

DESIGN OF A MODULAR DIGITAL COMPUTER SYSTEM
DRL 4

PHASE II REPORT

March 1, 1973

Prepared under Contract NAS8-27926

by

HUGHES AIRCRAFT COMPANY
FULLERTON, CALIFORNIA

for

George C. Marshall Space Flight Center
National Aeronautics and Space Administration

(NASA-CR-124171) DESIGN OF A MODULAR
DIGITAL COMPUTER SYSTEM Phase 2 Report,
16 Mar. 1972 - 1 Mar. 1973 (Hughes
Aircraft Co.) 308 p HC \$17.50 CSCL 09B

N73-33130

Unclas
20290

G3/08

PRICES SUBJECT TO CHANGE

FR 73-11-168

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
US Department of Commerce
Springfield, VA. 22151

FOREWORD

This report documents the accomplishments of Phase II of contract NAS8-27926 whose scope is the design of an Automatically Reconfigurable Modular Multiprocessor System (ARMMS). The Phase II time period was from March 16, 1972 to March 1, 1973. The design is being performed by the Data Processing Products Division of Hughes-Fullerton. M&S Computing, Inc. is providing support in the area of executive software design under subcontract to Hughes. The design is being directed by the Astrionics Laboratory of the Marshall Space Flight Center. The Contracting Officer's Representative is Dr. J. B. White.

In accordance with the data requirements of NAS8-27926, this report consists primarily of reproductions of internal reports not all of which have been edited or retyped for this report. As such, it reflects the evolution of the design rather than its culmination and must be read from that perspective. Each report is preceded by a brief discussion of its content and conclusions.

Major individual contributors to the report include the following:

<u>Author</u>	<u>Section</u>	<u>Subject</u>	<u>Page</u>
R. A. Easton	1	Summary of ARMMS Phase II	1-1
R. A. Easton	2	ARMMS Hardware Design - Phase II	2-1
T. T. Schansman K. H. Schonrock E. I. Eastin C. E. Turnour D. Hyde	3	ARMMS Control Executive System Design	3-1
S. Simpson B. Cohen R. Radys	4	ARMMS Component Technology Studies	4-1
J. Bricker W. L. Martin	5	Reliability Modeling in the Varying Configurations and Loads	5-1

Section 3 was not edited. The remaining sections were edited and this report prepared by R. A. Easton.

SECTION 1

SUMMARY OF PHASE II OF THE ARMMS DESIGN

1-2

SECTION 1

SUMMARY OF PHASE II OF THE ARMMS DESIGN

The primary objective of contract NAS8-27926 is to perform the system design of an advanced modular computer system designated the Automatically Reconfigurable Modular Multiprocessor System (ARMMS).

Any computer system justifies the cost of its development to the degree that it provides new capabilities or allows earlier ones to be satisfied at reduced cost. ARMMS is primarily oriented toward providing the following new capabilities for spaceborne computers for application in the 1975 to 1985 time period.

1. To provide a modular computer system which is responsive to many mission types and phases.
2. To achieve through modularity a higher computing capability than previously available for spaceborne application. A target of several million instructions per second has been chosen.
3. To provide the capability to choose to maximize reliability through the use of redundancy or to maximize processing capacity through multiprocessing. Moreover, this multi-mode capability must be dynamic; that is, a given system may alternate from one mode to another as a function of real-time requirements.
4. To maximize reliability in all applications through the incorporation of fault detection and recovery features and through the use of high reliability components.

The first consideration of any ARMMS design tradeoff is to avoid compromising these basic objectives. However, an advanced paper design will surely remain only that unless continuous concern is maintained for the practical requirements of implementation. Such design parameters as power density, weight, volume, pin count, device count, etc., must influence the design process.

The evolving baseline as presented here is oriented toward achieving the ARMMS objectives within a practical hardware and software context.

ARMMS is an outgrowth and extension of two NASA development programs, the MSFC Space Ultrareliable Modular Computer (SUMC) and the ERC Modular

Computer. The SUMC program has emphasized the development of a processor which is effectively partitioned for LSI implementation. To date, a breadboard TTL prototype has been constructed and a MOS LSI version is nearing completion.

A modified version of SUMC is anticipated to be the processor module of the ARMMS system. The breadboard of the ERC Modular Computer is undergoing evaluation at MSFC, and the experience gained will be relevant to ARMMS. The ERC Modular Computer had the common objective with ARMMS of achieving a variable configuration for varying levels of processing capacity and reliability.

In addition, the experience of numerous NASA, Air Force, and Navy architecture and design studies is being reviewed and incorporated into the ARMMS design where appropriate. In general, these efforts have considered a subset of the ARMMS objectives. For example, the JPL STAR is oriented toward long-life reliability. The MSC reconfigurable guidance and control computer study considers primarily space shuttle requirements. Other studies have considered space station computer requirements. All have identified design principles which form a substantial base of experience for the ARMMS development.

The 24-month contract is divided into three phases. The program plan covering Phases II and III is shown in Figure 1. At the inception of the contract, an initial baseline description was provided by MSFC. The primary efforts in Phase I was to establish general design guidelines necessary to achieve the ARMMS reliability and performance objectives; to survey published estimates of performance requirements for future space computers, and to refine the initial baseline.

The specific objectives to be achieved within the 24 month period are the following:

1. To perform the detailed design (to the gate level) of all module interfaces and switches.
2. To define the design of all ARMMS module types to the detailed block-diagram (register) level.
3. To perform the functional design of the executive software as it pertains to error detection and correction. (M & S Computing, Inc. is performing this task under subcontract to Hughes.)
4. To define the overall system response to all classes of failures.
5. To develop sample packaging concepts for an eventual implementation of ARMMS.

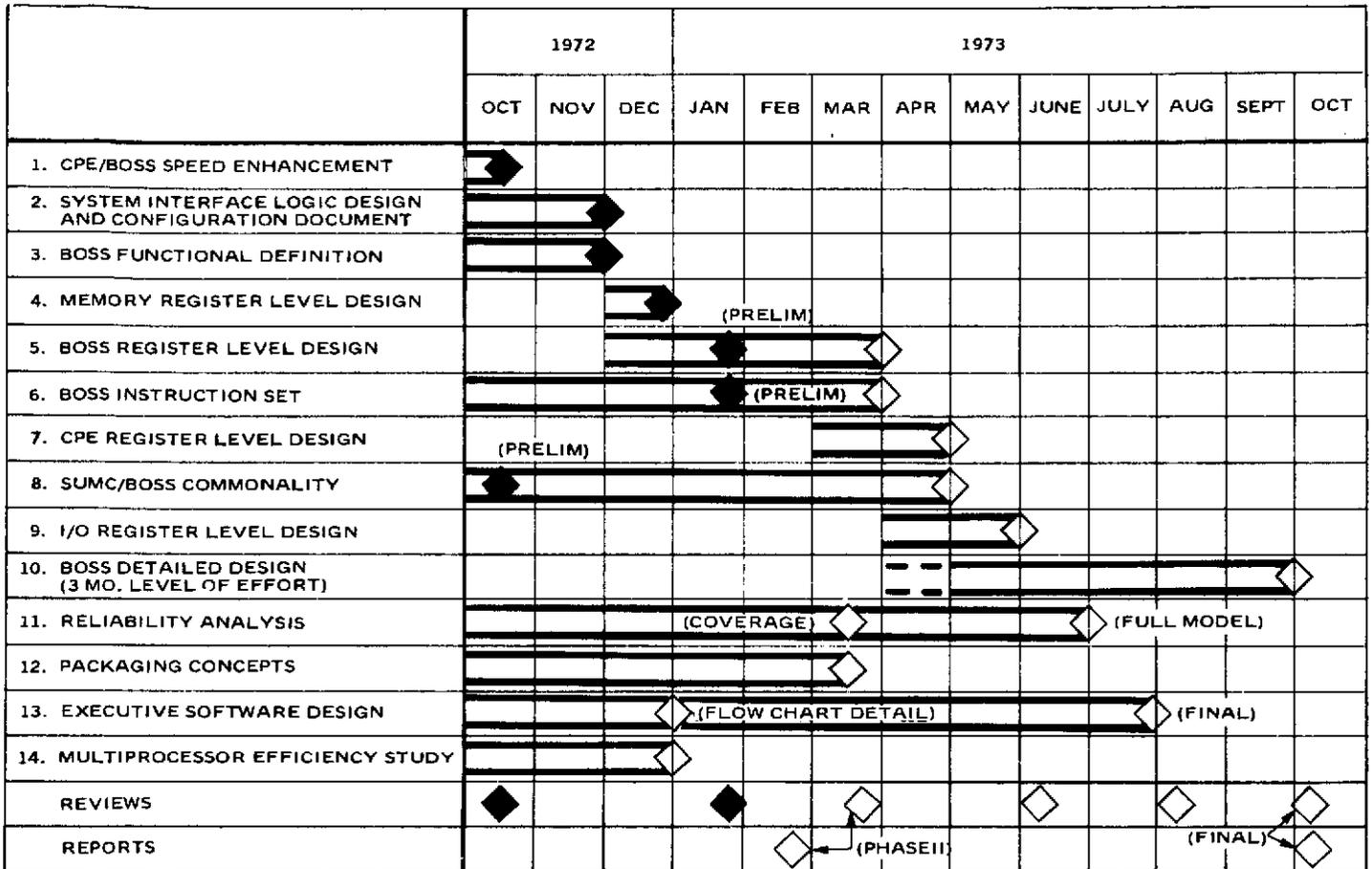


Figure 6. ARMMS Design Plan

6. To perform a 3 month level of effort detailed processor design study.
7. To develop and apply reliability models as needed to support the design.

All ARMMS work is expected to be completed on schedule with the following exceptions: The Add-on to contract NAS8-27926 originally specified a "Simulation of ARMMS Performance" task. During the time since that task was specified it has become apparent that this work would largely duplicate efforts along the same line by Computer Sciences Corporation under contract NAS8-21805. Therefore, it is proposed that an equivalent 3 month level of effort in the June through August 1973 period be devoted instead to the detailed logic design of the ARMMS Block Organizer and System Schedule (BOSS) module, and/or to detailed logic design of reliability and speed enhancement

modifications to the SUMC processor. Results of work performed to date on the BOSS register level design indicate that it should be possible to achieve a considerable degree of commonality between a modified SUMC CPE and BOSS. Detailed logic design in these areas is of prime importance in ARMMS because of their unique features whose characteristics cannot be directly extrapolated from earlier computer experience. The greater level of detail will also provide benefits to any follow-on to ARMMS and be directly applicable to an ARMMS breadboard. Specific areas to be investigated as time permits include but are not necessarily limited to the following areas not included in the original BOSS Detailed Design Task:

1. Detailed specification of the contents of the BOSS microprogram read only memory.
2. Detailed logic design of BOSS sequence control logic including instruction overlap. Assess the degree of applicability of this controller to a SUMC CPE.
3. Detailed logic design of BOSS and/or CPE error detection and masking logic.

As specified in the contract data requirements, the phase reports are to consist primarily of reproduction of contractor internal documents written during the phase. The remaining sections of this report consist largely of documents prepared at various stages of Phase II together with some new and updated material. The general subject of each is listed below:

Section 2 - ARMMS Hardware Design - Phase II

These studies started with an evaluation of the SUMC processor and suggestions for its modifications to increase its effectiveness in ARMMS. Next, 3 module configurations are discussed - one of which is recommended as the base-line for Phase III. Error detection and correction strategy for other configurations is then discussed followed by preliminary BOSS module and memory module register level designs with emphasis placed on an analysis of potential failure modes and techniques for detecting and/or masking them.

Section 3 - ARMMS Control Executive System Design

This section is M & S Computing's final report on their Phase II software design efforts. It covers software philosophy, task control, event recognition and response,

resource allocation and control, fault detection and diagnostic processing, information protection and input/output control. In addition an independent evaluation of ARMMS impacts on the baseline SUMC processor is included.

Section 4 - ARMMS Component Technology Studies

This section deals with ARMMS component technology studies involved in choosing a logic family, data bus technology, and power supply configurations. CMOS logic was chosen for module internal functions after consideration of all major logic families' projected characteristics for the 1975 time frame. CMOS will be assumed in developing ARMMS packaging characteristics during Phase III. The data bus studies placed an emphasis on loading considerations, detection theory, module interconnection methods, and reliability. Seven power supply configurations ranging from a single centralized supply to individual supplies per module were considered. A partially centralized configuration was chosen for further detailing in Phase III packaging studies.

Section 5 - Reliability Modeling with Varying Configurations and Loads

This section consists of the manuscript of a paper describing the most recent reliability model developed for ARMMS considering configuration and computation load refinements which can vary at deterministic times during the mission. It was delivered by J. Bricker at the 6th annual IEEE Computer Society International Conference (COMPCON) in September 1972. The remainder of the section discusses tradeoffs concerning whether to place ARMMS voter switches internal or external to processor and memory modules.

BLANK PAGE FOLLOWS

SECTION 2

ARMMS HARDWARE DESIGN - PHASE II

This section deals with the efforts in the area of ARMMS hardware design during Phase II. The studies started with an evaluation of the SUMC processor and suggestions for its modification to increase its effectiveness in ARMMS. Next ARMMS module configuration studies are discussed - three alternative configurations are proposed, one of which is recommended as the baseline configuration for Phase III. Given this configuration, error detection and correction strategy is discussed for providing maximum reliability and performance at a minimum cost for special hardware. Finally a preliminary BOSS module and a memory module register level design are described with particular emphasis on an analysis of potential failure modes and techniques for detecting and/or masking them. The level of detail of BOSS design, permits descriptions of microprogram and scratchpad memory organizations, integrated circuit partitioning estimates, and assumption of a preliminary standard and macro instruction set. Refinement of these designs and designs of ARMMS I/O and CPE modules will occur during Phase III.

I. Evaluation of the SUMC Processor for ARMMS

During Phase II of the ARMMS study MSC's SUMC processor was evaluated to find ways to enhance its speed and reliability in the ARMMS context and to assess its potential for commonality with ARMMS BOSS executive module. The existing SUMC design is excellent for many applications, especially those involving missions considerably shorter than 5 years and requiring frequent use of multiply, divide and square-root instructions. Other implicate changes to SUMC such as the width of busses to and from main memory are covered later in this report.

Speed Enhancement

An evaluation of the speed limitations of SUMC in ARMMS determined that the biggest speed bottleneck is likely to be the SUMC logic itself. Assuming either low-power MSI Schottky TTL (1973 time frame) or projected LSI CMOS using a silicon on sapphire technology (in the late 70's) maximum microinstruction clock rates would be on the order of 4 MHz. Data bus transmission from main memory to processor would be accomplished at 2 to 3 times this rate and main memory cycle times on the order of 800 nsec should be easily attainable at low power using plated wire techniques - hence these two areas should not be a problem. Using these numbers, the average instruction requires 3.5 μ sec to execute (examples: Add \approx 3 μ sec, Divide \approx 9.5 μ sec, jump \approx 2 μ sec). Such a processor should require less than 75 watts of power.

It should be possible to modify SUMC to reduce its power and complexity by some 40% while increasing its speed 50% and increasing its commonality with the sort of processor that would be required to execute BOSS functions efficiently and make the addition of error correcting circuitry more straightforward. These improvements are achieved through instruction overlap and logic simplifications.

An average speed increase of from 30 to 40% can be achieved by instruction overlap - i. e., fetching the next instruction while executing the present instruction thus saving memory access and bus transfer time. In the best case two overlapped cycles correspond to one non-overlapped cycle and a program can be executed twice as fast as before. This occurs when a program consisting of short instructions such as LOAD and ADD is accessing a memory with no contention from other programs. The worst cases occur on JUMP instructions, STOREs of data generated in the immediately preceding instruction, or when two programs both consisting of short instructions are in heavy contention for the same memory page. In these cases overlap becomes ineffective and the program runs at the same speed as it would have without overlap. The average speed increases noted have been verified by

computer simulations performed by Don Taylor of Computer Sciences Corp. These speed increases allow reducing the average instruction execution time to 2.5 μ sec at a 4 MHz microinstruction clock rate.

Instruction overlap logic should amount to about a 5% increase in complexity for ARMMS including increases in both the SUMC CPEs and the main memory modules. The added logic requirements include:

1. Logic to inhibit overlaps on JUMP and some STORE instructions .
2. Duplicated instruction registers to allow push-pull MROM access.
3. Memory address and data buffering.

For more details on instruction overlap timing you are referred to page 2-27 of this report entitled "Interface Timing. "

Once an instruction has been fetched it must follow the critical path shown in Figure 1, during the execution of each microinstruction step. Note that two adders are included in SUMC to speed up multiply, divide and square root operations. If only one adder were included in SUMC rather than the present two, the hardware would be reduced by about 10% and the clock rate could be increased by about 25% due to the decreased propagation delays, speeding up all operations except Multiply (M), Divide (D) and Square-Root (SQR) by 25%. The M, D, and SQR instructions would require approximately 70% more micro instructions than they do presently, hence they would take 26% longer to execute than presently. However, except for programs requiring large numbers of M, D, SQR operations, SUMC's speed would show a net increase (5% if all instructions are assumed equally likely to be executed). Only for programs with more than 25% multiply, divide, square-root instructions would any speed reduction be noted. Such programs are considered unlikely for ARMMS. Hence removing one adder will not degrade SUMC as a CPE in most applications and since multiply, divide, and square root are not needed in BOSS the modified SUMC CPE logic would exhibit greater commonality with BOSS logic. A final advantage to removing one adder is that this reduces the amount of redundancy needed in the system since adders cannot be checked using the same error detecting/correcting coding techniques proposed for the rest of ARMMS and hence would require duplication and comparing of outputs if their failures are to be detected. For all these reasons we recommend using only one adder in SUMC.

