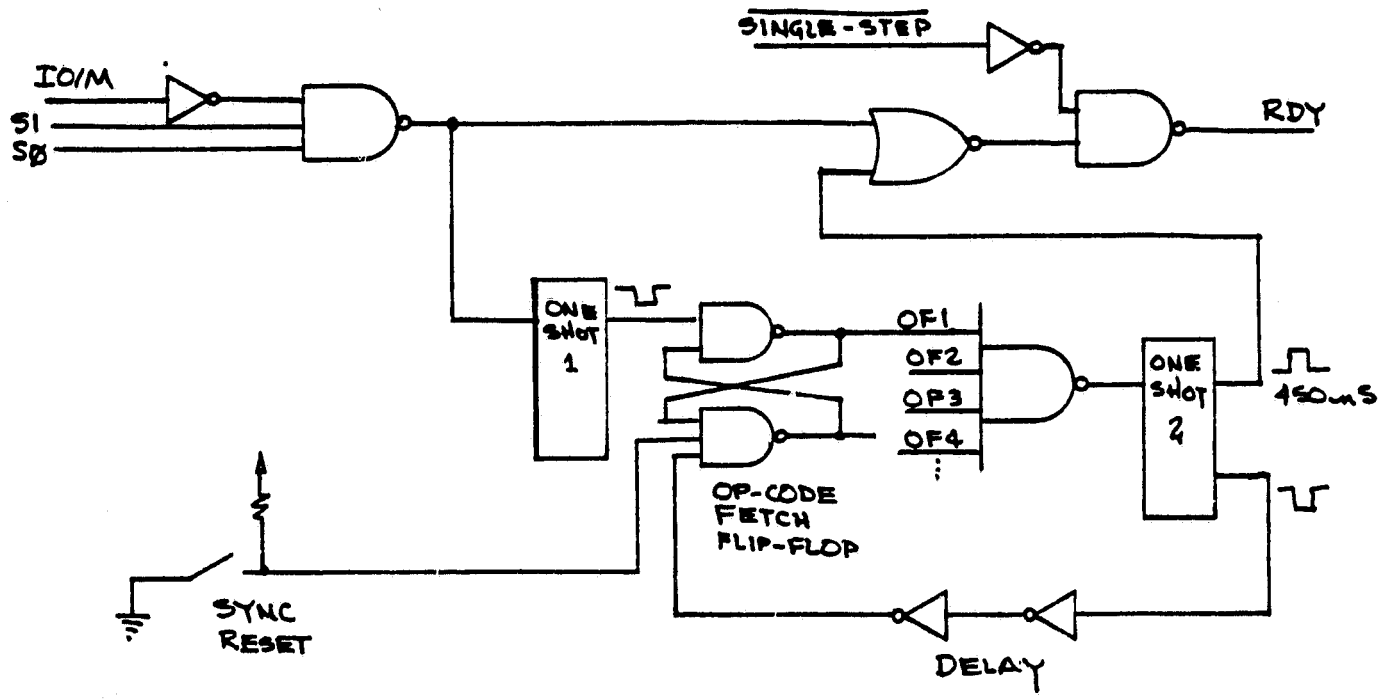

Figure 3. A Synchronization Technique.

be sent to all processors. The op-code fetch flip-flop is reset after a short delay and waits for the next instruction to go to the op-code fetch state.

In observing an oscilloscope, all processors appeared to be in perfect sync. However, we were not satisfied with this results because the start-up conditions were unknown. It was decided to see what would happen if the crystal oscillator of the first microprocessor was placed across the clock circuit of all microprocessors at the same time. With a counter attached to the CLK-OUT (clock-out) signal, we measured the variation in frequency. Table 1 shows the result of attempting to synchronize by using a single crystal oscillator. As can be seen, the frequency will decrease somewhat as more processors are connected.

# of Processors	Measured Frequency
1	3.07437 Mhz
2	3.07300 Mhz
3	3.07259 Mhz
4	3.07225 Mhz

Table 1. Frequency Variation of Single Crystal Technique.

Error Definition

Once we are sure that a group of processors are executing the same code at the same time, the next problem is to define what is considered to be an error. Since the bus is being monitored to determine if a fault has occurred, then addresses, data, and status must be continually observed. Addresses may be generated by the processors. Data may be generated by the processors, memory or input devices. Status may be generated by the processor or input devices. In the case of the Intel 8085, the status lines are WRITE, READ, IO/M, INTA, reset-out and the processor state lines S1 and S2.

In our study, research as limited to commerical microprocessors. Intel 8085 System Design Kits (SDK-85) obtained to form the basis of this study has the extra board space on the kits which allowed a TMR network to be placed in close proximy to the system. The size of the board also limited the error detecting to the addresses and the data only. This was due to the limited space for adding busses necessary for minitoring.

Errors can exist on address, data and status lines. In developing a TMR system, we must know when to check for errors. By observing the instruction cycle, Figure 4, we find that it is divided into a number of machine cycles namely, the opode-fetch, memory read cycle, memory write cycle, I/O read cycle, I/O write cycle, interrupt acknowledge cycles, and the bus idle machine cycles. All instructions consists of at least an opcode fetch cycle.

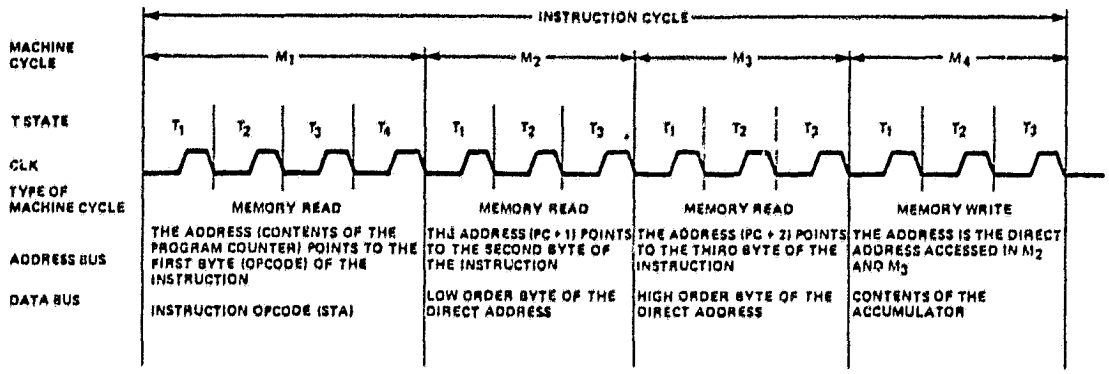

Figure 4. Typical Instruction Cycle.

Let's look at a typical instruction--store register. First the processor will enter the opcode fetch cycle in which the contents of the program counter is placed on the address bus. The READ command will be issued and the memory will respond by placing the opcode on the data bus. Next we enter two memory read cycles, one for the least significant address byte and one for the most significant address byte for the storage of the register content. In each of these memory read cycles, Figure 6a, the contents of the program counter will be placed on the address bus, the READ command will be issued and the memory will place a byte on the data bus. Finally, the processor will enter a memory write cycle (Figure 6b). The address obtained in the two memory read cycles will be placed on the address bus, the WRITE command will be issued and the contents of the register will be placed on the data bus.

The ultimate goal in this project is to detect errors instanteously and correct the system by bring in a new spare as soon as possible. This means every address and every data transfer must be checked. By observing the Figures 5, 6a, 6b, and 7 we see that addresses are valid whenever the ALE line is active and data is valid whenever WRITE, READ, or INTA is active. These lines can be used to derive the times for checking addresses and data.