In the interest of reducing power and complexity it would also seem desirable to cut the number of words in the MROM and scratchpad memories by 50% and to eliminate the IAROM - accessing the MROM directly from the instruction register. The CPE instruction set uses only 25% of the MROM words

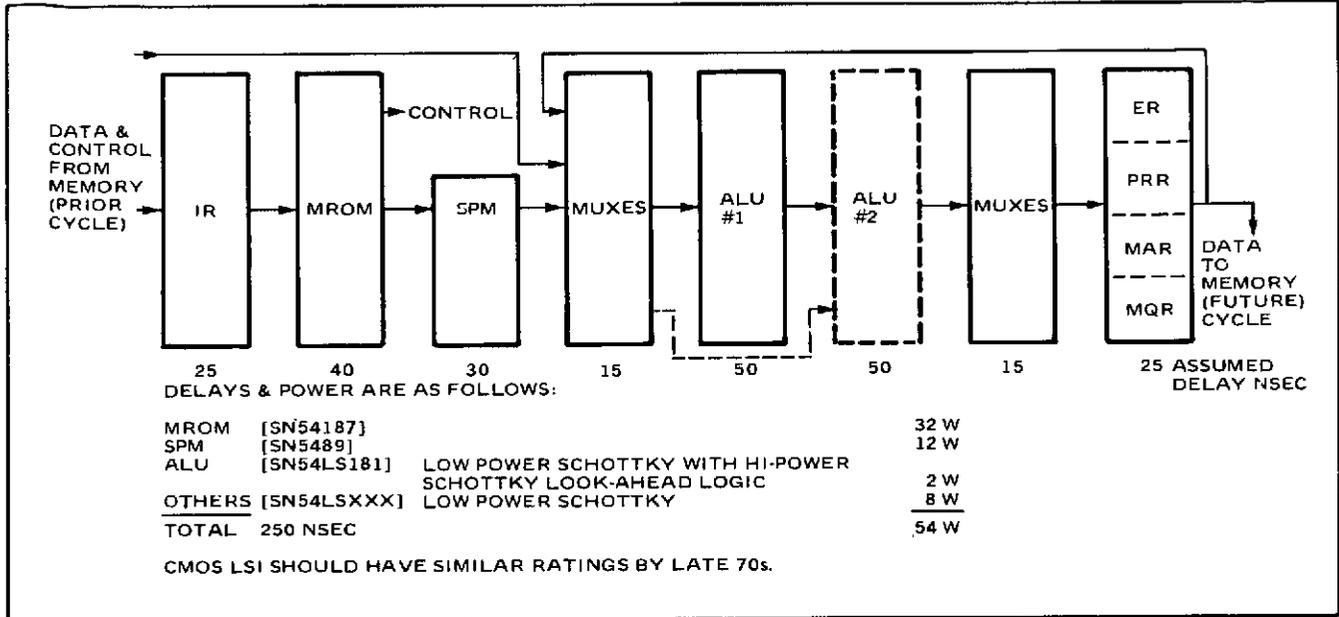


Figure 1. Critical Path Through Baseline SUMC

and 40% of SPM words. An 8-bit operation code field in instruction words from main memory would have adequate bits to address the MROM directly and efficiently allowing elimination of the IAROM function. Since fast ROM and SPM integrated circuits would consume 80% of an unmodified SUMC's power, these changes will reduce SUMC power by 40% and reduce its logic complexity 25%.

Comparison of modified SUMC's speed and power with that of the Hughes H4400 and JPL STAR computer processors strongly favor SUMC. Modified SUMC's speed-power product is an order of magnitude better than for H4400 or for STAR and 3.3 times better than for the unmodified SUMC. The comparison is summarized in Table I.

TABLE I. COMPARISON OF MODIFIED SUMC PERFORMANCE
TO H4400 AND JPL STAR COMPUTERS

	MOD. SUMC	H4400	JPL STAR
JUMP μ sec	1.4	1.4	36
ADD μ sec	1.2	1.4	36
MULT μ sec	5.0	6.0	SOFTWARE
DIVIDE μ sec	9.6	10.8	SOFTWARE
SQ ROOT μ sec	7.4	8.6	SOFTWARE
MEM. CYCLE μ sec	0.6	0.7	18
CLOCK MHz	10	8.33	0.5
POWER watts	30	425	10

CPE/BOSS Commonality

A study was conducted to assess the desirability of using a common SUMC processor design for both BOSS and CPE functions. This looked attractive because BOSS will execute routines for data scheduling, system test, repair, and configuration, and interrupt processing. For four simultaneous processing streams executing programs of an average of 5 msec. duration BOSS will execute at least 800 routines per second. To meet these function and speed requirements, BOSS will have to be a small special purpose computer including such instructions as LOAD, STORE, NO OP, JUMP, TEST, SPCJ, AND, OR, SHIFT, ADD, SUB, plus macro instructions to speed up frequently used processes such as table searches.

Advantages to having a common CPE and BOSS design would include the fact that if processor spares can accomplish either function fewer total modules would be required, potentially increasing reliability and lowering system cost. Maximum logic commonality should cut costs since only one module would need to be designed and tested and commonality would make small BOSS-less systems more feasible. Figure 2 is a very much simplified block diagram showing a functional partitioning of SUMC into BOSS only, CPE only, and common functions. BOSS will look functionally similar to the SUMC-CPE - however SUMC instructions such as MULT, DIVIDE, Square Root, floating point and double precision will not be needed and special system monitoring and control logic will be required in BOSS but not SUMC. This latter difference leads to a far greater interconnection problem for BOSS than for a CPE but aside from the interface most BOSS only logic would be in the form of a different MROM program for BOSS than for the CPE.

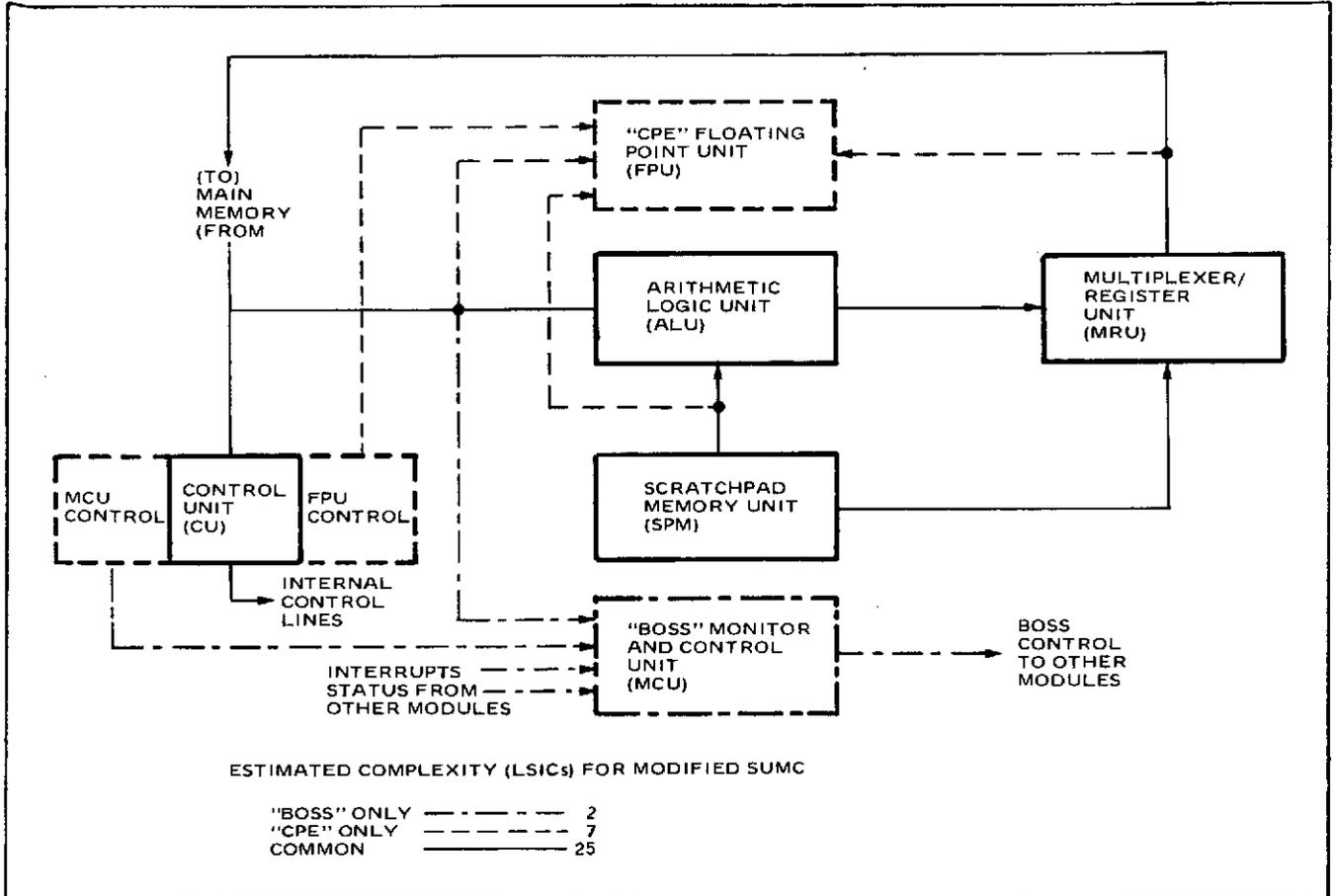


Figure 2. SUMC Module Partitioning

We have come to the conclusion that despite their similarity, BOSS and CPE modules should not be made identical because of the wasted non-common logic involved (20-25%), the increased intermodule switch complexity if any module is allowed to assume either BOSS or CPE status, and the physical problems of inter-connecting status and control lines between all CPE/BOSS modules in a compact structure. Further we feel that BOSS should physically be one module with several (probably 4) identically partitioned parts, any combination of which can be operated in TMR or in duplex in the event of failure of all but 2 of the parts. This will allow maximum packaging efficiency on the assumption that each BOSS partition will contain 25-30 LSICs and BOSS overall may have 200 or more interconnects to other ARMMS modules. We do strongly recommend that an effort be made to maximize logic commonality between BOSS and CPE LSICs to minimize system development costs. As the BOSS and CPE designs become better defined in Phase III, this commonality will be assessed further.

Making SUMC Self-Testing

Finally, studies of adding a self-test capability to SUMC have been conducted to enhance its reliability over a five year mission. Hamming-Parity error detecting and correcting codes could be used to test the address and data registers and memories. Selective duplication within a processor and/or special test routines to be executed periodically would be needed to test the processor's control and arithmetic logic.

Coverage is defined as the conditional probability that all faults will be detected or masked before they result in erroneous computation. Designers of successful self-testing computers to date such as Jet Propulsion Laboratories' STAR and Bell Telephone Laboratories' ESS feel that error detection coverage approaching 100% is feasible for much less than a 100% increase in logic with these techniques. In SUMC the added complexity should not exceed 20%. Assuming 100% coverage this allows TMR-like operating capabilities (i. e., correcting errors without interrupting the program flow) with a duplex system since, when the voters indicate a disagreement, the output of the unit whose fault detection circuits sense the faults will be ignored. In many cases this allows uninterrupted program flow even if TMR capability has been lost due to a failure. It can also allow two duplex streams to be made of four processor modules giving TMR-like reliability at twice the system operating speed. A simplex system can detect errors and retry instructions, and generally perform like a duplex system without self-test would. Finally, self-test capability is essential to a BOSS processor and if SUMC can be made self-testing commonality between CPE and BOSS logic is increased. Therefore, despite the increase noted in SUMC logic and the fact that self-testing systems cost more to design and test it is felt that the increase in system performance and reliability afforded by a self-testing SUMC processor outweighs these disadvantages and should be pursued along the lines discussed above and later in this report.

Summary

In summary, it should be possible to build a modified SUMC processor using CMOS or Schottky TTL logic that will execute 425,000 instructions/second in a BOSS configuration with high reliability over a five year mission. In ARMMS system speeds exceeding one million instructions/second should be obtainable in configurations requiring from 100 to 300 watts of power. Some of the areas of major impact to the SUMC design are summarized in Table II and an analysis of SUMC instructions including overlap and CPE/BOSS commonality considerations is shown in Table III. Areas not covered in this section will be discussed later in this report.

TABLE II. AREAS OF MAJOR IMPACT ON SUMC DESIGN

● Voter Placement	● Interrupt Handling
● Replicated Busses	● Software
● Data Path Width	● Instruction Set
● BOSS Interface (Control Commands and Status Monitoring)	● Error Correcting Codes
● System Speed	● Power Switch
● Internal Fault Detection	● Processor Synchronization in TMR Mode
● Memory Protect	● Reconfiguration Features

TABLE III. SUMC INSTRUCTION ANALYSIS

Instruction	Avg Cycles Execution	Avg Cycles Mem Access	Total No Overlap	Total with Overlap	Improvement with Overlap	No. of Stored Microinstr.	Useful in Boss
LA	8	16	24	12	12	1	Yes
LB	8	16	24	12	12	1	Yes
LX	8	16	24	12	12	1	Yes
STA	8	19	27	15	12	1	Yes
STB	8	19	27	15	12	1	Yes
STX	8	19	27	15	12	1	Yes
DELY	6	8	14	8	6	1	Yes
NOP	6	8	14	8	6	1	Yes
HLT	6	8	14	8	6	1	Yes
JMP	8	8	16	14	2	1	Yes
TE	8	8	16	13	3	3	Yes
TG	10	8	18	15	3	6	Yes
JZ	10	8	18	15	3	4	Yes
JP	9	8	17	14	3	2	Yes
JXNZ	9	8	17	14	3	4	Yes
AXI	6	8	14	8	6	2	Yes
TINS	8	8	16	13	3	4	?
JMPI	6	8	14	14	0	1	?
SPCJ	10	19	29	21	8	2	Yes
RAR	16	8	24	18	6	5	?
RAL	16	8	24	18	6	5	Yes
SRAD	28	8	36	30	6	6	No
SLAD	28	8	36	30	6	6	No
SRL	18	8	26	20	6	6	Yes
EOR	10	16	26	14	12	2	Yes
OR	10	16	26	14	12	2	Yes
AN	10	16	26	14	12	2	Yes
EORR	10	8	18	12	6	3	?
ORR	10	8	18	12	6	4	?
ANR	10	8	18	12	6	4	?
A	8	16	24	12	12	2	Yes
S	8	16	24	12	12	2	Yes
AR	10	8	18	12	6	4	?
SR	10	8	18	12	6	4	?
M	46(30)	16	62(46)	50(34)	12	16	No
D	92(60)	16	108(76)	96(60)	12	19	No
SQR	72(40)	8	80(48)	74(42)	6	7	No
DA	14	24	38	20	18	6	No
DS	14	24	38	20	18	6	No
DSQR	78(46)	8	86(54)	80(48)	6	8	No
FA	20	16	36	24	12	19	No
FS	16	16	32	20	12	11	No
FM	52(36)	16	68(52)	56(40)	12	14	No
FD	96(60)	16	108(76)	96(64)	12	19	No
Total	19.2(15.6)	12.3	31.5(27.9)	23.2(19.7)	8.2	239	44
BOSS	9.9	12.5	22.4	14.6	7.8	79	25

Notes: Numbers in () assume two adders, otherwise one assumed.

Fetch and IO microprograms of 22 microinstructions are included in totals.

II. ARMMS Hardware Configuration Studies

One of the major efforts under ARMMS Phase II has been to study alternative configurations for interconnecting various ARMMS modules. These configurations were developed chronologically as A, B and C. Configuration A was characterized by busses between all major module classes with voting on the bus outputs in the TMR mode being done internally to the modules. Configuration B was similar to Configuration A except that the voters were considered as a separate module class rather than as a part of other modules. Reliability modeling led to the conclusion that the reliability gain due to this additional partitioning was marginal in comparison to the added interface wiring and software configuration complexity caused by the additional voter module class since voter switches are very simple compared to processors or memories. Hence, Configuration B was considered to be less desirable than Configuration A. Configuration C is a refinement of Configuration A that completely internalizes all voter switches, and functionally combines main and BOSS memory and several bus classes further reducing software configurational complexity and increasing reliability through more efficient utilization of memory and bus resources. Configuration C is now the ARMMS baseline. The 3 configurations are shown in Figures 3, 4, and 5 and described below.

Configuration A

The major characteristics of Configuration A are the following:

- a) Bus System. A system of 4 busses interconnects each set of modules. Switches are decentralized and under BOSS control. Busses consist only of wires interconnecting appropriate module classes. Implicit in each interface is some word assembly/disassembly hardware.
- b) A small high-speed local store is contained within each processor. This store retains the previous eight instructions for efficient looping and possibly one instruction ahead for overlapped instruction fetch. When memory access is initiated, the address is first compared to data in the faster local store.
- c) Error Detection and Correction – All processors contain a single error correcting, double error detecting network at the memory interface. Processor to I/O data is not coded since critical output data will utilize a TMR or duplex configuration.
- d) Memory Protection – In addition to the protection afforded by voting and error correction, protection against unauthorized or untimely access is provided in configuration A by a system of locks and priorities contained in each memory module's address decoding network.
- e) Configuration Control – Configuration control is a BOSS function. BOSS has the capability to send power switch setting, voter and bus switch setting(s), status commands, and status requests. Each module has a Module Status word which can be transmitted to BOSS on request.

2-10

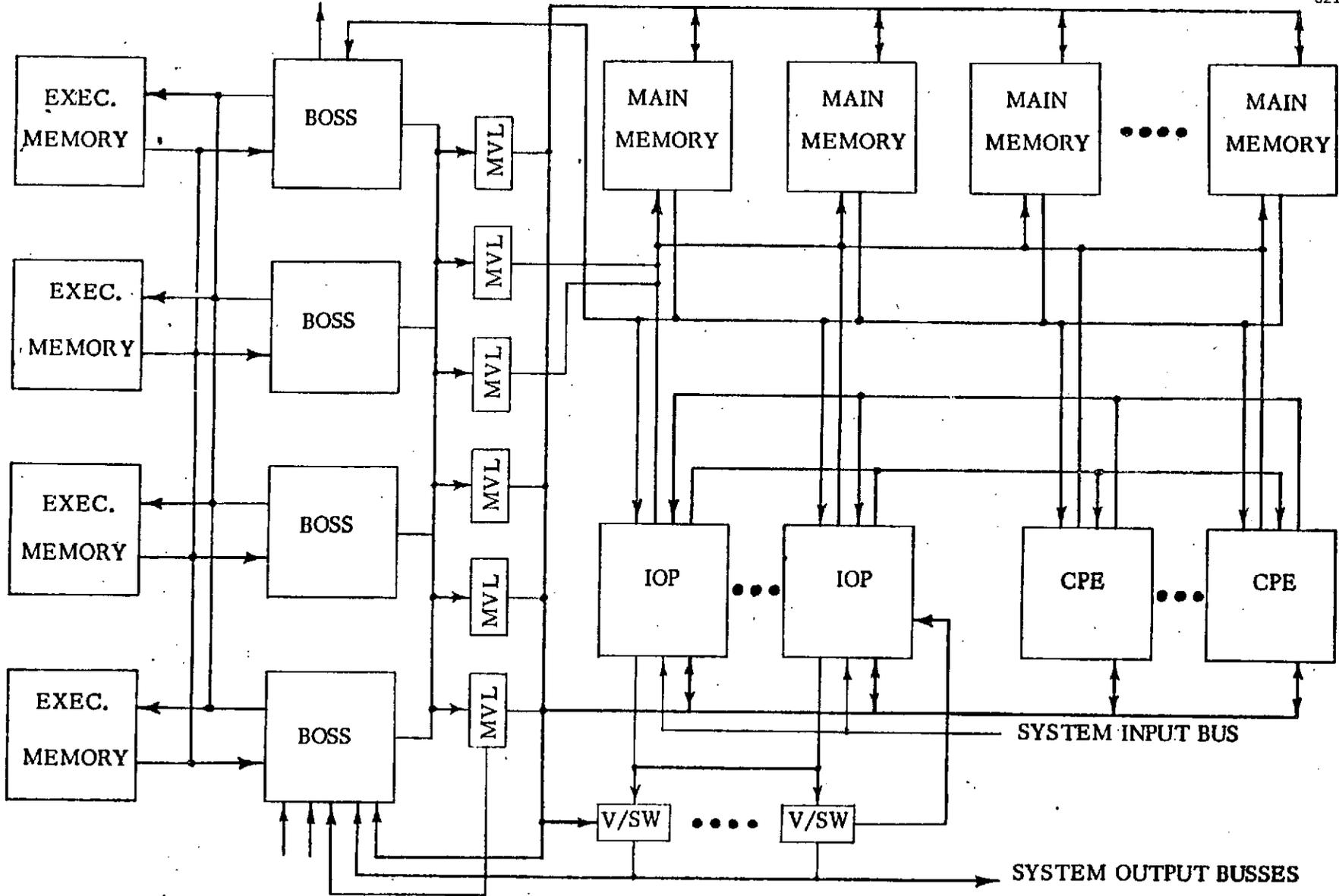


Figure 3. ARMMS System Configuration "A"

2-11

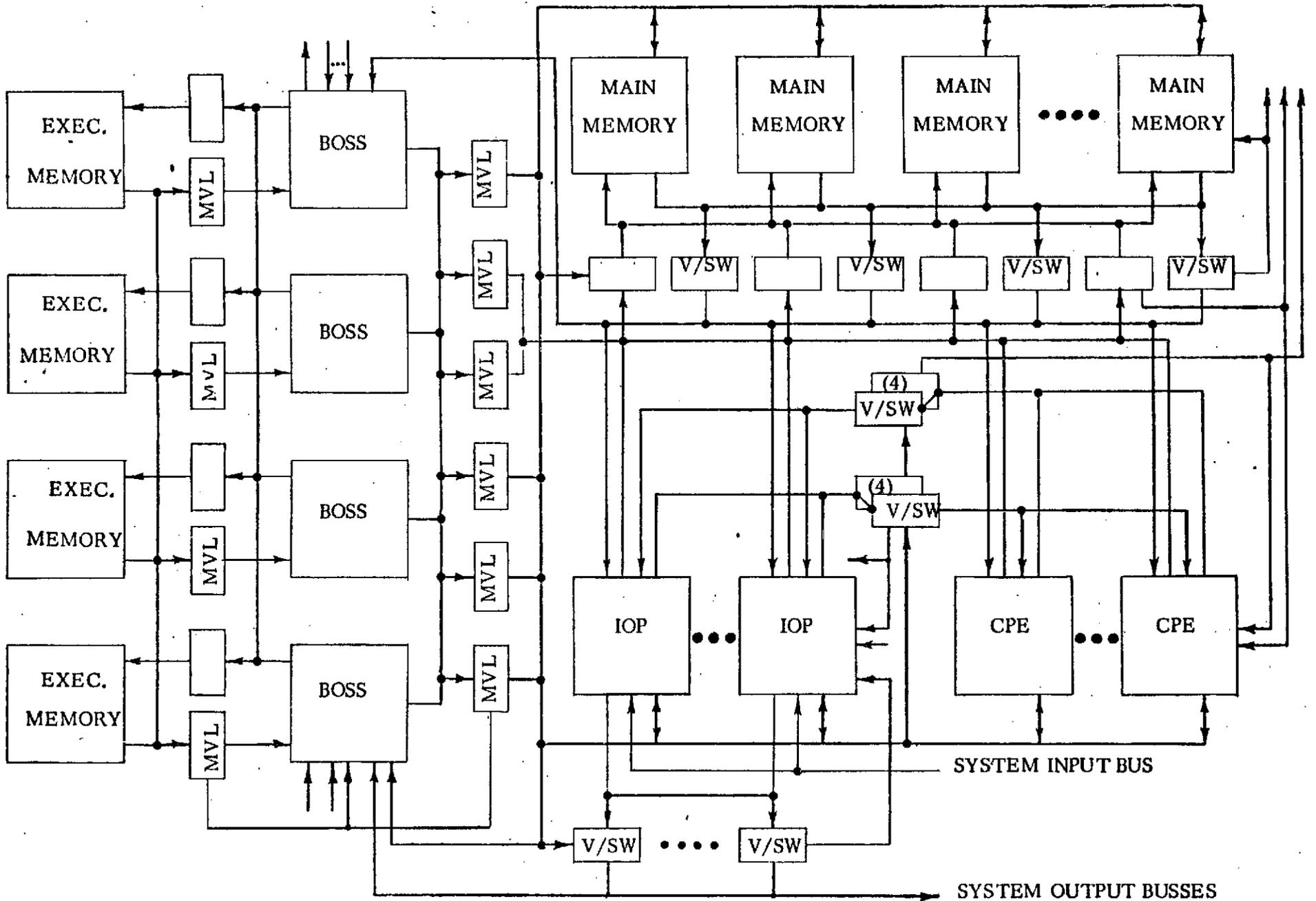


Figure 4. ARMMS System Configuration "B"

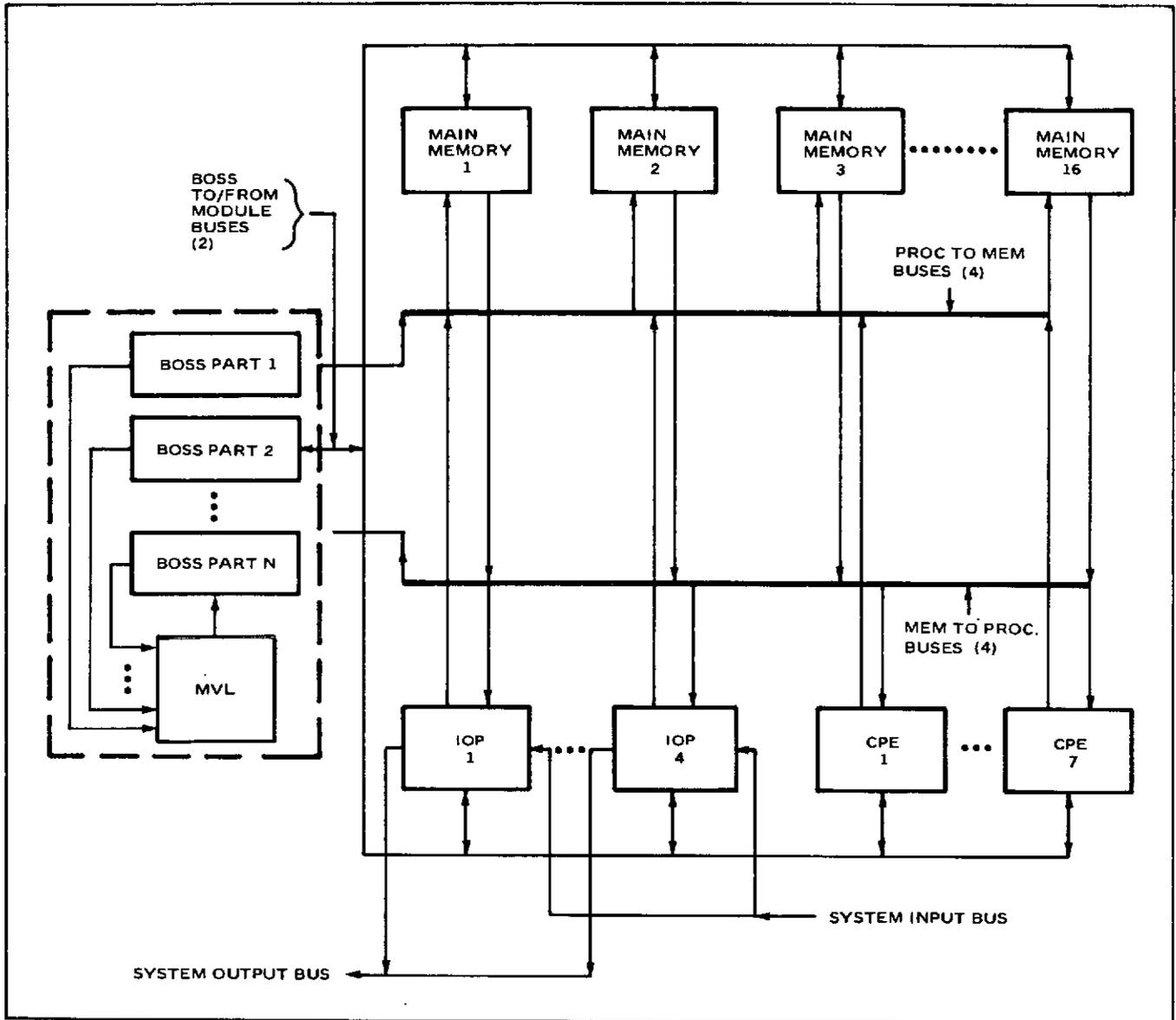


Figure 5. ARMMS System Configuration "C"

- f) **Voter Placement.** Voters connected to BOSS and external I/O are critical since there is no serial checking of their outputs. Consequently, these voters are located external to the modules and their outputs are compared in BOSS. Signals which are internal to the system can be voted on at least twice so that voter failures can be masked. Internal voters have been placed in the input interface of each module.

Configuration B

The major distinction between Configuration A and B is that voters in Configuration B are individual units inserted in the bus system between appropriate modules. Each voter receives inputs from all four busses and is set up by BOSS to vote on any three, to duplex any two, or to simplex the specific bus to which it is assigned. The output of each voter is then bussed to each destination module which has been set up by BOSS to receive on a particular bus. The major tradeoff aspects of this approach are that at the total amount of voting hardware in the system is reduced at the expense of added module types, interconnections, and busses, additional software and reduced bus speed. Figure 6 compares the 2 voting methods. The reliability study leading to the decision to reject Configuration B is described in the reliability section of this report.

Configuration C Design Philosophy

One of the toughest challenges ARMMS faces is reliable rapid reconfiguration at a reasonable cost in power, volume, and complexity. A prime consideration of Configuration C is minimization of the number of module classes and the number of system level interconnections between modules without sacrificing reliability or performance. To this end many of Configuration A's busses, and ports, and all of its external voter modules have been eliminated and their functions are now performed by the remaining modules and/or busses. Four module classes and three internal bus classes remain:

BOSS – This single, subpartitioned module will execute routines for data and I/O scheduling, interrupt processing, system test, repair, and configuration, and power and clock switching and distribution. BOSS will be an internally redundant self testing and repairing special purpose computer including such instructions as LOAD, STORE, NO OP, JUMP, TEST, SPCJ, AND, OR, SHIFT, ADD, SUB, plus macro instructions to speed up frequently used processes such as table searches and special control instructions used for monitoring and controlling other ARMMS modules. BOSS will consist of four or five identical subpartitions "B" containing power supply, timing oscillator, memory bus interface and control bus voting components.

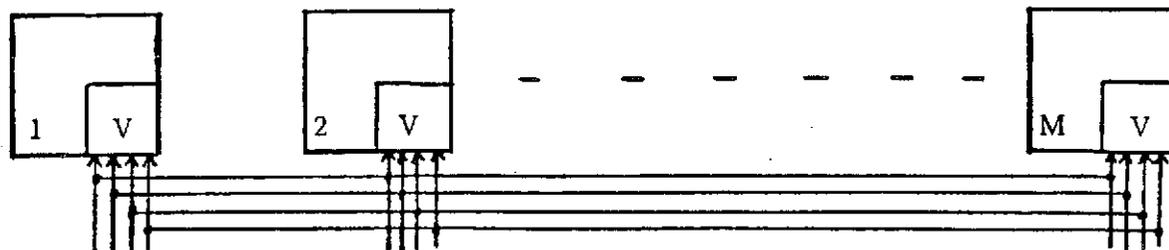


Figure 6A. A Module Class with Internal Voters

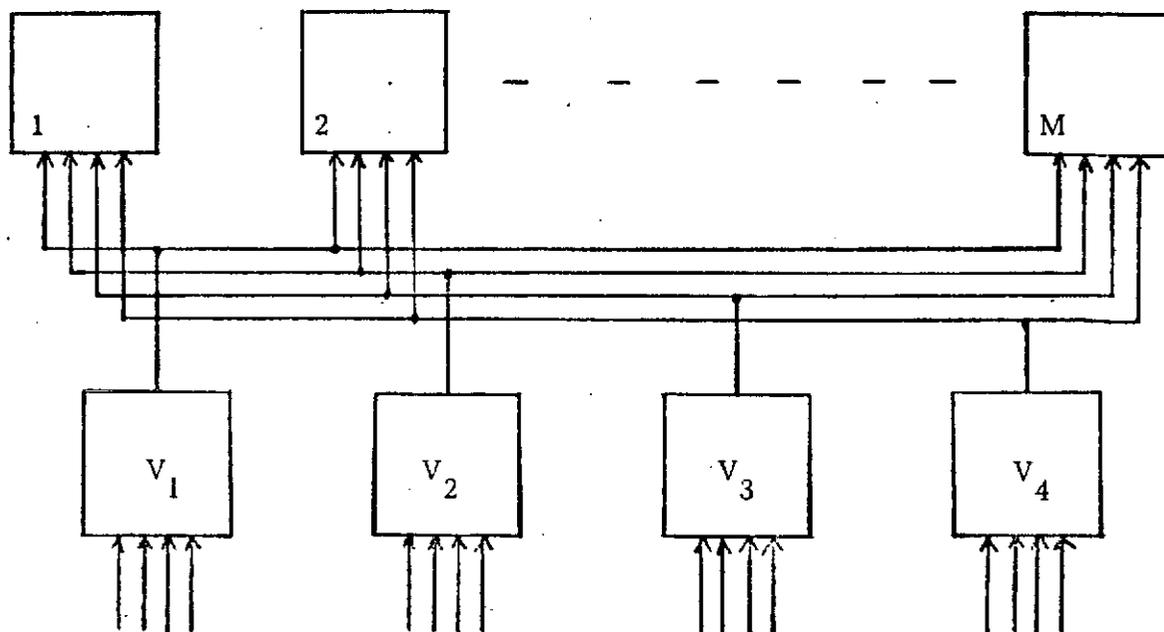


Figure 6B. A Module Class with 4 External Voters

IOP – ARMMS can accommodate up to 4 I/O processors. Each I/O processor contains standard logic matching it to ARMMS system interfaces. Internally the processors can be mission dependent containing either general or special purpose logic. Identified IOP functions include paging between bulk and main memory modules, spacecraft status monitoring and preprocessing, and spacecraft control. IOPs can be used singly, in pairs, or in triads, or can be internally redundant with multiple bus outputs.