There are limitations on the size of the error detecting network based on the speed of the microprocessor. The 8085 uses a 6.144 megahertz (Mhz) crystal. This frequency is divided by a factor of two to obtain a clock out signal of 3.072 Mhz. The period of this clock cycle is 325.5 nanoseconds

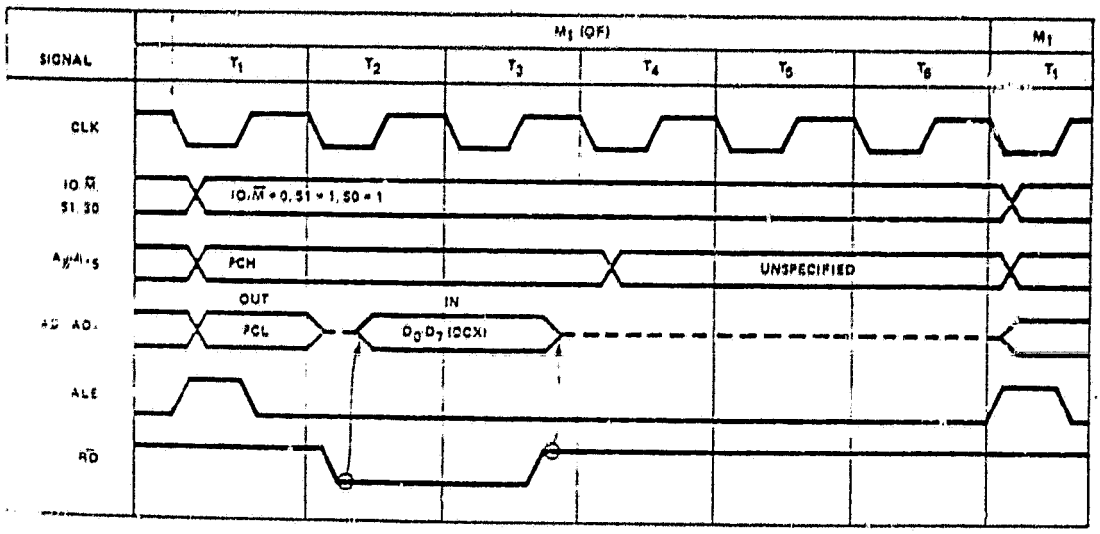

Figure 5. Op-code Fetch Machine Cycle (Intel).

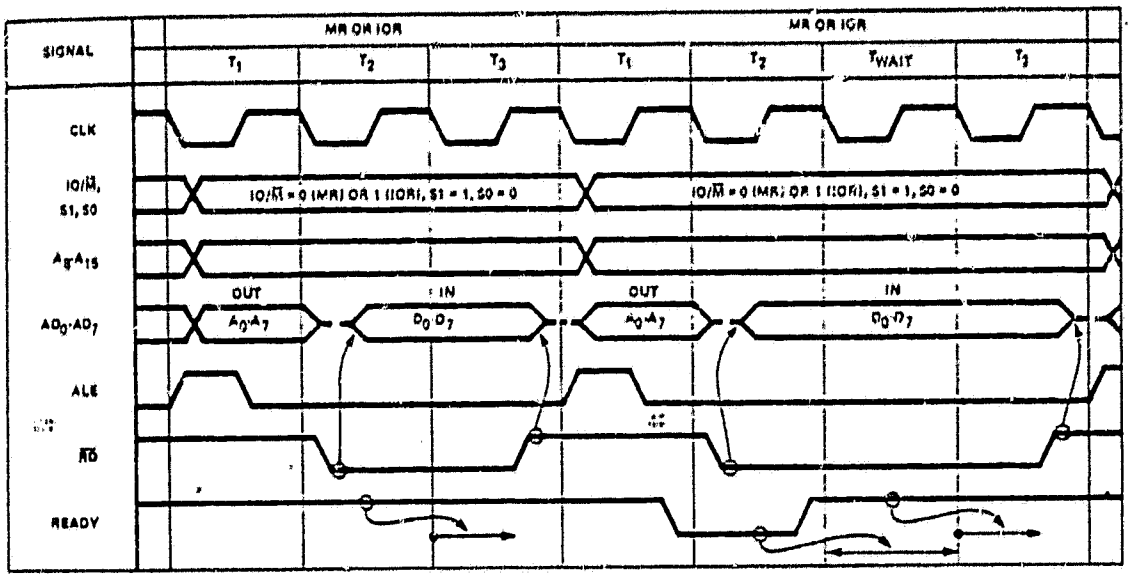

Figure 6a. Memory READ Cycle (Intel).

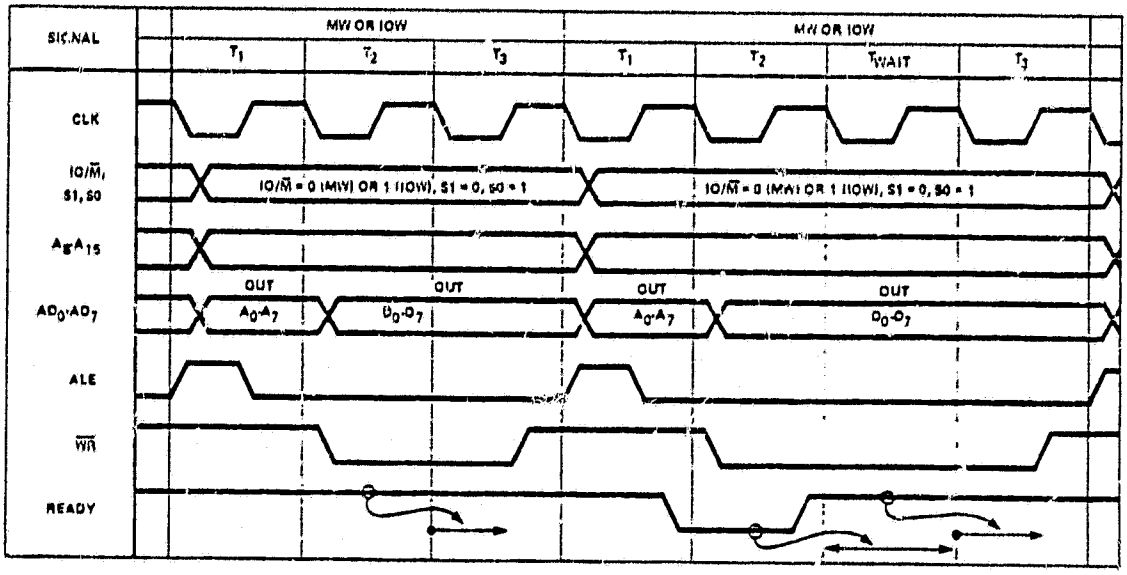

Figure 6b. Memory WRITE Cycle (Intel).

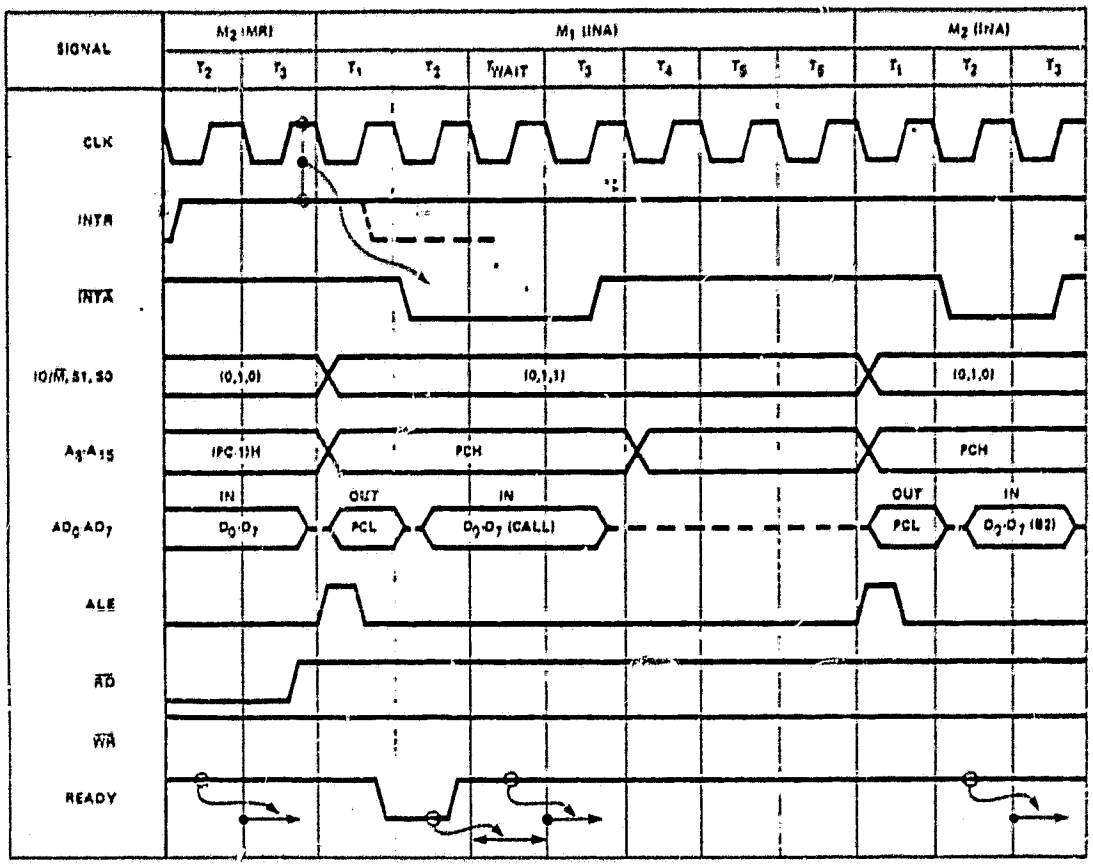

Figure 7. Interrupt Acknowledge Cycle (Intel).