CPE – ARMMS can accommodate up to 7 CPEs (Central Processing Elements). Up to 4 CPEs can be on line simultaneously with up to 4 IOPs and BOSS. CPEs can be utilized singly, in pairs, or in triads depending upon mission requirements. It is envisioned that the CPE will be an outgrowth of the SUMC processor modified to include self test logic. BOSS monitor and control interfaces and overlapped memory accessing.

MM – ARMMS can accommodate up to 16 main memory pages corresponding to 16 active memory modules in simplex configurations or larger numbers in dual or triad configurations. The total number of modules would be limited by bus driving components and might nominally be 25. The nominal module size is 8,192 words each containing 32 bits of data plus a 7 bit single error correcting, double error detecting code for data.

PMB – ARMMS contains 4 processor to memory busses. Each CPE is connected to 2 of these busses. IOPs are also nominally connected to 2 PMBs but can be connected to any number depending upon their design. BOSS and all memories will be attached to each of these 4 busses. Each bus contains 13 data lines, including error coding, and an Access request line. Software will keep track of the 2 non-existent bus ports on each processor in the same way as it does failed bus ports.

MPB – ARMMS contains 4 memory to processor busses each of which is connected to every processor module in order to allow TMR voting between any triad of busses and unlimited choice of processors with which to make up the triad. Each bus contains 13 data lines, and a response line.

BMB – Finally ARMMS contains 2 (one plus a spare) BOSS to/from module busses on which BOSS sends control codes to processors and memories and receives status information upon request. All commands and responses are coded and commands are address-tagged on this bus. The bus will nominally consist of 8 data lines plus dedicated parity, clock, and sync lines. BOSS may command or interrogate other modules at will or in response to individual interrupts from them.

Memory Usage

Configuration C memory usage differs in several respects from Configuration A:

- 1) Instead of a local store, BOSS utilizes temporarily dedicated portions of main memory for most storage requirements. This eliminates the need for separate busses between BOSS and dedicated memory and allows spare memory modules to be dedicated either to BOSS or to processing as needed, reducing the total number of modules required.
- 2) Instruction overlap speeds ARMMS processors up sufficiently to negate any advantages of a high speed local store hence no words are retained automatically in the processor after their use except in processor registers.
- 3) Base and bound registers are assumed for memory protection rather than Configuration A's more complex lock and key scheme.
- 4) Rather than providing separate busses between IOPs and CPEs configuration C provides for CPE to IOP transfers via a processor to memory and memory to processor bus pair using a memory as a switching point for a special transfer command. A need for a direct IOP to CPE path has not been established and has not been provided for although such a path could be handled similarly to the CPE to IOP transfer – again using a memory as a switching point.

Startup Procedure

In Configuration C the system start up procedure is as follows: Initially all BOSS partitions and all memories are powered. Memories all initially answer to a Page 0000 address. The BOSS partitions vote to choose the three lowest numbered partitions "A" capable of agreeing with one another and the lowest numbered partitions "B" self-checking OK. Other BOSS partitions are then voted off. The BOSS module then accesses memory location 0...0 which contains previously stored data as to which BOSS partitions were in use previous to power down and which memory modules contain BOSS memory. These partitions and memories are powered and the other memories powered down. Following successful completion of BOSS diagnostics with this memory the configuration routine is called to begin powering up and testing other memories and processor modules as required, and provide them with program status words (PSW) on the BMB lines to allow resumption of normal operation. It is expected that BOSS configuration control, job and I/O scheduling and interrupt processing would be performed as in Configuration A.

Error Detection

Configuration C error detection differs from Configuration A, in that:

- 1) Instead of a negative acknowledge scheme, all modules will communicate their error detection findings directly to BOSS where BOSS will make judgements as to which modules are in error by analyzing the data and running diagnostic routines on the modules in question as needed.
- 2) A 6-bit Hamming code plus an overall parity bit will be appended to stored memory data to allow correction of one error occurring in memory. This equivalent to providing a spare memory bit plane in that 2 bit planes must fail before a memory module will be considered failed but it avoids the problem of actually switching in a spare plane and loading it with previously stored data. This combination of codes will also detect up to 4 errors allowing for detection of a failed bus line for example. A similar code will be used to detect improper memory address accessing. Code checks will be performed with every inter-module data transfer. Decoding occurs prior to the ALU, with decoding following the ALU. The logic in between is duplicated and outputs are compared since the Hamming-Parity code is destroyed by arithmetic operations. This is discussed further in the section on Error Detection Strategy.
- 3) Voting on BOSS commands to other modules is done internally to BOSS (in subpartition B) using duplicated voters and busses and internal subpartition switching in the event of a voter disagreement. This eliminates the need for the external BOSS to voter busses and command voters of Configuration A. If a need for redundant output busses from I/O is established a similar scheme can be employed. I/O processors can be made internally redundant as needed and the I/O to voter busses and external output voters eliminated.

Modules will first try to detect and correct errors by masking in the TMR or duplex mode or rollback and retry methods in simplex mode or in duplex mode cases where masking cannot be achieved. In both cases errors will be tallied. If the modules are not successful in correcting the error BOSS will be interrupted and will obtain status information from the modules in question via the BMB lines. BOSS will determine which module has failed through diagnostic routines, place it at the bottom of that module classes' spare queue and try other modules until a good one (hopefully) is found, place the good module on line, and resume computation. In the TMR mode the task will continue to completion at top priority and then the diagnostic procedure will be applied. ARMMS will be considered to have failed if and only if BOSS cannot find a usable module in each class by this procedure or

if an erroneous computation goes undetected. A module is not considered to have failed until the failure manifests itself. Using internal error detection within modules allows masking of errors in duplex mode and detecting them in simplex so as error detection coverage approaches unity duplex operation looks like TMR and simplex looks like duplex. In many cases this could allow higher throughput and longer system life due to using fewer modules per stream.

Intermodule Interface Approach

An intermodule interface has been designed that allows any CPE, IOP, or BOSS module to address any non-protected memory page. It allows any combination of simplex, duplex, or TMR streams with any combination of relative priorities to co-exist with minimum bus contention providing that no more than 4 CPEs, 4 IOPs, and BOSS are involved simultaneously. Volatile storage defining a module's role in ARMMS has been minimized and coded such that transients cannot cause an undetected change in the module's status. The interface allows all modules of a class (CPE, Memory, etc.) to be virtually identical. Interface gate complexity and module to module interconnections have been minimized. Whenever a stream is formed BOSS sends each processor module involved a stream status code on the BMB lines defining all bus connections within the stream. Once assigned to a stream a processor always uses the pair of busses specified by the stream status code for communication to and from memory eliminating bus contention among processors of a given type. For redundancy each processor can output on a choice of two busses. This choice is made by BOSS command. To reduce bus contention between processors of different types a hierarchy is established such that I/O and BOSS modules can inhibit CPE modules from starting a new memory access cycle when the former modules require access to a memory bus. Similarly BOSS (but not CPE) modules can inhibit I/O modules' bus access. Once any module has been granted access it will continue to have it until transfer of the word involved has been completed. Usually only processors using busses needed by other processors are inhibited except that all processors operating synchronously in a duplex or TMR stream are inhibited if one or more processors in the stream are inhibited insuring maintenance of synchronization between these processors. Modeling indicates that speed lost due to bus contention between processors of different types should be less than 3% exclusive of memory contention losses that are independent of the interface design.

BOSS assigns each memory module a page address and a high, middle, or low bus response assignment in the case of memory accessible by a TMR stream (or a high or low assignment for access by a duplex stream). If further studies result in elimination of the TMR memory mode, TMR streams will be able to access duplex memories. Presently a memory module can be accessed only by stream of equal or lower critically than its mode. Access by a stream of higher critically results in an interrupt to BOSS to initiate a paging routine to increase that page's criticality. Memory page size will equal memory module

size. All memory modules assigned to a given page output on the same bus when accessed by a simplex stream or on different busses according to their bus response assignment when accessed by duplex or TMR streams. Examples are shown in Figures 7 and 8. All duplex or TMR stream processors receive memory outputs on all busses assigned to that stream. Each processor access request contains a page address and a bus priority code. Processors will continue to request access until it is granted or until they are temporarily inhibited by other processor's desire to access.

The assignment codes discussed above require 6 bits from BOSS to memories, and 5 bits from BOSS to processors plus extra bits for error detection coding. Each module input interface includes voting and fault detection coding logic. These interfaces can be implemented at an estimated cost of 4 to 5 LSICS/module (~250 gates each) - this represents approximately a 10 to 15% increase in ARMMS overall logic.

The ARMMS priority structure will involve both hardware and software elements. The hardware recognizes a minimum of 16 different priority levels. The software then selects different subsets of these 16 as program requirements dictate. The highest hardware priority goes to BOSS since the efficiency of the rest of the system depends on BOSS completing its tasks efficiently. The second highest priority is a special TMR CPE mode used only in the event of an error in one of three TMR channels to insure completion of the TMR task with maximum speed prior to initiating diagnostic tests on the stream. The next seven priorities are for I/O streams on the assumption that the timing of external events happening and mass data transfers is more difficult to control than the timing within processing streams and hence IOP memory access requests should be given higher priorities than CPE access requests. The seven lowest priorities are for CPEs.

So long as BOSS, I/O, and CPE programs are mostly segregated into different memory pages all 3 types of programs should be able to be executed simultaneously with minimal bus or memory contention. When these programs wish to access the same memory page the internal logic design of the memory access logic will tend toward letting the streams access the memory a word at a time in turn since each processor will release the memory temporarily between access requests letting the next higher priority stream gain access for one word. This results in all contending streams slowing down but none stopping entirely. Obviously this does not preclude the need for designing the software to minimize memory contention if ARMMS is to perform efficiently as a multiprocessor.

The seven priority levels available for normal I/O and CPE scheduling are ordered in descending priority as shown in Table IV allowing the 14 modes listed in the table. The logic allows any of the combinations listed for CPEs to be used simultaneously with any of the combinations listed for IOPs. Note that

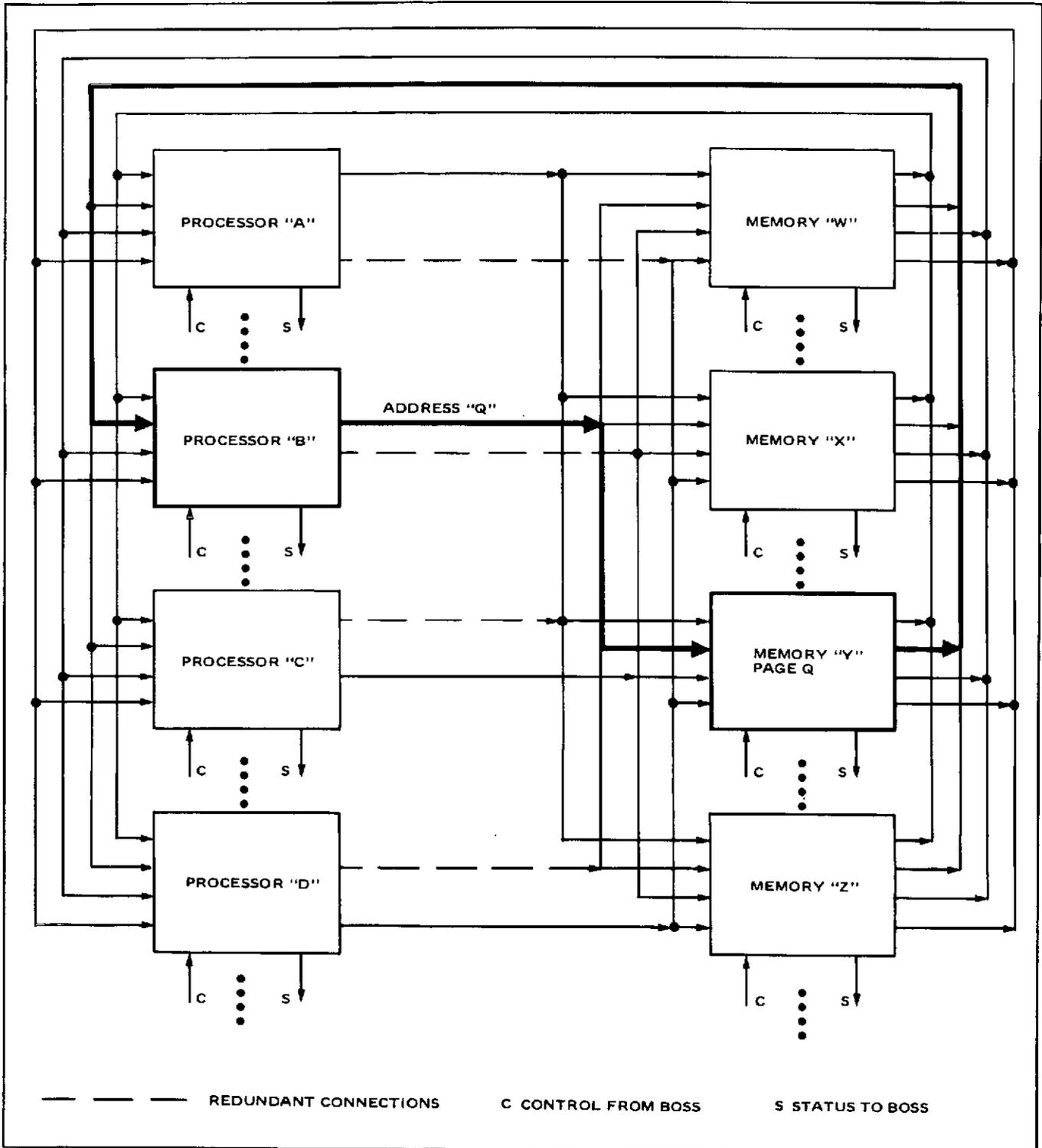


Figure 7. ARMMS Processor/Memory Interconnections – I. Processor B Access to Memory Y In Simplex

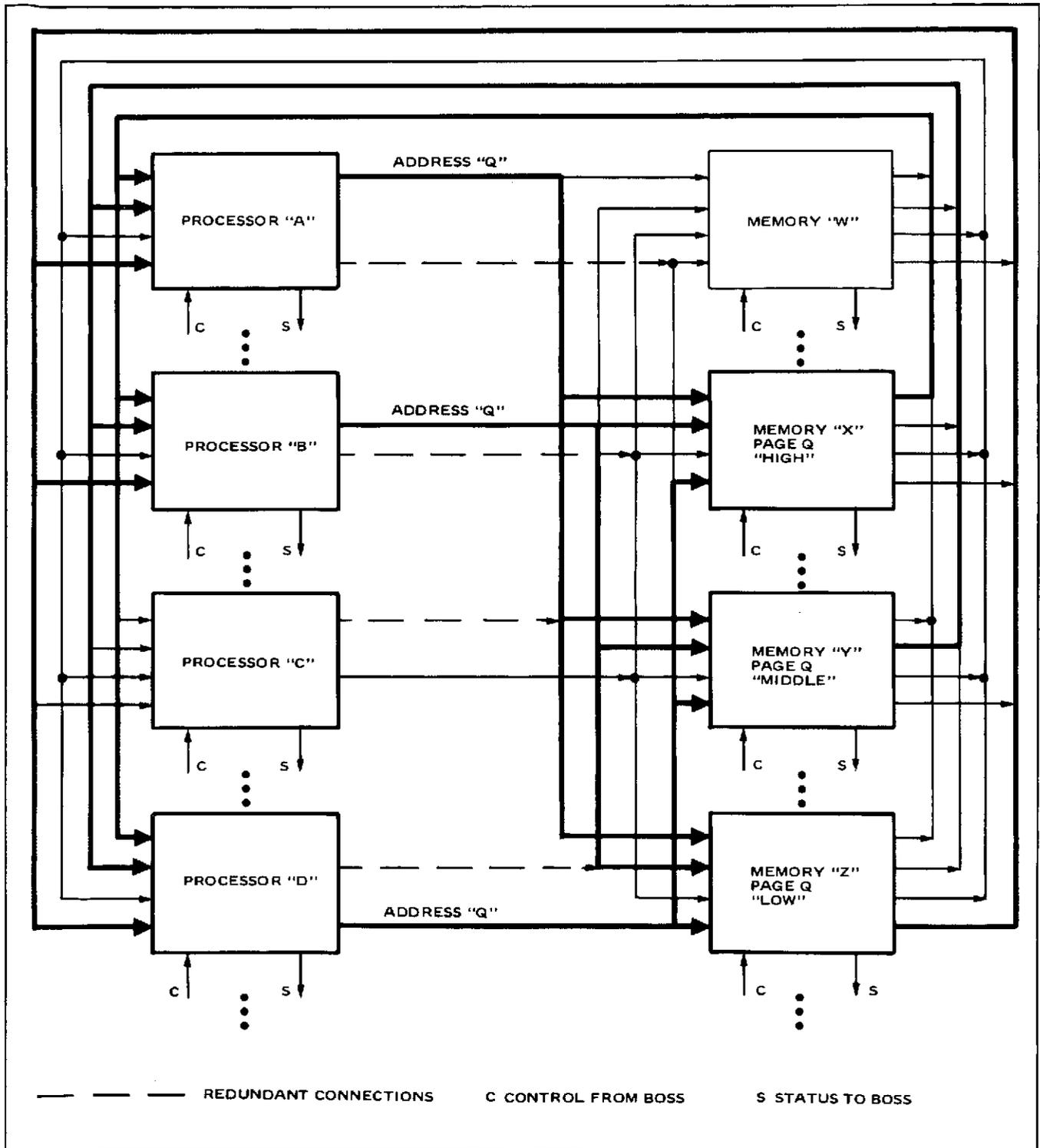


Figure 8. ARMMS Processor/Memory Interconnections – II. Processors A, B, D Access To Memories X, Y, Z X, Y, Z In TMR