(ns). From Figure 5 we see that addresses are valid on one clock cycle and data is valid on the very next clock cycle. This means that the checking must be accomplished in one clock cycle time period (325.5 ns) or less, or the system has to be slowed by using a READY signal.

Should an error occur, the system must store the state of the processors, update a new memory, and restore the state before a spare can replace a failed microprocessor module. This will require many machine cycles and hence the required time to perform this operation is greater than 325.5 ns. Therefore, the computer system must be temporary stopped when a new spare is to be made active.

The question aries, "What if an error occurs during a critical timed I/O operation?" This condition can be handled by the hardware of the Error Recovery System and the software generated by the programmer for this system. Prior to entering a critical timed I/O operation, the software programmer can disable the Error Recovery until the system exits the critical timing period. The addresses and data used during this period can be the "voted" addresses and data.

TMR Controller

The key to the entire design is the TMR controller. Its purpose is to 1) bring the microcomputer modules up in a TMR configuration by selecting three modules as the heart of the system, 2) allow the system to continue to run until an error is detected, 3) disallow any interruptions during a critical I/O operation, 4) detect and identify failed modules, 5) cause all modules to save their state of operation when an error is discovered, 6) select a correct module's memory to copy and write the RAM section of the new processor, 8) restore the state of operation after the memory update, and 9) start looking for another error.

The state diagram for the TMR controller is shown in Figure 8. After the system is reset, the TMR controller will set the processor selectors to 00. Thus, the controller observes the buses of the three main microcomputer modules. The controller then goes to a RUN state. The controller will remain in this state until either there is a critical I/O operation or an error is discovered.

If a critical I/O operation is to take place the controller will set the Critical I/O flip-flop (CIO). When this operation is complete, the CIO flip-flop will be reset.

If an error is detected, the controller will exit the RUN state and the CIO flip-flop will be tested. If the CIO flip-flop is set, the error detect and error recovery circuits will be disabled. The controller will continue to RUN with and error until the CIO flip-flop is reset. When this happens, the controller will issue a vector interrupt to all active modules. The modules

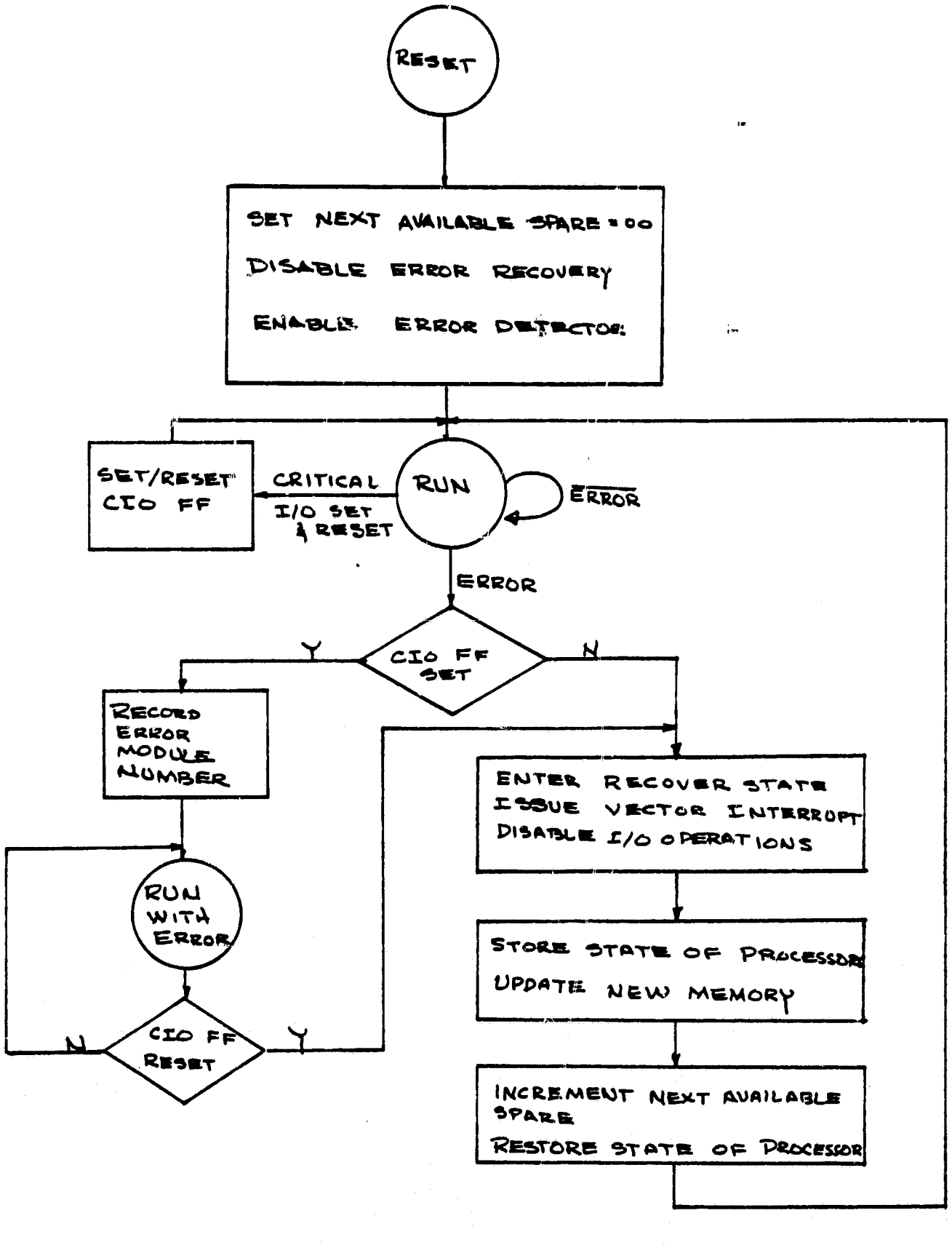

Figure 8. TMR Controller State Diagram.

stores all pertinent information regarding the present condition of the task. The failed module may have completely deteriorated by this time but the remaining good modules will have accurate information in their state vector.

The TMR controller must now select a correct module and update the memory of the new module to be brought into the main TMR configuration or a correct memory can be read and the information broadcasted to all memories of all processors. Along with the information transfer will be the state vector of the correct processor.

The next available spare (NAS) counter will be incremented to select the next module to be brought into the main configuration. The command is issued to restore the state of the machine. However, when this happen all processors have the same state vector. Finally, the TMR controller will go back to the RUN state and look for new errors.

The TMR controller block diagram is shown in Figure 9. The purpose of the Error Decoder is two-fold. First this component will be responsible for detecting error and second, it will identify which microcomputer module is in error. The diagram of Figure 9 shows the Error Decoder as having only a few output lines. In actuallity the lines shown must be multiplied by the number of address, data and status lines to be checked. Since only one error line and three ID lines are required, the concentrator circuits of Figure 9 are required. The ALE, WRITE, and READ lines are required to establish timing signals. These signals tell the TMR controller when addresses and data is valid. Since we do not know which ALE, WRITE, or READ signal will be

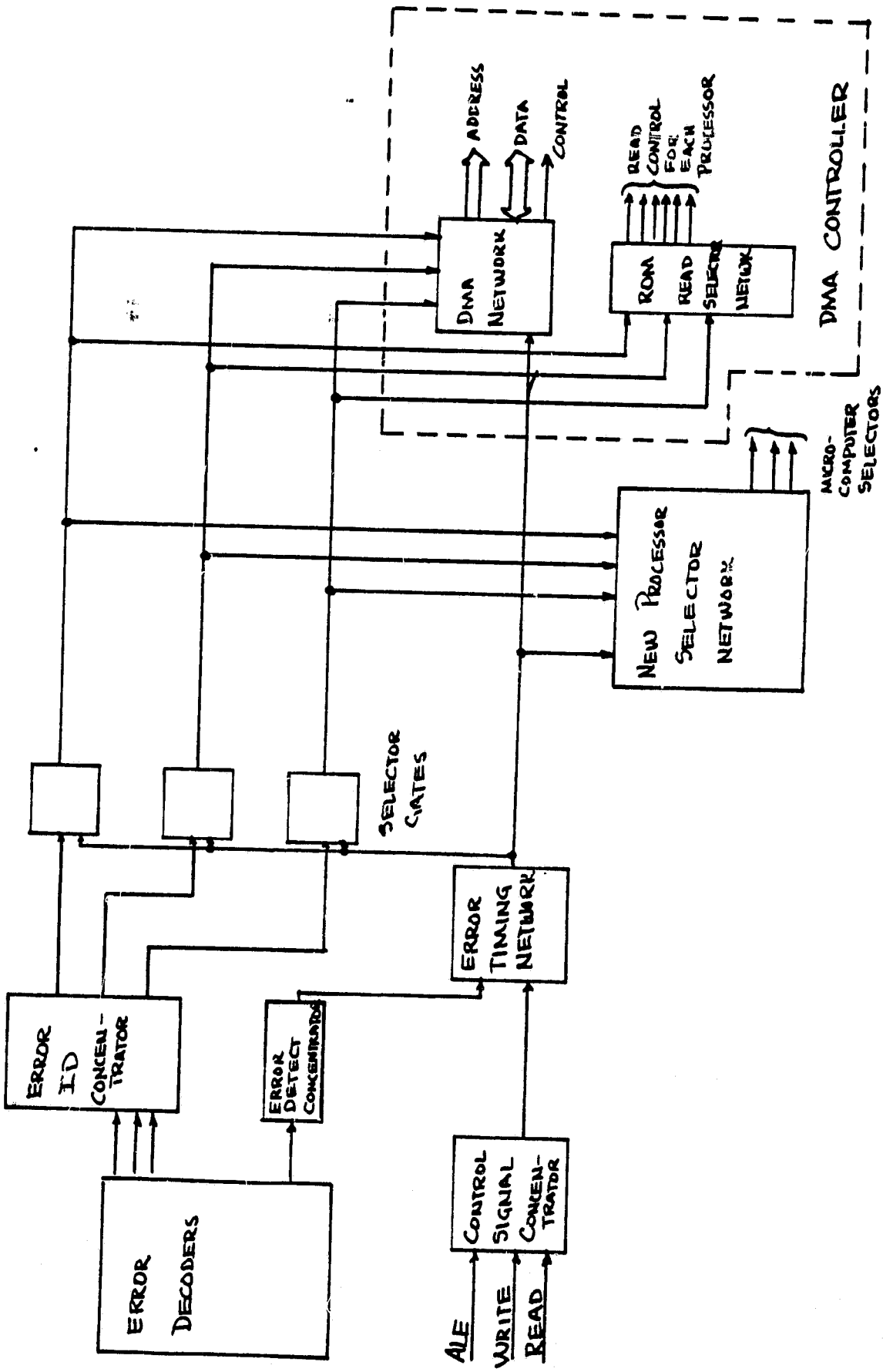

Figure 9. TMR Controller Block Diagram.

valid, it is necessary to concentrate all of them and vote on when data is valid.

The new processor selector, (NPS) function is to keep track of which processor is next to be brought into the TMR network. This is accomplished by using a clock and a counter. The counter is enabled only when there is an error. The output of the counter goes to a selector latch only if the corresponding microcomputer module is the one at fault. The network is shown in Figure 10.

The DMA controller has several functions. It is responsible for 1) creating the correct RAM addresses, 2) sending the READ command out to a correct microcomputer's memory, 3) receiving the data into a buffer, 4) broadcasting the WRITE command to all memories, 5) checking for last address, and 6) incrementing the address counter. The basis part of the DMA controller is shown in Figure 14.

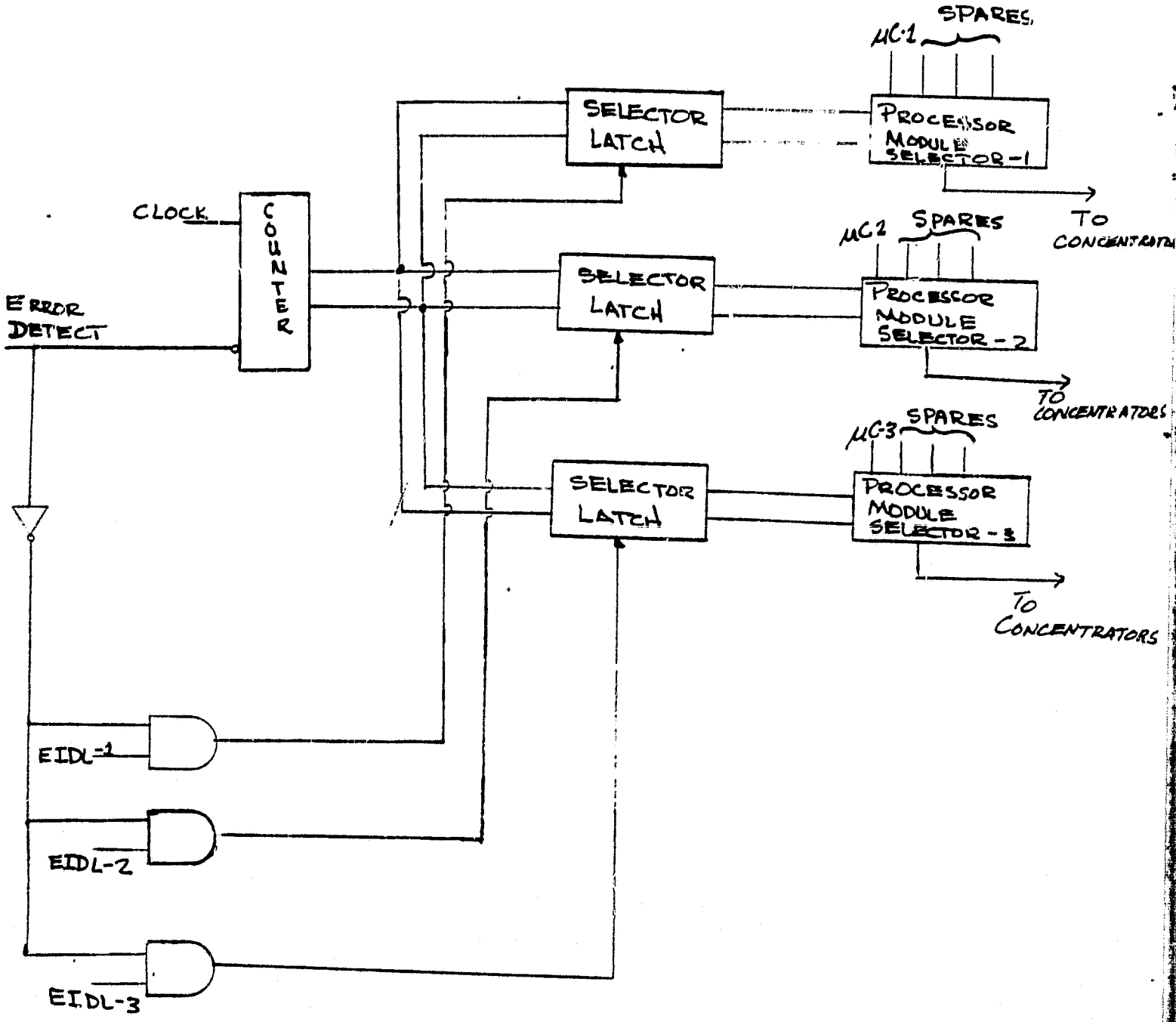

Figure 10. New Processor Selectors.