TABLE IV. ARMMS PROCESSOR PRIORITY ASSIGNMENTS

	Priority Code	Proc. Type	Stream Criticality
1. (Highest)	0000	BOSS	TMR
2.	0001	CPE	TMR (Special)
3.	0010	IO	SIMPLEX A (SA)
4.	0100	IO	DUPLEX A (DA)
5.	0110	IO	TMR (TR)
6.	1000	IO	SIMPLEX B (SB)
7.	1010	IO	DUPLEX B (DB)
8.	1100	IO	SIMPLEX C (SC)
9.	1110	IO	SIMPLEX D (SD)
10.	0011	CPE	SIMPLEX A (SA)
11.	0101	CPE	DUPLEX B (DB)
12.	0111	CPE	TMR (Normal) (TR)
13.	1001	CPE	SIMPLEX B (SB)
14.	1011	CPE	DUPLEX B (DB)
15.	1101	CPE	SIMPLEX C (SC)
16. (Lowest)	1111	CPE	SIMPLEX D (SD)

NOTE: IN A FULL PROCESSING STREAM AN IOP MAY BE GIVEN THE STREAM'S CPE PRIORITY CODE.

IOP AND CPE STREAMS MAY INDEPENDENTLY HAVE THESE 14 MODES:

<u>4 Processors</u>	<u>3 Processors</u>	<u>2 Processors</u>
(SA, TR) or (TR, SB)	(TR)	(DA)
(DA, DB)	(SA, DA) or (DA, SB)	(SA, SB)
(SA, SB, DB) or	(SA, ..., SC)	
(SA, DA, SB) or		1 Processor
(DA, SB, SC)		
(SA, ..., SD)		(SA)

the choices allow for any combination of relative priorities between streams of differing criticality and that the software system can change the priority assignment of a given stream at will. Also that combinations such as 2 duplex IO streams and a simplex plus a TMR limited processing stream are allowed. If IOPs and CPEs are to be tied together in the concept of a "full processing stream" via software both processor types could be given either the same CPE or the same IOP priority assignment by BOSS. Otherwise BOSS assigns IOPs only I/O priority codes and CPEs only CPE priority codes and the hardware provides for complete independence of the I/O and limited processing streams subject only to software restrictions.

Signals Across Intermodule Interfaces

In order to access data from memory a processor must provide a 4-bit page address to select one of 16 memory pages, a 4-bit priority request to allow the given memory page to choose the highest priority stream's request and determine if the correct number of processors agreed on this request, the number being determined by the priorities mode (simplex, duplex, or TMR), a 3-bit 2 out of 3 coded Read/Write/Transfer request and a 13 bit word address to select one of up to 8,192 words in a memory module. The first 8 of these 24 bits must be present for a memory to make a decision as to whether or not to grant the request. In addition a sync or "access request" signal must be present to tell the memory that it is supposed to be making such a determination if these 8 bits are to be transmitted on lines that can also carry word addresses and data that might otherwise be confused with page and priority information. The processor to memory bus must be at least 8 bits wide plus the access request line and any desired parity lines in order to function efficiently.

One micro-instruction period is available in which to transfer an address between a processor and memory for the shortest instructions with maximum instruction overlap. If more time is used the transfer will slow down the processor. Similarly 1-1/2 micro-instruction times are available in which to transfer data from the memory back to the processor without incurring time penalties.

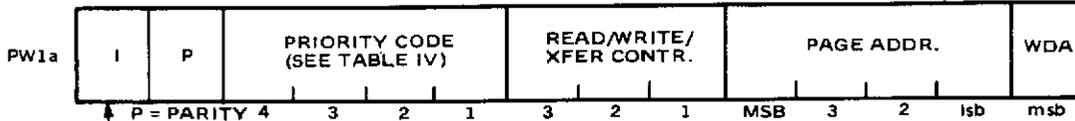
Assuming a 32-bit word plus 7 error correction code bits this totals 39 bits. Five clock times would be needed to transfer this data over an 8-bit bus, 4 over a 10-bit bus, or 3 over a 13-bit bus. The address would be transferred in 3 clock times over a 10-bit bus or in 2 clock times over a 13-bit bus. Taking into account the awkward timing and the marginal speed capability of running the transfers at 3 times the micro-program instruction rate with 10-bit busses 13-bit busses and a transfer to micro-instruction clock ratio of 2:1 were chosen both to and from memory. Wider busses would not provide any additional speed advantages.

In addition to data lines, the memory to processor bus must contain a dedicated memory response line to signal the processor that the first 13-bits of address have been accepted and the processor is to continue the transmission to completion. If a processor does not receive this response signal it will continue to transfer the first 13-bits of the address to the memory interface until either the processor is inhibited by another processor or the memory responds to the data. Since only one processor can use the bus at a given time all requests and responses are unambiguous. Unsolicited "Data Ready", "Memory Cycle Complete", and "Lock Control" lines between memory and processor are not required in Configuration C.

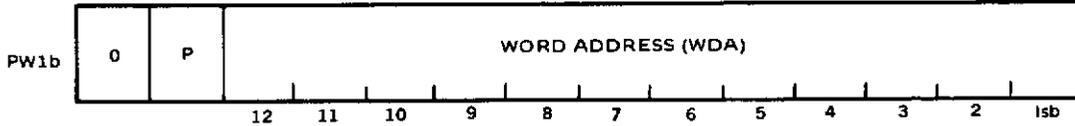
Three additional lines are required in connection with the memory busses at the processors only. Each processor receives inhibit lines from each of the other two classes of processors and sends an inhibit to these other two classes, describing each processor's bus activity. In addition an I/O busy line may be required from IOP to CPE in the event of several CPEs wishing to access a given IOP simultaneously. This will depend on the details of the IOPs and is shown for completeness. Note that the BOSS module receives the IOP's Memory Access Request as an inhibit rather than the IOP's normal inhibit line which does go to the CPE. This is because the IOP's memory access request line will not go true until all busses needed by the IOP have been cleared of traffic and hence this line will inhibit BOSS only in the event that the IOP can gain access to the memory through use of free busses or inhibiting CPEs, maintaining BOSS priority over the IOP. The information to be transferred to or from a memory by processors is summarized in Figure 9.

The BOSS to-or-from Module Bus must carry addresses capable of differentiating between all ARMMS modules for the purpose of sending commands: Each module can then contain a decoder that responds only to commands addressed to it. With up to 25 memory modules, 4 IOPs, and 7 CPEs this can be accomplished with an 8-bit (2-byte) 2 out of 4 code. Following the addressing, commands identified to date could easily be transmitted within 1 to 2 additional clock times. In addition to the data lines, clock, sync and possibly parity lines are required from BOSS to each module. In addition to the bus BOSS will have to provide for non-bussed power control lines to and interrupt lines from each module. For reliability's sake each of the lines between BOSS and the processors is made redundant and are compared within each module. The BMB is a two-way bus due to its assumed light usage as it was in Configuration A. All interconnections for each type of module in ARMMS are summarized in Tables V.a, V.b, and V.c. The BOSS has substantially more interface lines than do the other modules but since it is packaged in a larger package and internally subpartitioned rather than specified as a group of small modules they should present no mechanical packaging or reliability problems. I/O modules may prove to be the most pin limited depending on their size and the nature of their connections to external inputs and outputs. However, Hughes packaging technology experts indicate that connectors capable of meeting ARMMS demands are currently available.

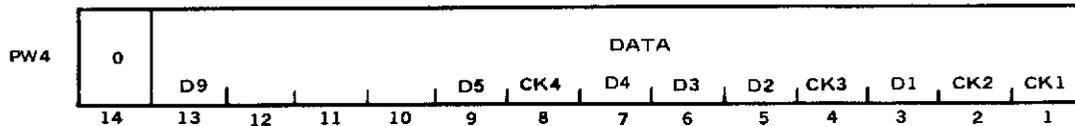
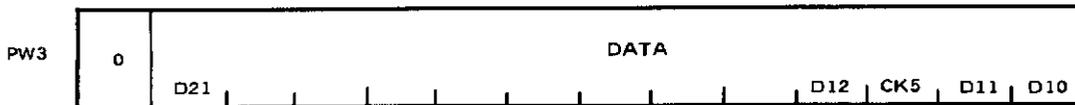
PROCESSOR TO MEMORY BUS.



ACCESS REQUEST (AR)
(DEDICATED LINE)



(WRITE OR XFER ONLY)



MEMORY TO PROCESSOR BUS.

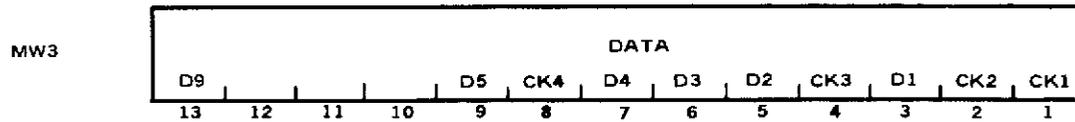
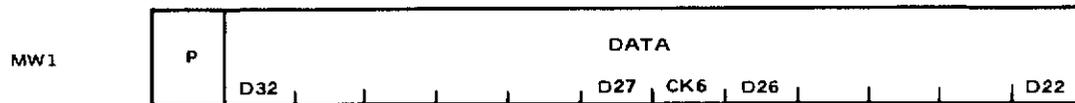


Figure 9. Configuration C Memory/Processor Word Formats

TABLE V.a. CPE INTERFACE LINES ESTIMATE

PROCESSOR TO MEMORY BUS

DATA/ADDR.	}	13
Incl. error det/corr. code		
ACCESS REQ.		1
INHIBIT TO OTHER PROC.		1
		<u>15 x 2 = 30</u>

MEMORY TO PROCESSOR BUS

DATA	}	13
Incl. error det/corr. code		
MEMORY RESP.		1
INHIBITS FROM OTHER PROC.		2
		<u>16 x 4 = 64</u>

BOSS TO/FROM MODULE BUS

COMMANDS/STATUS	8
PARITY	1
SYNC	1
CLOCK	1
	<u>11 x 2 = 22</u>

NON-BUSSED LINES

INTERRUPT TO BOSS (red.)	2
POWER CONTROL (redundant)	2
IO BUSY (optional)	4

POWER SUPPLY 2

TOTAL 30+64+22+10 = 126

IOPs will have same lines as CPE plus additional system input and output bus connections and discrete command lines. Assume 30 lines per bus plus 2x redundancy and 25 commands yields 219 IOP lines total. Note that processor xfer request and access request lines must go to IOPs to signal CPE IOP xfers. (8 lines).

TABLE V.b. MEMORY INTERFACE LINES ESTIMATE

PROCESSOR TO MEMORY BUS

DATA/ADDR	}	13
Incl. error det/corr. code		
ACCESS REQ.		1
		<u>14</u> x 4 = 56

MEMORY TO PROCESSOR BUS

DATA	}	13
Incl. error det/corr. code		
MEMORY RESPONSE		1
		<u>14</u> x 4 = 56

BOSS TO/FROM MODULE BUS

COMMANDS/STATUS	8
PARITY	1
SYNC	1
CLOCK	1
	<u>11</u> x 2 = 22

NON-BUSSED LINES

INTERRUPT TO BOSS (red.)	2
POWER CONTROL (red.)	2

POWER SUPPLY

TOTAL 56+56+22+6 = 140

TABLE V.c. BOSS INTERFACE LINES ESTIMATE

PROCESSOR TO MEMORY BUS

DATA/ADDR.	}	13
Incl. error det./corr code		
ACCESS REQ.		1
INHIBIT TO OTHER PROC.		1
		<u>15</u> x 4 = 60

MEMORY TO PROCESSOR BUS

DATA	}	13
Incl. error det./corr code		
MEMORY RESP.		1
INHIBITS FROM OTHER PROC.		2
		<u>16</u> x 4 = 64

BOSS TO/FROM MODULES BUS

COMMANDS/STATUS	8
PARITY	1
SYNC	1
CLOCK	1
	<u>11</u> x 2 = 22

NON-BUSSED LINES (assumes 36 modules controlled)

INTERRUPTS TO BOSS	72
POWER CONTROL	72
POWER SUPPLY	2
	<u>146</u>

TOTAL 60+64+22+146 = 292

Interface Timing

Analysis has shown that microprogram clock speeds on the order of 4 to 5 MHz will be the maximum to be realistically expected from a modified SUMC with SOS CMOS technology in the late 70's time frame. An optimistic baseline choice of 5 MHz has been made. Plated wire technology easily allows non-volatile storage with access times of 300 ns and cycle times of 600 ns for read and 800-900 ns for write accesses to memory at reasonable power figures. Semiconductor technology could also achieve these figures but was deemed undesirable for ARMMS due to its volatility. Bus speeds on the order of 15 MHz are obtainable but these numbers must be reduced to take interface logic into account. Bus speeds of 10 MHz (twice the processor speed) should provide a good match for the memory and processor speeds listed. These figures will be adopted as baseline numbers for Configuration C. Worst case numbers may be somewhat lower but should continue to track each other well.

Figure 10 shows a non-overlapped instruction. For n micro-instructions such an instruction would require $2n + 8$ 10-MHz clock cycles for all instructions except writes and interprocessor transfers, $2n + 11$ cycles for WRITES (assuming that the data must be present before the WRITE cycle is initiated), and 5 cycles for transfers. Figure 10 shows an overlapped instruction where a new instruction is fetched while the previous one is being executed. Instructions requiring a 2nd operand require two fetches. Basically overlapped fetches require 600 nsec each vs. 1200 nsec without overlap but JUMP instructions and WRITE instructions involving the previous instruction's operand cannot be overlapped and hence require an extra fetch limiting the average speed increase with overlap to 40% rather than 100% — still an impressive improvement. When more than 4 clock cycles are required by an overlapped fetch and execution (i.e., more than 2 micro-instructions involved), or a WRITE is involved, the next memory access will be delayed as shown by the cross-hatched areas in Figure 10 or Table VI or the processor busy test in Figure 11. Recall that the page address, priority, Read/Write/Transfer fields, and the most significant bit of the word address are transferred on the first address transfer clock-time and the remaining 12 word address bits on the second address transfer clock-time assuming that the first transfer was accepted. When a processor inhibit or lack of memory response occurs all operations except for completion of a previously started data transfer cycle cease until the conditions change. Hence any clock cycles lost in this way are directly additive to the execution time of the program in the absence of delays and the timing diagrams remain the same except for these delays.