Error Detection

In this section we want to identify the point at which we detect errors. In the TMR configuration, the error detector is constantly observing data transfers of the three microcomputer modules. Lets look at the first data line D(0) of the three modules that make of the TMR portion of the network. Table 2 is a list of the eight possible states of this data line. If the error detector receives 000 or 111, then it is assummed that all modules are performing as required and there are no faults. If the results is either 011 or 100, then it is assummed that the first module is at fault. A "011" would indicate that modules 2 and 3 are reporting a '1' and module 1 is reporting a '0'. On the other hand, if "100" is recorded, then module 1 is reporting a '1' while modules 2 and 3 are reporting a '0'. Using the same argument, we can see that "101" and "010" would indicate a fault on module 2 and a "110" or a "001" would indicate a fault on module 3.

One method of detecting these errors is with the exclusive-or tree. However, the codes above could also be entered into a decoder. The advantage of using a decoder is obvious. If lines 0 7 of the decoder is activated, then there is no fault. If lines 3 or 4 are activated, microcomputer module #1 is at fault. Module #2 is at fault when line 2 or 5 becomes active, and module #3 is at fault when line 1 or 6 is active. This portion of the error detection network is shown in Figure 11 as the check decoders.

We can use the same decoders to determine the correct output.

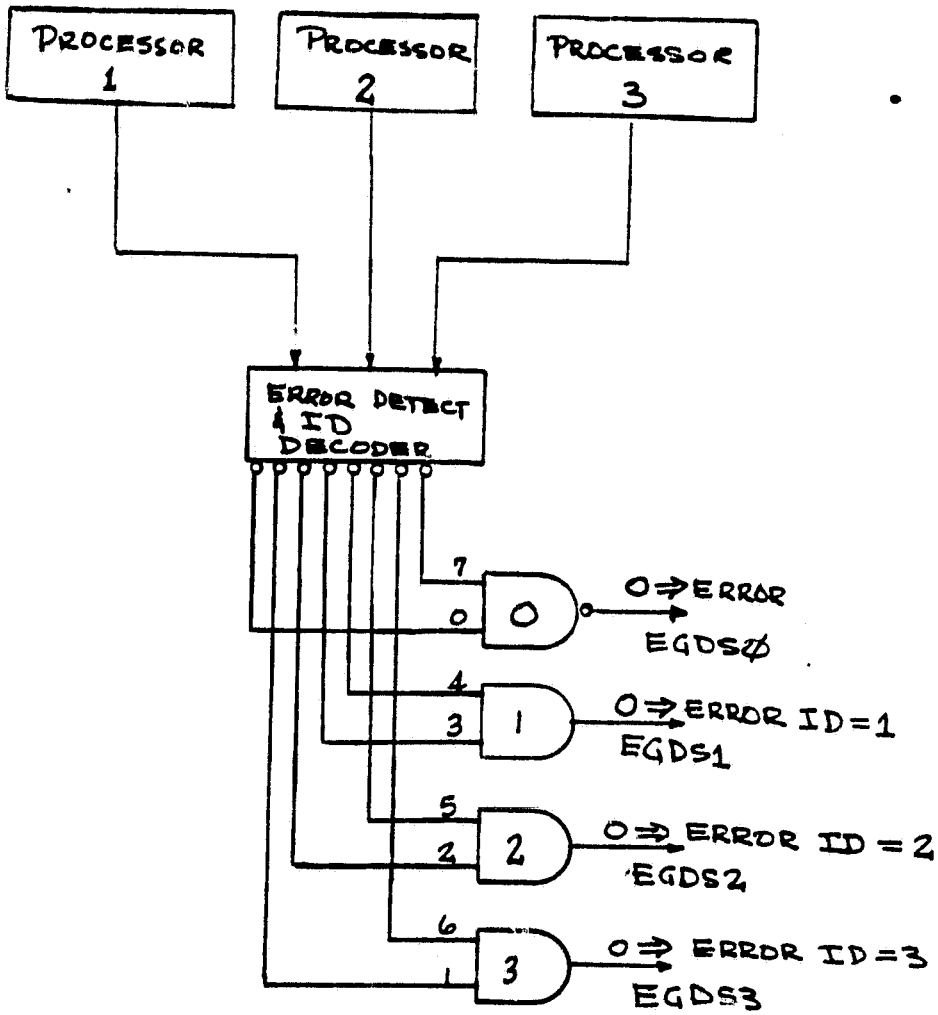

Figure 11. Error Detect & Identification.

By merely connecting the output of the decoder to a different network (Figure 13) the correct value can be obtained. This may be necessary in situations where we can not stop the processors to make a switch. By observing Figure 13 we find that what is required is to take lines 3,5,6,and 7 to a four-input nand gate. If the output is to voted to be a '1' then one of these four lines will be low causing the output to be a logical '1'.

There is also a need for this error detector to be self-checking. In observing FIGURE 11 we see that whethere there is a fault or not, the nand gates must maintain a condition in which there is only one logical '1'. The most probable fault is that the decoder does not decode, that is, all output remain high. Of course, another possible fault is that two or more lines are decoded at the same time. In either case, circuitry can be added to detect these failure conditions. The simpliest condition to detect is all output remaining high. This can also be done with a single 4-input gate.

Error Reporting

The scheme for detecting faults on one line is shown in Figure 11. The first problem is to determine how many lines of the bus we want to check. It was decided that the checking would be done over the address and data lines only. This decision was reached by considering the cabling width and cable count of the hardware that we used. In an actual system, the cabling size would not be a determining factor. Reliability would have highest priority.

The output of Nand gates 1 through 3 of Figure 11 is fed to the input of Figure 12 such that, the signal line EDG1 represents the error status of TMR module-1, EDGS2 represents the error status of TMR module-2, and EDGS3 represents the error status of TMR module-3. These three signals are latched to preserve the error information.

The output of Nand gate 0 of Figure 11 is fed into the upper network of Figure 12 to determine if a fault has occurred or not. The output of this network is fed into a one-shot to produce a latch pulse for the error detector network.

The next problem is to determine when we should look for an error. Since the address and data lines are to be checked, it is obvious that we must check whenever addresses or data is to be transferred over the bus. Observing Figures 4, 5, 6a, 6b and 7 we see that addresses are valid whenever ALE is high and data is valid when either WRITE, READ, or INTERRUPT ACKNOWLEDGE is valid. We use this information (in TRM form) to fire the one-