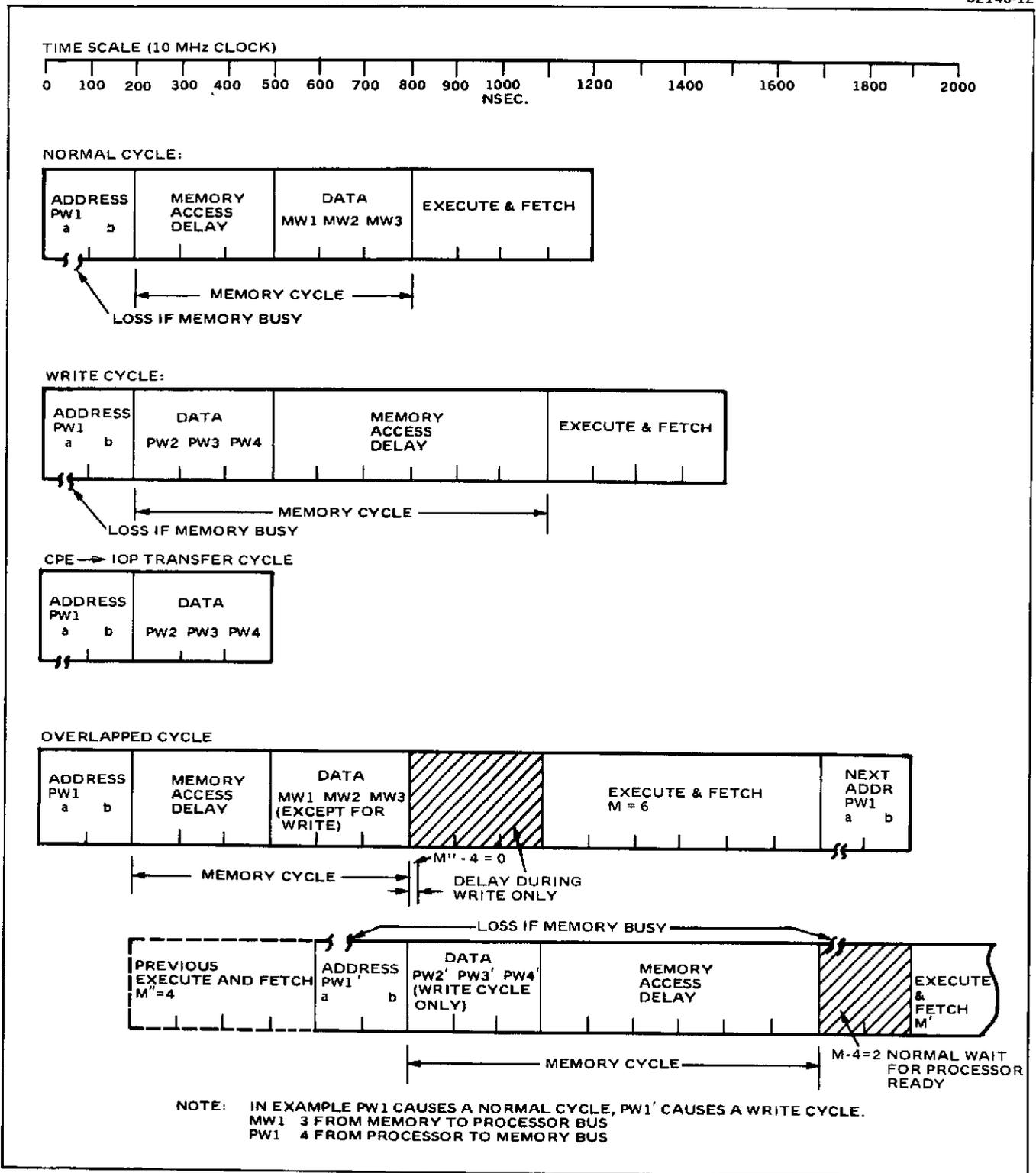


Figure 10. Configuration C Memory/Processor Transfer Timing

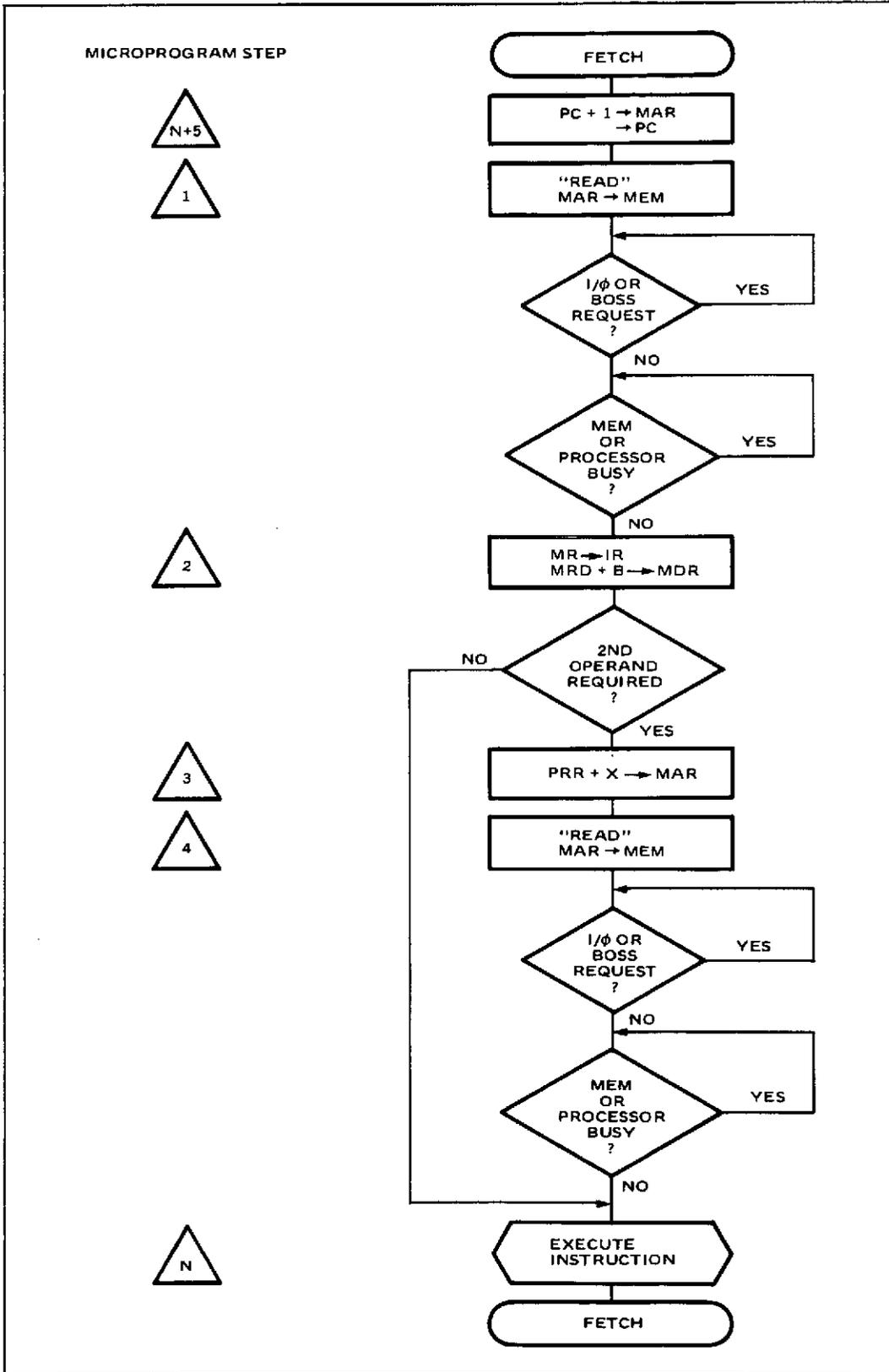


Figure 11. MOD-SUMC Fetch Cycle

As noted in the text and Figure 10 data transfer to or from the memory requires three 10 MHz clock cycles and a memory access time of 300 nsec requires 3 of these clock cycles. Execution time requires $2n$ clock cycles for n micro-instructions. The interested reader is referred to Table III of this section for examples of timing of SUMC micro-instructions using overlapped fetch cycles. Figure 11 shows the overlapped program's fetch and execution cycle. Table VI shows the execution of a sample program. Events in the left hand column occur simultaneously with those in the right hand column. Terminology is from SUMC documentation.

Interface Logic Details

The Configuration C drawing shows the interface logic to a register level, but a few words are in order as to finer details. Within each processor is an access request network that will request memory access whenever an appropriate bit appears in the processor's micro-program, subject to the inhibitions from other processors discussed earlier. The choice of inhibiting factors is controlled by the Stream Assignment Register. The logic also correlates memory responses to its access request and, when a response from the correct memory modules occurs, sets a flip-flop allowing the access to go to completion and inhibiting other access to the bus until the cycle is complete as signalled by a second micro-program bit within the processor. Figure 12 shows a gate level drawing for this logic in the case of a CPE module. Logic for IOP and BOSS is similar.

Figure 13 gives a detailed view of the logic within each memory module's access control block in the Configuration C diagram. As the data comes in on each bus, busses whose access request lines are true and have page addresses agreeing with a memory module's page address will be tested for access to the memory registers. The 16 priorities are decoded and applied to the request detection and priority ordering logic. If this circuit detects the correct number of requests of the highest priority present at the time of the test and the memory is not already in use the memory responds on the busses assigned to the processor generating the request and gates the response decision into the Response and Criticality fields of an Assignment Holding Register and to the voting logic to allow the voted data to go to the memory registers and to set up the proper output bus paths for the memories data output in the case of a Read. When the cycle is complete the Response and Criticality fields of the Assignment Holding Register are cleared and the memory is ready for the next access.

Each module also contains voting logic which will vote any combination of 3, compare any combination of 2, or transfer any one busses' inputs to an appropriate module register signaling any disagreements to the module's status/command network which will interrupt BOSS as appropriate. In simplex and duplex modes the voting logic correlates error detecting code outputs with its

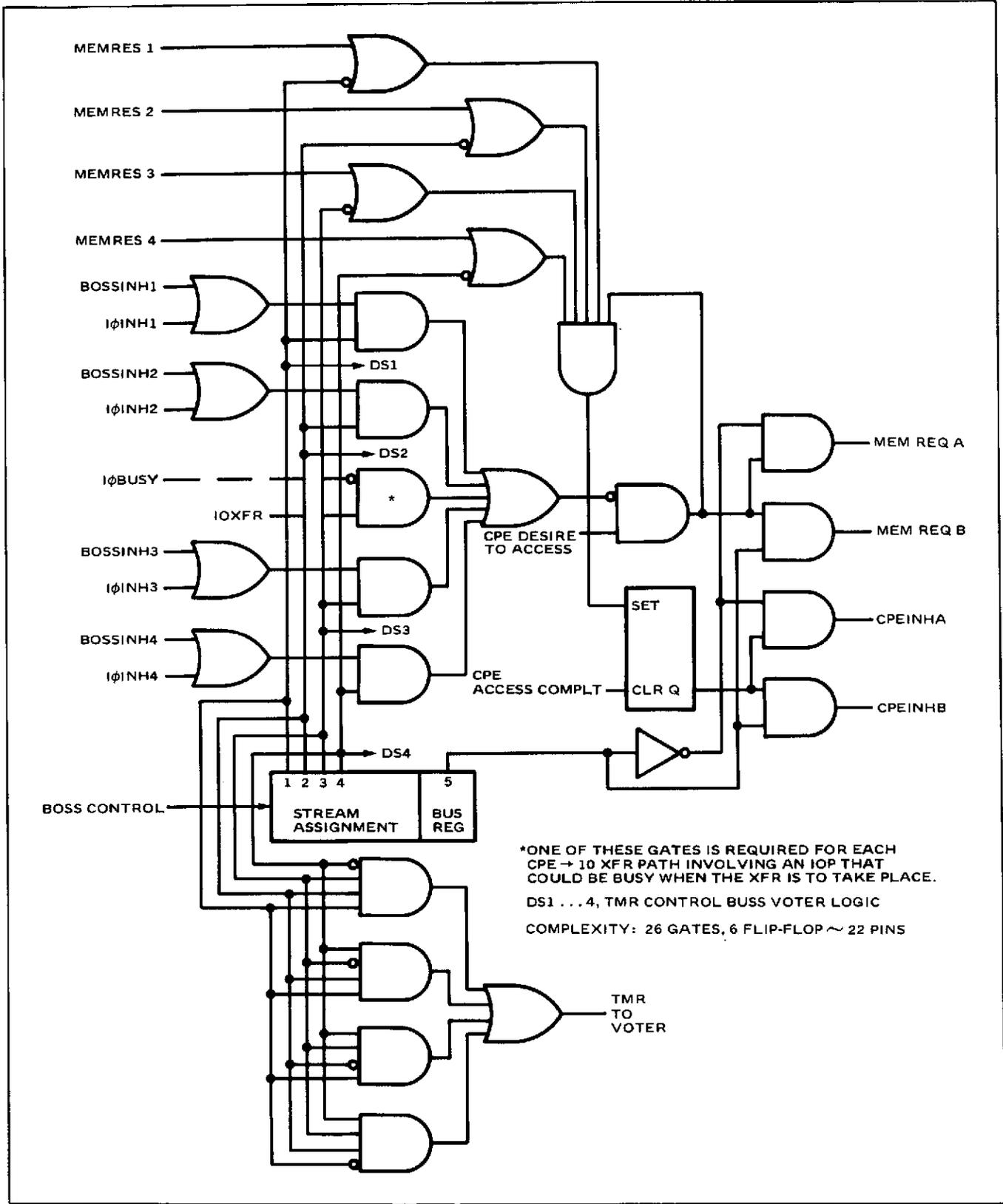


Figure 12. CPE Access Control Logic

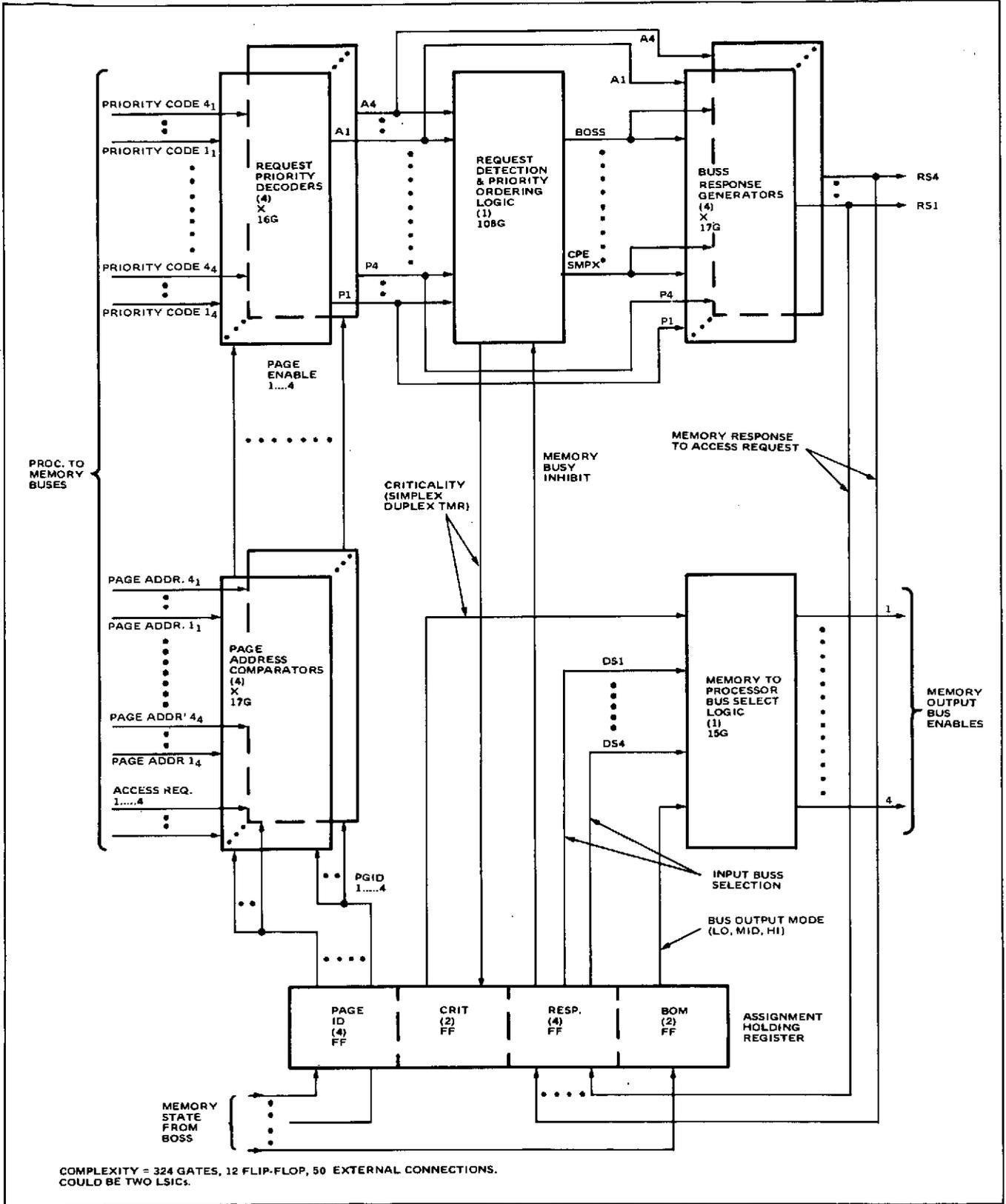


Figure 13. Memory Access Control Logic - (16 Priority Levels)

own decisions allowing for masking most errors in the duplex mode and detecting them in simplex. In processor modules the voter paths are controlled by the Stream Assignment Register while in memory modules they are under the control of the Response and Criticality fields of the Memory Assignment Holding Register. This logic allows for maximum software flexibility in the ARMMS configuration process with a moderate amount of hardware.

III ARMMS Error Correction Strategy

A major ARMMS objective is to provide an efficient trade-off between a high-reliability mode in which most faults are masked and a parallel processing mode where faults may cause momentary disruption of activity but must not preclude recovery. Originally these were envisioned as TMR and simplex modes respectively.

From the mission profile analysis of Phase I it became clear that the majority of the processing load has to be performed in a high reliability mode. However this implies only that any error should be detected prior to propagation, not that it should be immediately corrected (i. e., masked), only a minority of tasks required immediate connection. Our Phase II hardware reliability studies indicate that both processor and memory modules can be made to detect most faults and mask many others at the cost of a complexity increase on the order of 20% so that even in the simplex mode 99% of all faults are estimated to be detected and 55% to be masked. In the duplex mode detection is expected to be virtually 100% and 99% masking is expected to be achieved. If these results were compared to duplex and TMR operation with non-redundant modules respectively we see a hardware decrease of from 40 to 20% achieved for comparable operational reliability (i. e., if a single non-redundant module has a complexity of 1.0 and an error coded module has a complexity of 1.2, then the comparisons are 1.2 vs. 2.0 and 2.4 vs. 3.0 modules). Most programs would be run in error-coded simplex, some in error coded duplex and for some missions a TMR processing mode might not even be necessary to achieve ARMMS desired degree of reliability.

Figure 14 shows the ARMMS data path used for error analysis. The objectives of this study were to determine:

1. Which codes are most efficient for error detection
2. Is error detection adequate or is error correction also beneficial
3. Where should the error coding logic be located
4. How should error detecting/correcting codes interact with TMR voter switches for maximum benefit.

2-37

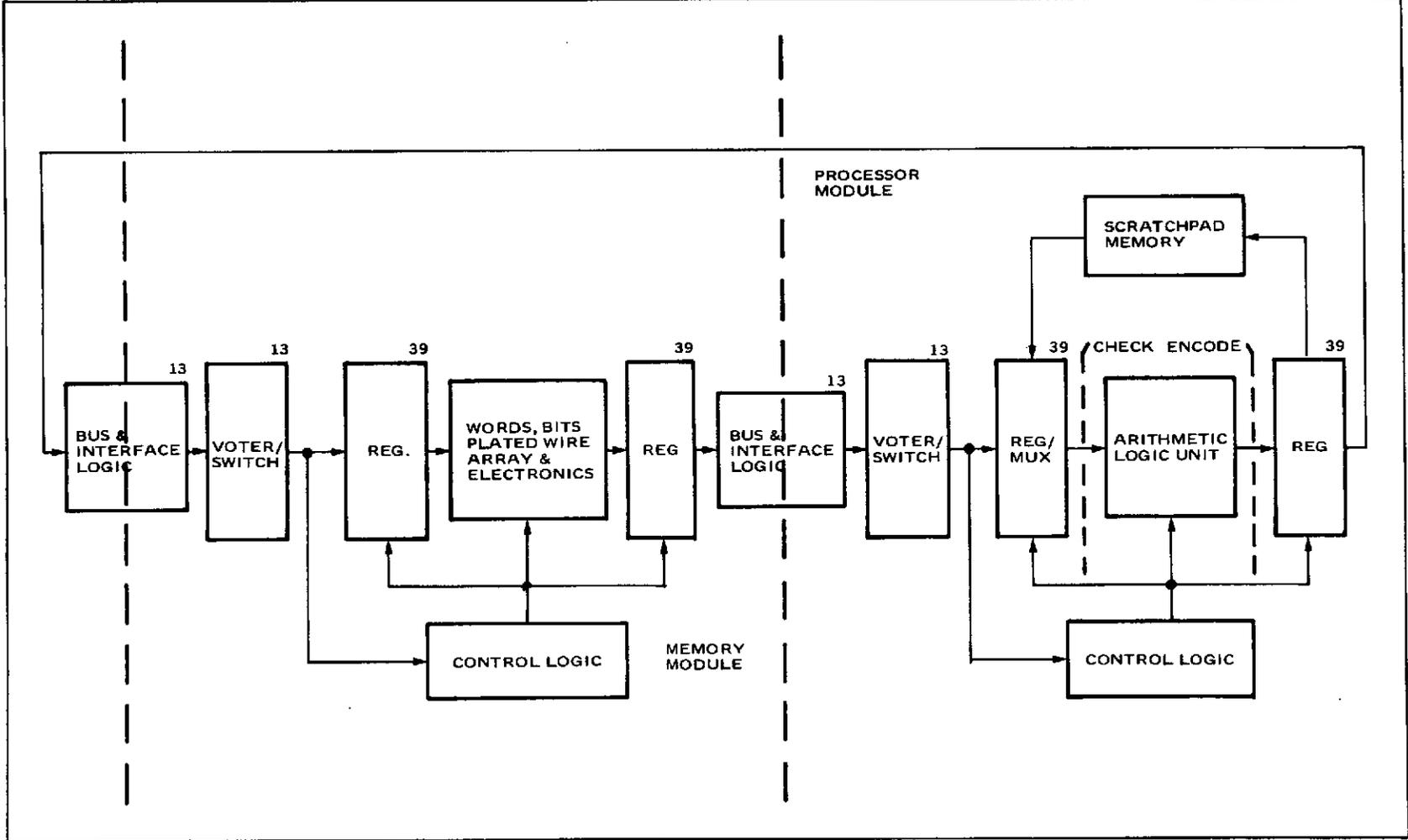


Figure 14. ARMMS Data Path For Error Analysis

As will be discussed in the sections on memory and BOSS processor design 55% of memory and processor logic is subject to faults effecting only a single bit of a word. Thus, a single error correcting code can mask failures in this logic. This provides a strong argument for the use of a code that can correct as well as detect errors since such codes are not difficult to generate.

Choosing an ARMMS Error Code

Originally a residue code was considered for ARMMS error detection because it is not destroyed by arithmetic operations in the processor. However our recent studies indicate that only 10 to 15% of the processor's logic is involved in arithmetic operations and of course no memory logic is. Further, while residue codes detect errors, they do not correct them. Finally, if a residue code is internally generated in each processor and no speed penalty is to be allowed for this process at least as much residue codes logic would be required as for duplicating the processor's ALU and comparing outputs. Therefore a Hamming plus overall parity code is recommended along with duplication of ALUs in all CPE modules, rather than using residue codes. BOSS ALUs need not be duplicated if BOSS partitions will never be operated in the simplex mode and our reliability studies indicate that with 4 BOSS partitions the chance of 3 failing within 5 years is less than .0002; hence BOSS need not be operated in simplex.

Six code bits are required for single-error correction of 32 bit words using a Hamming code. If an additional overall parity bit is used in addition to the Hamming code all odd numbers of errors will be detected and the combination of these two codes will detect up to 3 errors and 50% of error combinations involving more than 3 errors. As illustrated in Figure 15, checkers could be placed either in each memory module or in each processor module but since most ARMMS configurations should contain fewer processor modules than memory modules and the Hamming-parity code checkers should ideally be located as close to the ALU as possible for maximum coverage with minimum hardware, both checkers will be placed in each processor module. A Hamming code is generated as follows:

1. n bits provide SEC/DED protection for $k=2^n-n-1$ data bits
2. Example for $n=3$ and $k=8-3-1=4$:
Construct Parity check matrix whose columns are all non-zero code vectors.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$C_1 \quad C_2 \quad D_1 \quad C_3 \quad D_2 \quad D_3 \quad D_4$

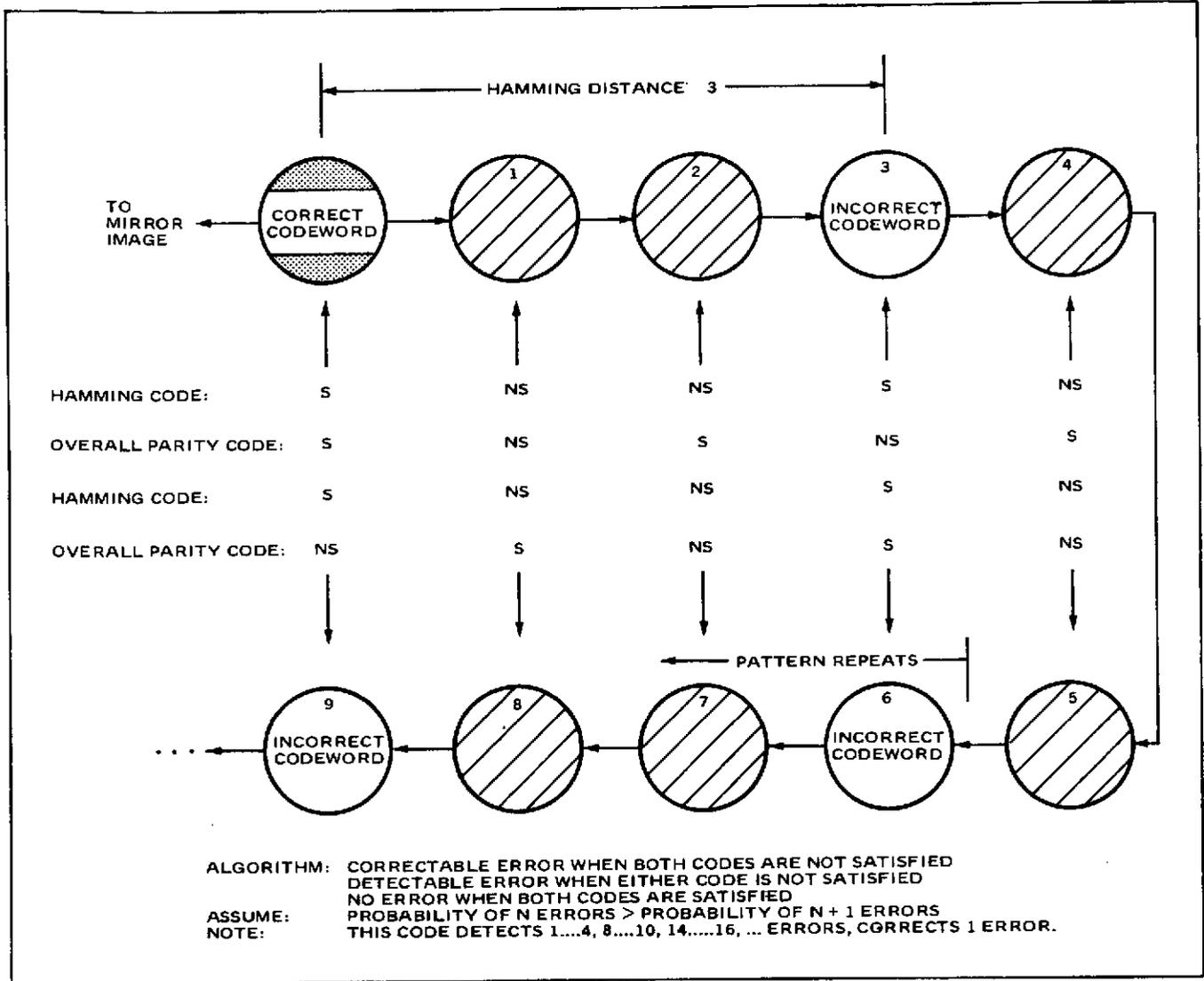


Figure 15. Hamming Plus Parity Code

- All columns containing a single "1" are code vectors. Other column vectors can be expressed as linear combinations of these code vectors.

$$C_1 = D_1 \oplus D_2 \oplus D_4$$

$$C_2 = D_1 \oplus D_3 \oplus D_4$$

$$C_3 = D_2 \oplus D_3 \oplus D_4$$

- States of (C_3, C_2, C_1) now indicate which bit is in error:

(C_3, C_2, C_1)	Bit in Error
001	C1
010	C2
011	D1
100	C3
101	D2
110	D3
111	D4

- These decoded states are used to invert the bit in question correcting the single error.
- A parity code over the data and code bits allows detection of multiple errors in addition to correcting the single error.
- Assuming LSICs of ~ 250 gates each.

32 bit Parity Check	1/2 LSIC	x2
32 bit Hamming + Parity Check	3/4 LSIC	x2
32 bit Error Correction Add-On	3/4 LSIC	x1
Total Logic per Processor	3-1/4 LSIC	

(6 Hamming Code bits plus a parity check bit are required to encode a 32 bit word in this way.)

Error Coder – Voter/Switch Interactions

It has been determined that the simplest voter/switch design would pass data to a code checker and registers in the simplex mode, compare data bit-by-bit outputting "1" to the code checker and registers in the duplex mode, and vote on the data in the TMR mode. This requires only one holding register and one code checker per module. It masks single bit errors in all modes, and "no output" and multiple "Stuck on 0" errors in all but the simplex mode, while detecting single bit, not output, and many multiple bit errors in all modes. In duplex or TMR operation, if 2 processors both show a data error this places the blame on the memory. If only one shows an error blame is placed on the processor showing the error and its output is set identically to "0" for that operation in which case the memory module's voter/switch will accept the output of the good module as noted above.

The method just discussed (Method B) along with an alternative (Method A) are shown in Figure 16. Method A has an advantage over Method B only in duplex operation where it can isolate faults to a specific module rather than to a module class and can mask multiple "stuck on 0" errors from a module that would otherwise be detected but not corrected. Method B is recommended due to its simplicity – saving about 12 ICs per module over Method A or perhaps 400 ICS in a large ARMMS configuration. Figure 17 is a detail of the voter switch logic of Figure 16 for one data bit. This logic will be duplicated 13 times for a 13 bit bus.

Most error code logic resides in the processor modules. Errors are detected and corrected at the ALU input and data is encoded at the ALU output. Error detection and correction can be implemented at a cost of under 4 LSICs (250 gates each) per processor. This is approximately the same amount of logic that would have been required to implement a residue code checker and about twice what would have been required for parity checker plus voting.

Error Detection and Masking Procedure

When errors occur, the processor masks the error if possible, otherwise it attempts a roll-back. If masking or roll-back is successful the processor completes its task before interrupting BOSS, in all modes. Once BOSS is interrupted it will usually have to run software diagnostic routines on a module to positively verify the failure and then replace the offending module.

BOSS places modules considered to have failed at the bottom of the spare module queue and then tries other modules in the queue until a good module (hopefully) is found. BOSS then places the good module on line and continues the computation. ARMMS will be considered to be failed if, and only if, BOSS cannot find a working module in each class by the above