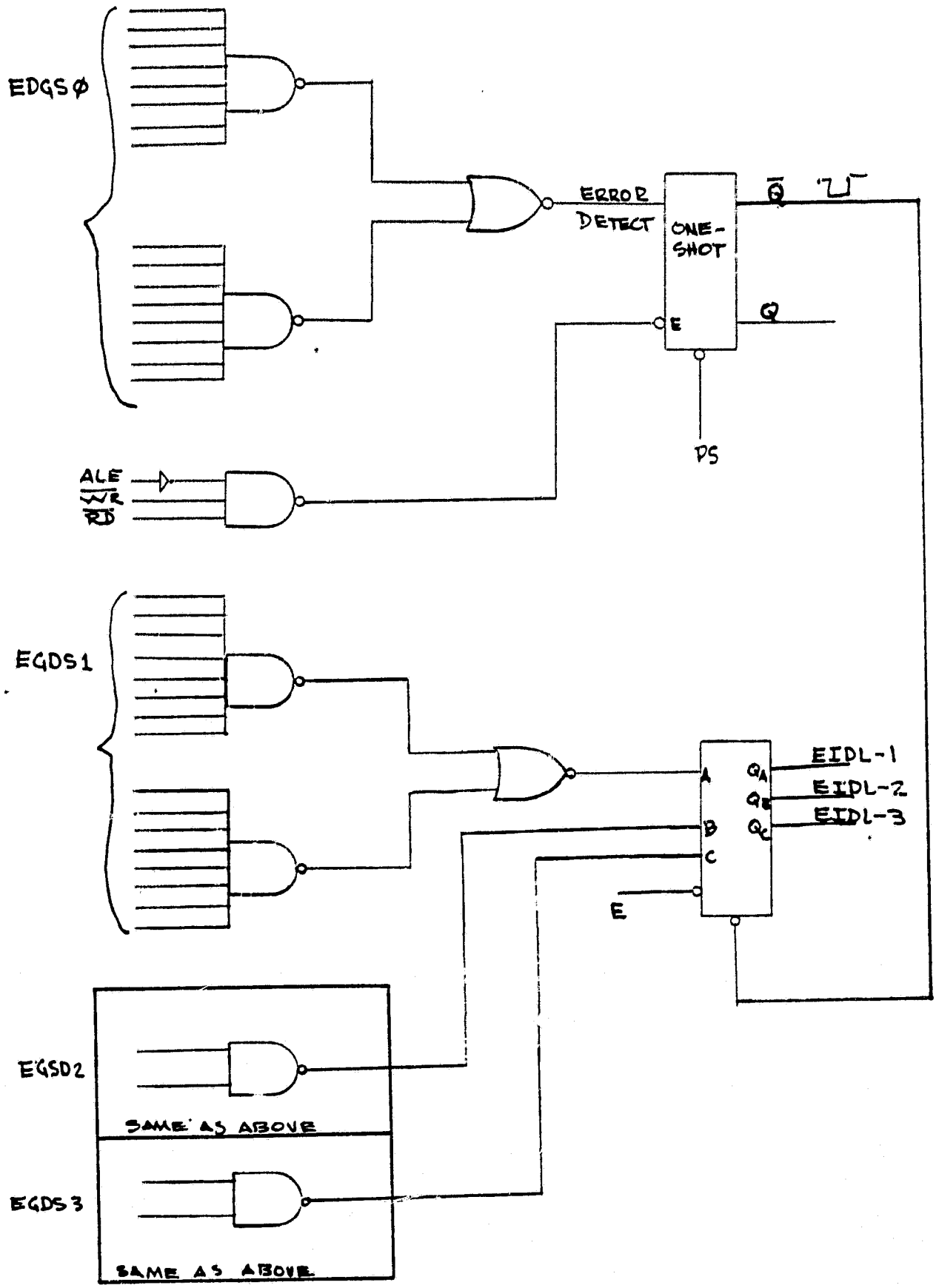

Figure 12. Error Concentrator.

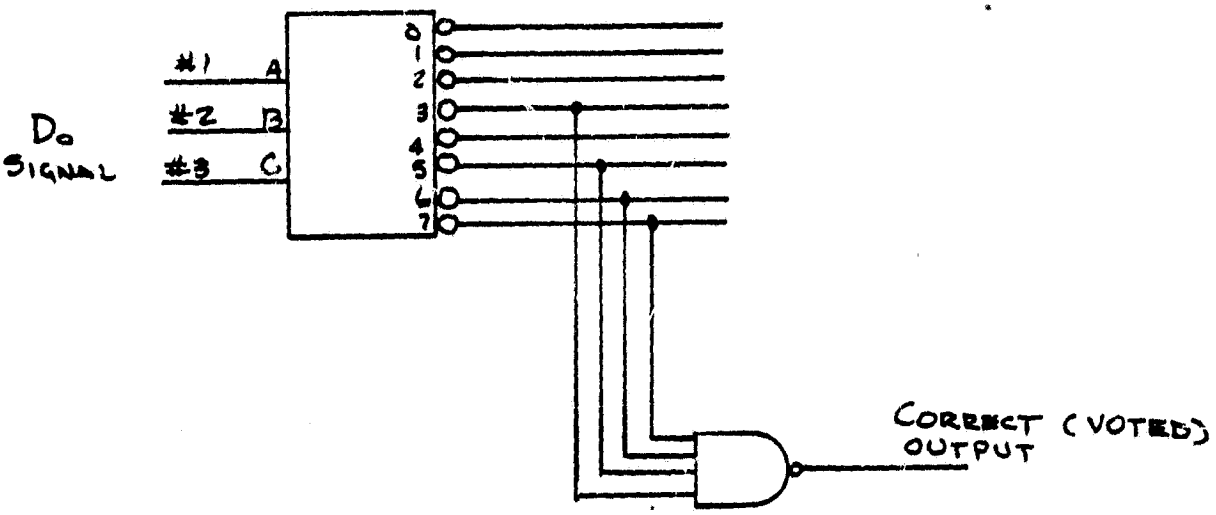

Figure 13. Correct Output Generator.

A B C	RESULTS
0 0 0	No Error
0 0 1	Error on Processor 3
0 1 0	" " " 2
0 1 1	" " " 1
1 0 0	" " " 1
1 0 1	" " " 2
1 1 0	" " " 3
1 1 1	No Error

Table 2. Error ID Codes.

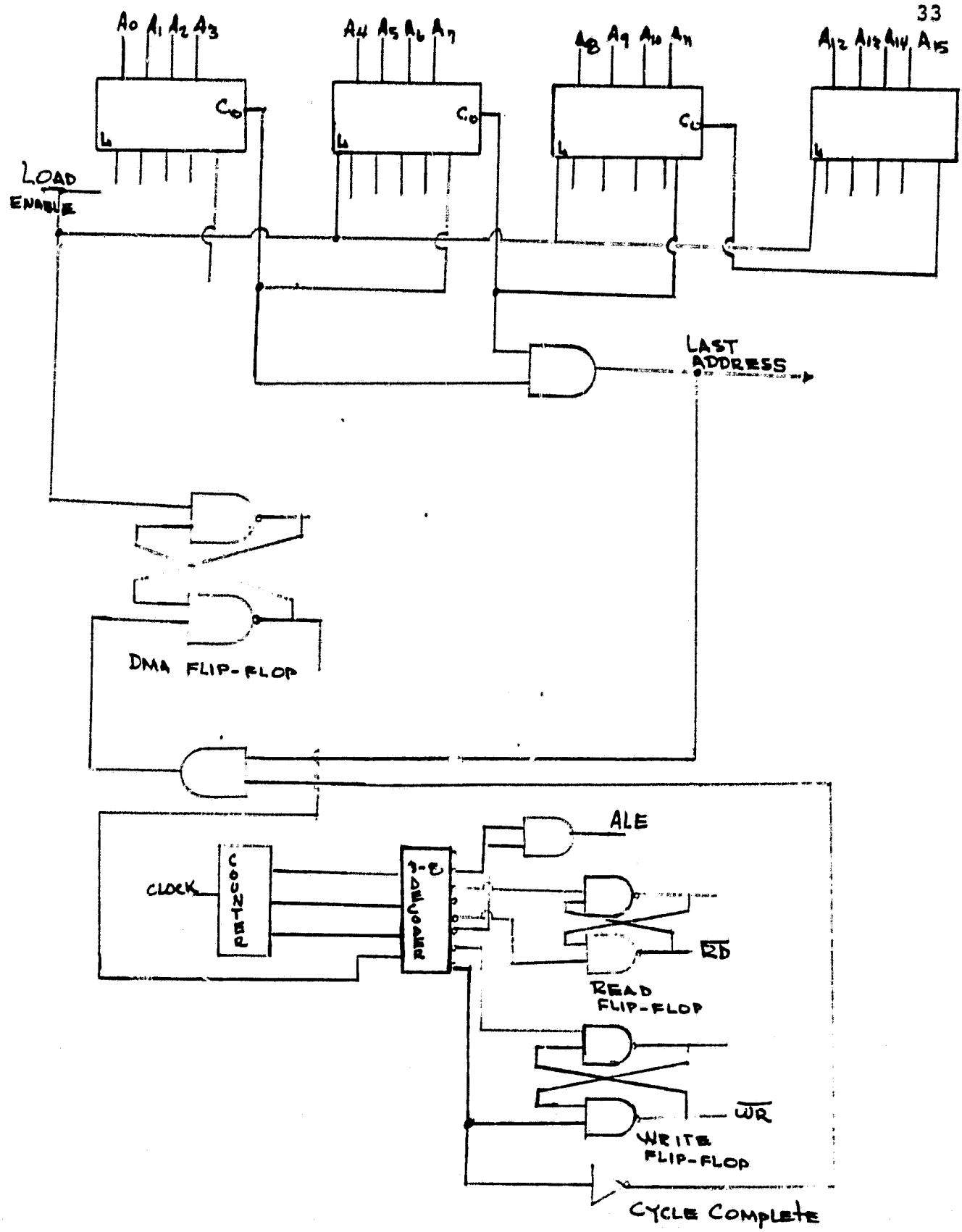

Figure 14. DMA Controller for Memory Update.

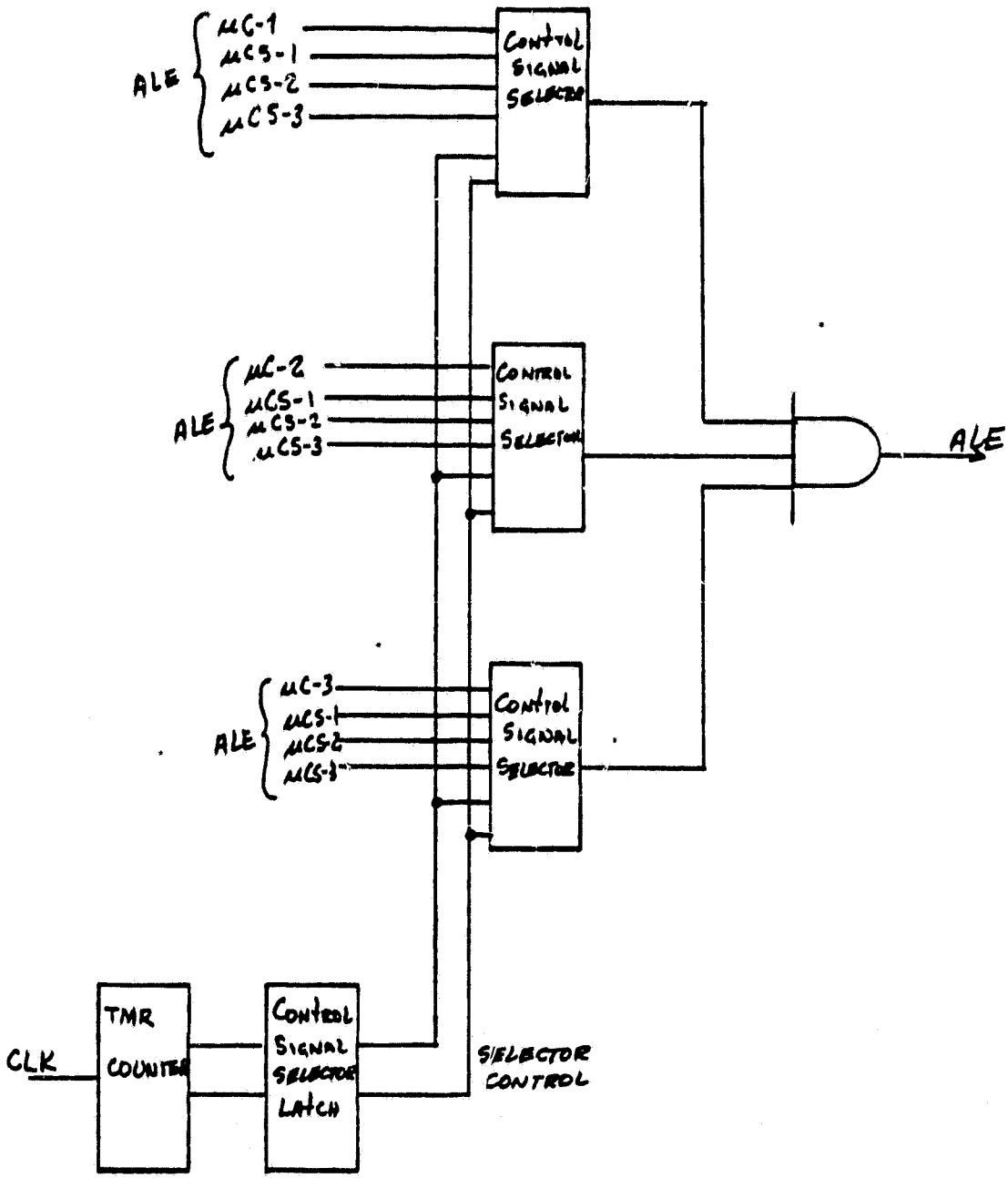

Figure 15. Control (ALE) Concentrator.

shot which will produce a latch pulse if there is a fault.

One problem tht should be solved is that of knowing which control lines (ALE, READ, or WRITE) to use to generate tne timing pulses for the error checking. Under single fault assumption, the failure is assumed to be associated with the address or data bus. We can therefore use the circuit of Figure 15 to concentrate the control line signals.

Memory Exchange

The exchange of data between a good processor's memory and a new processor to become a part of the TMR configuration is handled in four steps. The first step is to store the present state of the processors. This is accomplished by using a vector interrupt and the algorithm of Figure 16a. The second step is select a correct memory to read from. The third step is to use a DMA controller to read data into a buffer, one datum at a time, and to write the contents of that buffer to all memories. The fourth step is to restor the state of the processors by using the algorithm of Figure 16b.

A machine to select a correct memory to read from can be designed by using the state diagram approach. The state diagram is shown in Figure 17. The input to the state diagram is taken from the Error ID Latch (EIDL) of Figure 12. If EIDL is (111), then there is no error and the machine will remain in state S_0. If EIDL is 011, then the first microcomputer module has failed. Microcomputer module 2 or 3 can be use as the correct memory. The machine will go to state S_1. The output associated with state S_1 must indicate that memory 2 or 3 are available to be copied. We chose 2 to be specific. Lets suppose that EIDL has 101 while the machine is in state S_0. Microcomputer module 2 has failed. The machine will go to state S_2. Memories 1 and 3 are available for coping. Since we must chose one of these, we select the first one to copy. The selection of memories continues until there is two or less processors in the system. Presently, the machine is designed to go to a HALT state. However, more work can be done on this project to make the machine

```
PUSH    PSW
PUSH    B
PUSH    D
PUSH    H
MVI     B,'OOFF'
PUSH    B
LXI     H,O
DAD     SP
STORE   H,L   */ at special address
PUSH    H
STA     8000
```

Figure 16a. Software to Store the State of Processors.

```
LOAD    SP  */ from special address
POP     B
POP     H
POP     D
POP     B
POP     PSW
POP     PC
```

Figure 16b. Software to Restore State of Processors.

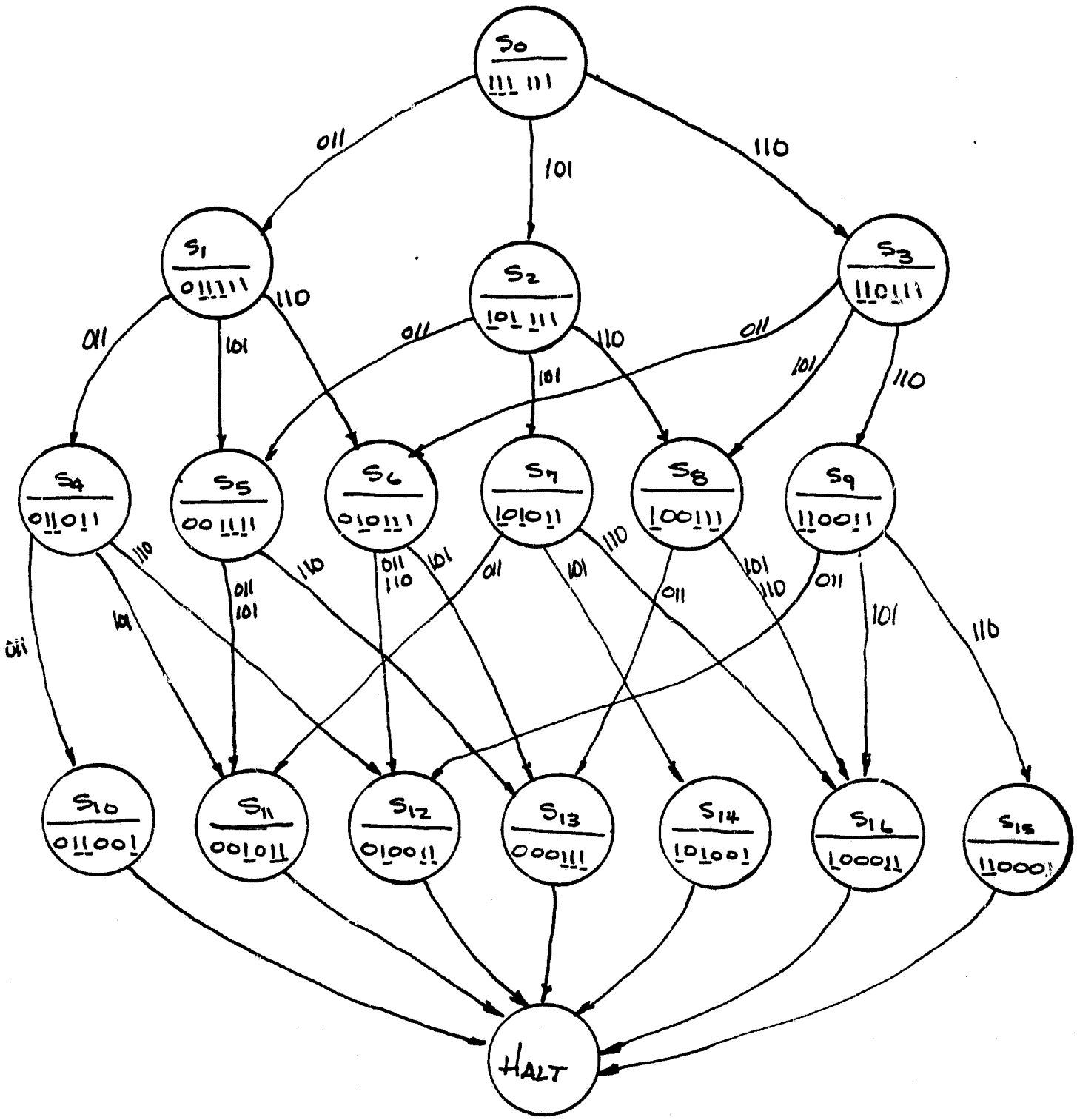

Figure 17. State Diagram for READ Select.

degrade more slowly. One final note on the machine. The bottom part of each state indicate the status of the processors. Where there is a 0 in a position, this represents an inactive processor. The positions with a 1 indicate a good processor and a position with an underlined 1 indicates an active processor.

The machine is implemented in ROM. Figure 18 indicate the essential locations and their content. The Address Field is partitioned into two parts, the input from Figure 12 and the present state. The Content Field is also partitioned into two parts, the encoded output and the next state. Figure 19 is a physical realization of the Memory Select ROM network.

The DMA controller consist of a small counter and decoder to generate DMA microcommands and four hexadecimal counters to generate the addresses for the memory exchange. The hexadecimal counters are concatenated such that the carry out of one stage becomes the clock-in of the next stage. The counters can be parallel loaded to start at any address. The DMA flip-flop is controlled by external logic and is reset by the Last Address Gate when the cycle is complete. The generated ALE, READ, and WRITE signals are produced by the decoder. First the ALE signal is generated to indicate that the addresses are valid. Then the READ command is issued to only one processor as specified by the ROM network. At this point data is transferred into a buffer. Next, the ALE signal is again generated and a WRITE command is sent to all processors. Finally, the Cycle Complete signal is generated. The controller is finished when a Cycle Complete signal and a Last Address signal is received.

ADDRESS IN P.S.	CONTENT OUT N.S.	ADDRESS IN P.S.	CONTENT OUT N.S.	
111 00000	000 00000	110 00111	011 10000	
011 00000	010 00001	011 01000	100 01101	
101 00000	001 00010	101 01000	001 10000	
110 00000	001 00011	110 01000	001 10000	
011 00001	010 00100	011 01001	010 01100	
101 00001	011 00101	101 01001	001 10000	
110 00001	010 00110	110 01001	001 01111	
001 00010	011 00101	011 01010	111 10000	HALT
110 00010	001 01000	011 01100	111 10000	STATE
011 00011	010 00110	011 01101	111 10000	
101 00011	001 01000	011 01110	111 10000	
110 00011	001 01001	011 01111	111 10000	
011 00100	010 01010	101 01010	111 10000	
101 00100	011 01011	101 01011	111 10000	
110 00100	010 01100	101 01100	111 10000	
011 00101	011 01011	101 01101	111 10000	
101 00101	011 01011	101 01110	111 10000	
110 00101	100 01101	101 01111	111 10000	
011 00110	010 01100	110 01010	111 10000	
101 00110	100 01101	110 01011	111 10000	
110 00110	010 01100	110 01100	111 10000	
011 00111	011 01011	110 01101	111 10000	
101 00111	001 01110	110 01110	111 10000	
		110 01111	111 10000	

Figure 18. Essential RCM Contents for Memory Select

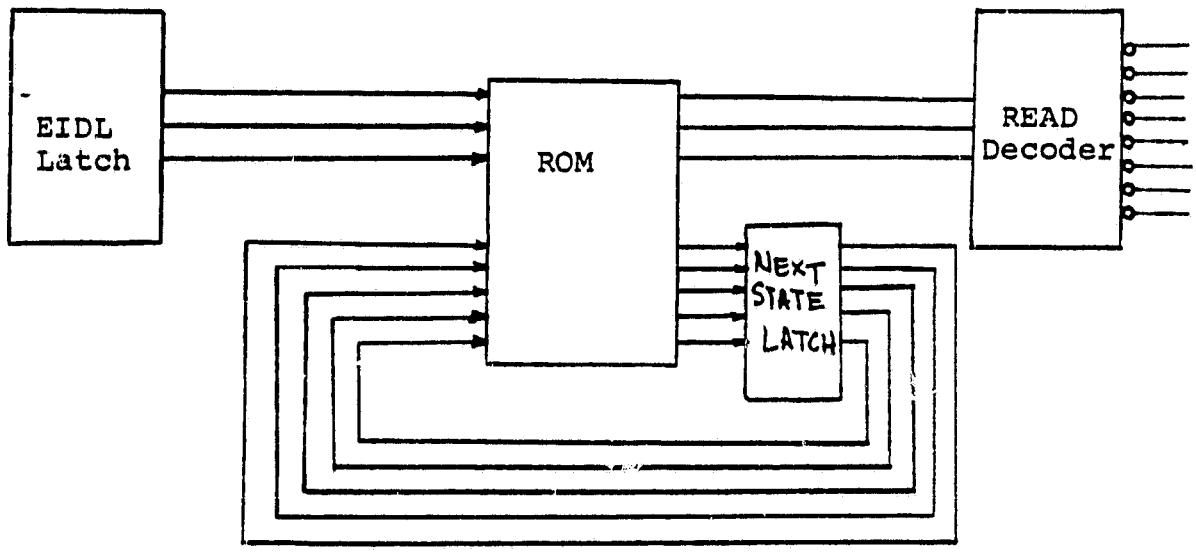

Figure 19. READ Select Network.

Summary

This project was started during a period when 8-bit microprocessors were the state of art. Now 16-bit and 32-bit microprocessor are almost common place. During this period microcomputers on a single chip are also common but they have little memory available on the chip. The methodology presented in this report is compatible with any microprocessor based computer system. However, if there were more test points available, the job of increasing reliability would be much easier.

Typical microprocessors have 40 pins available to the user. This means that little more than addresses and data are available. By placing the clock circuit inside the processor, almost no checking can be done on the timing circuit which is required to synchronize a TMR system. A tightly coupled real-time system is required for efficient aerospace computers. Therefore, the system designer must make use of every test-point available.

There are a number of different approaches. We can triplicate memories, add software checks or insert software breakpoints. But, the most efficient system is one in which testing is transparent to the processing of the system. This report presents such an approach.

Synchronization has been accomplished by using a single crystal. As the the number of processors are increased, the frequency decreases. The decreases are generally less than 100 hertzs per processor. This is a minimum decrease in performance, and a significant reduction in part count for this function.

Instead of using voters, our approach uses logic decoders for

detecting an error and identifying the failed module. Again the part count is reduced by getting two functions performed for the price of one.

The design also uses a ROM as a circuit element to select a good memory to copy when an error occurs. The alternative was to use a triplicated memory. This technique has several short commings. First, if one of the memory modules is lost due to the first failure, then the second failure takes the system down. Second, a system that is to be designed around one of the microcomputers-on-a-chip cannot use this technique.

Overall, the techniques examined during this report will aid in the design of a high-speed aero-space computer irregardless of whether that system is designed around 8, 16, or 32 bit microprocessors or microcomputer-on-a-chip.