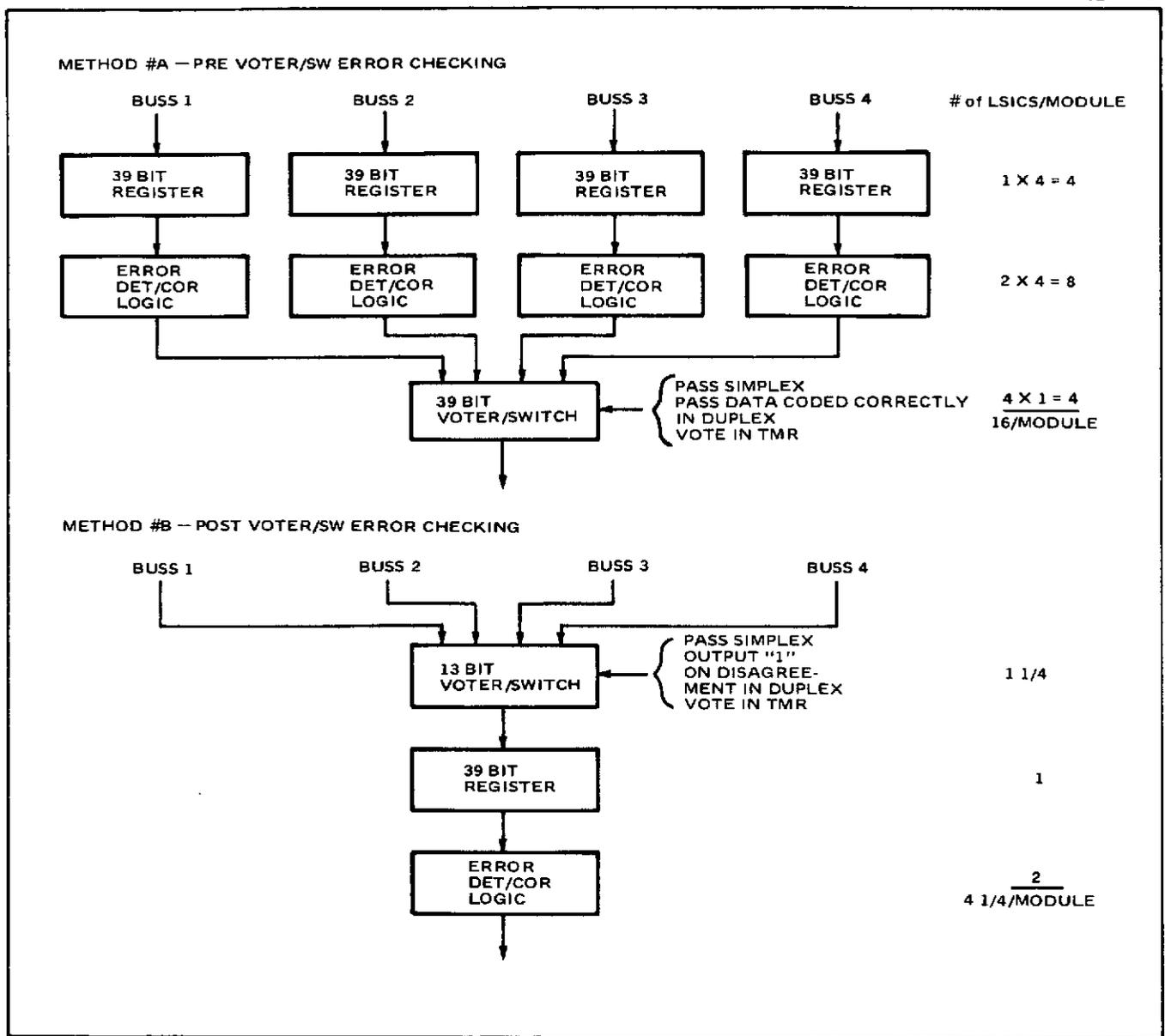


Figure 16. Error Checker/Voter-Switch Interaction

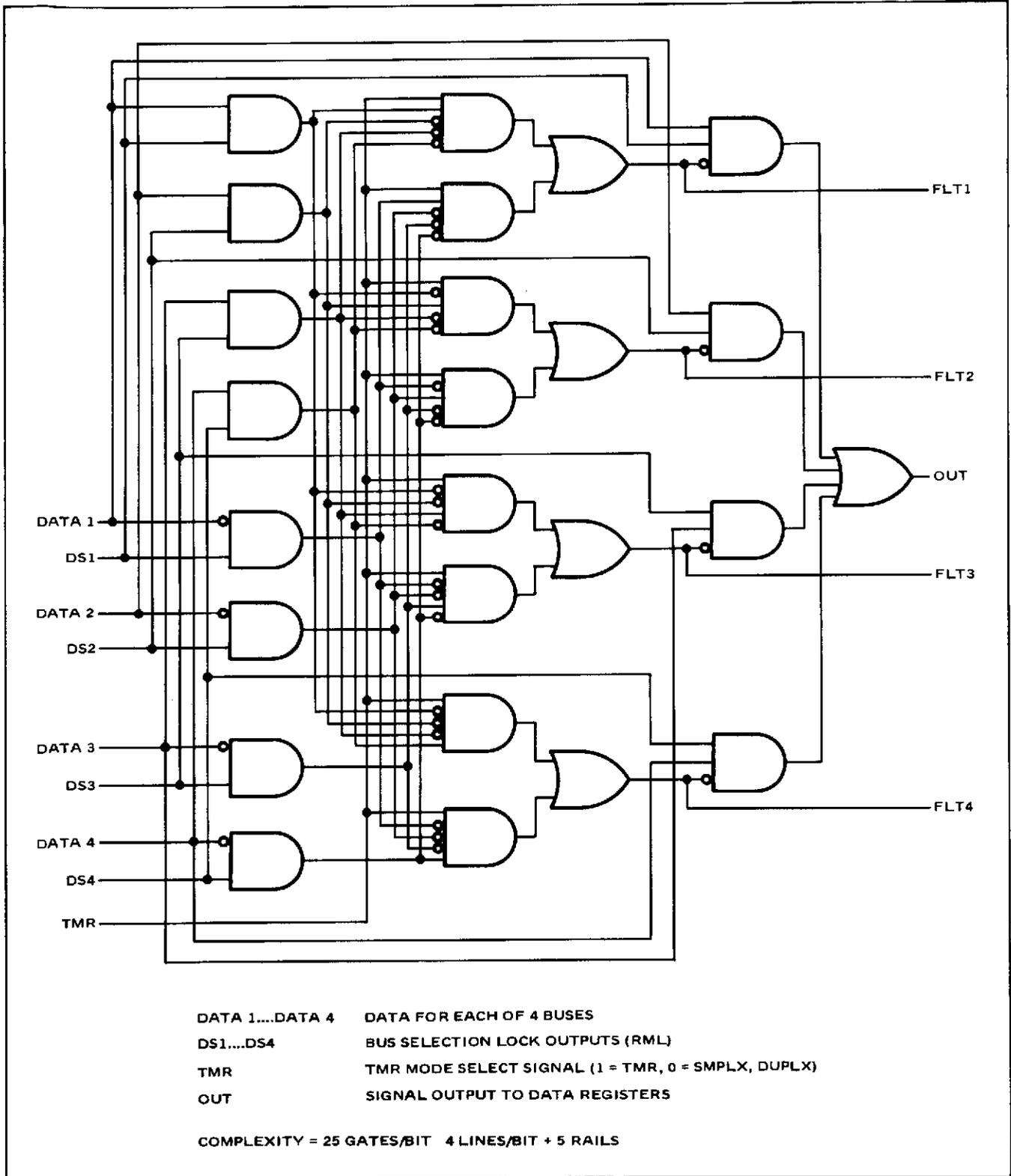


Figure 17. Universal Bus Voter/Switch (One Bit Slice – 13 Required Per Module)

procedure or an erroneous computation given undetected. A module is not considered to have failed until that failure manifests itself either through errors detected during normal operation or during a periodic software diagnostic routine.

Estimates of ARMMS intermodule error detection coverage have been made assuming the module designs discussed in the next sections. These results are summarized in Table VII, and show the relative likelihood of various types of memory and processor errors and whether or not they can be expected to be detected and/or corrected as a function of stream criticality (simplex, duplex or TMR operation).

TABLE VII. ARMMS INTERMODULE ERROR DETECTION COVERAGE STUDY RESULTS

	Simplex	Duplex	TMR	Rel. Prob.	
Memory Bit Errors	Correct	Correct	Correct	.60	Mem
Memory Word Errors	Detect	Correct	Correct	.39	
Bus/V-Sw. Errors	Detect	Detect	Correct most	.01	
Processor Bit Errors	Correct	Correct	Correct	.15	Proc.
Processor Word Errors	Detect	Correct	Correct	.81	
Loss of Control Errors	May cause failure	Detect	Correct most	.03	
Total Detected	~99%	~100%	~100%		
Total Corrected	~55%	~99%	~100%		

IV Preliminary BOSS Register Level Design and Technology Study

A preliminary register level design and reliability analysis have been completed for BOSS along with a tentative basic instruction set and list of macro instructions. Effort in Phase III will further refine the BOSS design. A partitioned BOSS module should be capable of achieving a reliability of .9998 over a five year mission and would require approximately 100 LSICs (of 250 equivalent greater complexity each) to implement.

BOSS Functional Description

BOSS will execute routines for data scheduling, system test, repair, and configuration, and interrupt processing. For four simultaneous processing streams executing programs of an average of 5 msec. duration BOSS will execute at least 800 routines per second. To meet these function and speed requirements, BOSS will have to be a small special purpose computer including such instructions as LOAD, STORE, NO OP, JUMP, TEST, SPCJ, AND, OR, SHIFT, ADD, SUB, plus macro instructions to speed up frequently used processes such as table searches requiring correlations and list processing.

BOSS will look functionally similar to the SUMC CPE – however SUMC instructions such as Multiply, Divide, Square Root, Floating Point and double precision will not be needed and special system monitoring and control logic will be required in BOSS but not in the CPE. BOSS will be capable of accessing and testing half-words, bits, and variable length fields for efficiency in list handling. If modified SUMC related design is used for BOSS, speed requirements would limit average BOSS program lengths to about 875 operations per task assuming 4 streams operating simultaneously with a 5 msec average task length. Individual BOSS processor partition complexity is expected to be from 50 to 60% of the present SUMC complexity or from 70 to 80% of that of a modified SUMC processor.

Originally BOSS was envisioned as a group of identical modules any three of which could be operated in TMR to provide ultra-high reliability. However, as mentioned in the configuration section, BOSS will have nearly 300 system level interconnects and if a group of BOSS processors were used each one would need almost this many interconnects. In addition, with individual BOSS processor modules, location of BOSS power and configuration control, command voting, oscillator and power supply logic becomes a problem. One solution is to group these functions into a very simple and hence very reliable internally redundant "super-BOSS" module. The interconnect problem which can effect both volume and reliability is solved by grouping the BOSS processors and the "super-BOSS" physically into one module requiring only one set of system level interconnects. The BOSS processors and the "super-BOSS" become partitions "A" and "B" respectively. Reliability estimates based upon BOSS register level design indicates that 4 "A" partitions and 2 "B" partitions should meet ARMMS reliability goals.

BOSS Reliability Analysis

By operating BOSS in at least a duplex mode (and in TMR so long as possible) failures in other BOSS logic will be detected – particularly those in the ALU and control logic blocks. Parity checks can be performed inexpensively on BOSS memory and the Hamming Parity logic is required in order to check

the main memory, keeping the cost associated with self-checking BOSS to a minimum. Over 90% of BOSS failure modes can be masked to allow continued operation so long as at least 2 BOSS partitions "A" are operational. BOSS is estimated to have a .9998 probability of successful duplex operation after 5 years and a .9976 probability of continued TMR operation over that period, assuming 4 partitions "A" are flown. These reliability figures assume the register level designs shown in Figure 18. Table VIII lists BOSS failure modes along with resultant error patterns, failure rates and suggested corrective action as a function of the component block failing. Table IX gives a preliminary CMOS LSIC functional partitioning estimate for both BOSS and memory modules. It is expected that most BOSS integrated circuit designs would be usable in the CPE as well and this will be investigated further in Phase III. Memory modules are anticipated to use 6 chips of 3 types and BOSS partition "A"s are anticipated to use 25 chips of 9 different types. The dashed lines in Figure 18 delineate these partitions.

Referring to Figure 18 similarities can be seen between BOSS and SUMC since SUMC was used as a starting point. However, the memory Input and Instruction registers are duplicated to allow for instruction overlapping, there is no MQ register or floating point unit since these functions are not needed in BOSS, error detection logic and bus interfaces and voting logic have been added, and the ALU-multiplexer structure has been simplified. At a detailed level radical changes are expected in the structure of the microprogram read-only-memory and scratchpad memory and in general the design has been simplified and streamlined to increase the processor's speed and ease error detection and correction. Hence SUMC hardware is not likely to be useful for BOSS and the CPE should probably be an extension of the BOSS design rather than a modification of SUMC since these should tend to maximize commonality and minimize cost within ARMMS.

Referring to Table VIII it can be seen, assuming no duplication of the ALU logic within a BOSS partition and at least duplex operation, that 75% of BOSS failure modes will be maskable and that virtually all will be detectable. In TMR operation, virtually all failures can be masked. The number in the table should also be representative of CPE failure rates except that with duplication of ALU and floating point arithmetic logic the conditional probability of being able to detect a failure given that one occurs while operating with simplex mode rises accordingly. As noted earlier simplex operation of BOSS is not necessary or desirable in ARMMS while simplex processor operation is both to be expected and desirable. Note that parity checks are made on BOSS internal memories. Ideally one parity bit would be included in each word in each memory chip so that parity could be computed for each chip allowing testing for an "all 0" output condition in addition to testing for odd numbers of stuck bits. Certain on-chip addressing logic problems are not detectable

2-47

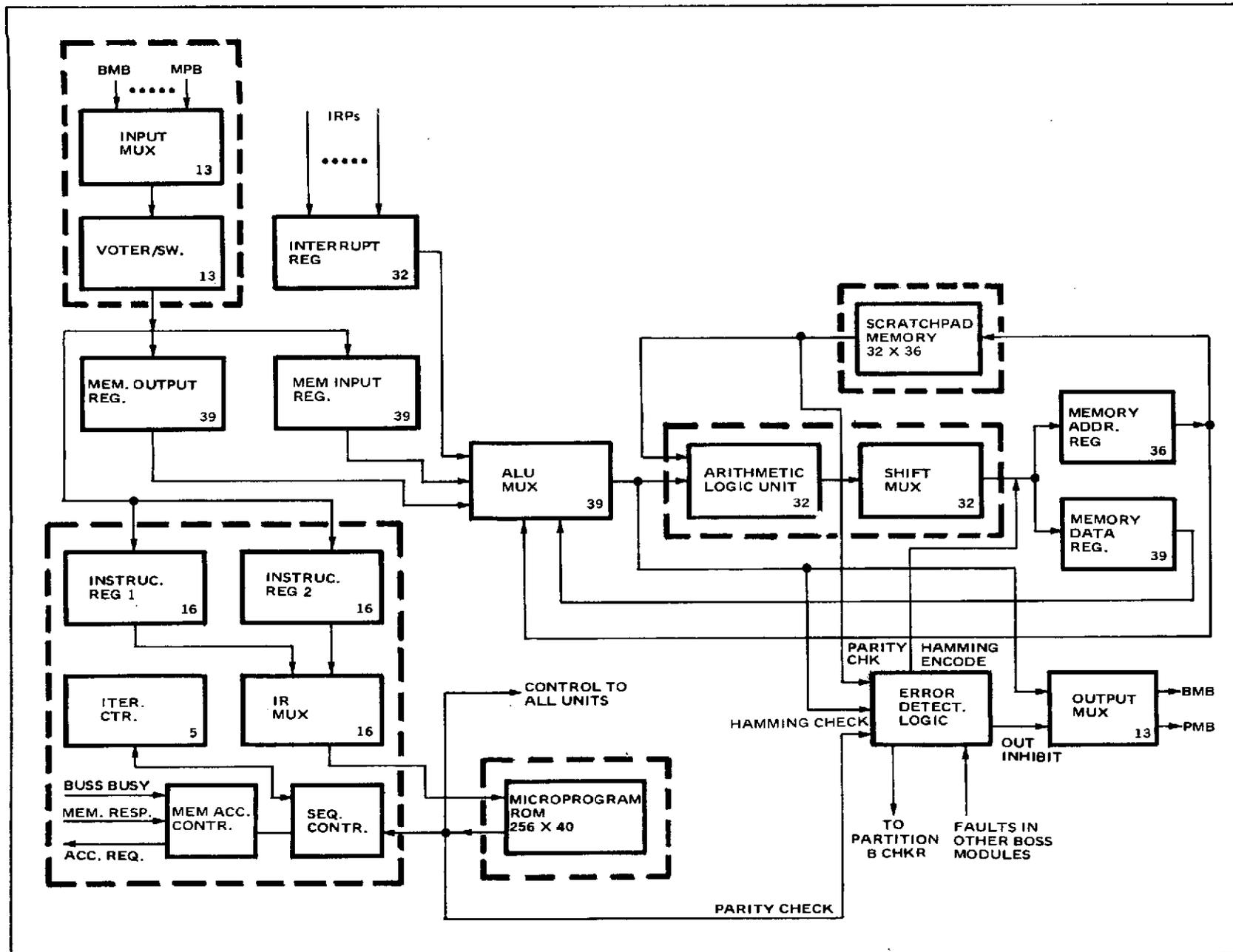


Figure 18. ARMMS BOSS Functional Block Diagram

TABLE VIII. BOSS FAILURE MODES

Components Failing	Result	Failure/10 ⁶ Hrs	Corrective Action
Input Mux or Voter/Switch	Triple-Bit Error	.04	Detect with H-P Code- Inhibit Output
Mem In, Addr, Data Reg Output and ALU Muxes	Single Bit Error	.12	Detect and Mask with H-P Code
Scratchpad Memory (SPM)	Single Bit Error	.10	Detect with parity check inhibit out est. coverage = 0.9
Arithmetic Logic Unit	Multiple-Bit Error	.10	Detect in duplex, mask in TMR. Est coverage = 0*
Microprogram ROM (MROM)	Control Bit Error	.10	Detect with parity check inhibit out est coverage = 0.9
Instruc. Reg. & Mux	SPM or MROM Addr bit error	.02	Detect by comparing with memory in reg-inhibit out
Interrupt Reg., Iter Ctr, Seq & Mem Acc Contr	Improper Execution, Loss of Sync.	.02	Detect in duplex, mask in TMR est. coverage = 0
Error Detection Logic	False Error Indication	<u>.10</u>	Inhibit Output
TOTAL		.60	

*This failure mode could be detected and masked internal to the partition by duplication of the ALU and comparing outputs. Since BOSS is not to be operated in simplex this redundancy is not necessary nor recommended although it is desirable in processor modules.

TABLE IX. PRELIMINARY CMOS LSIC PARTITIONING ESTIMATE

	Function	Bit Width	Quantity
Memory (6)			
1.	Voter/Sw-Register-Output Mux	3	4
2., 3.	Access and BOSS Control Logic	N/A	2
BOSS (25)			
1.	Voter/Sw. - In and Out Mux	7	2
2.	Data Reg. - ALU Mux	8	5
3.	ALU - Shift Mux - H/P Corrector	8	4
4., 5.	Error Detection	N/A	4
6., 7.	Sequence-Access-BOSS Control Logic	N/A	2
8.	Scratchpad Memory (32 Word)	9	4
9.	Microprogram ROM (256 Word)	10	4

Most BOSS chips should be usable in the CPE as well.

250 gates/chip complexity CMOS logic assumed throughout.

with a parity check therefore memory coverage is expected to be ≈ 0.9 rather than unity. Coverage is assumed to be unity in the tables "corrective action" column except as noted. When one partition's output is inhibited, the memory module's voter/switch will mask this output allowing the other partition's correct output to propagate to the memory. The same thing is true of partition "B" command voting logic.

Given a non-maskable failure in a BOSS partition the replacement algorithm implemented a partition "B" is as follows:

1. Power on partitions 1, 2, 3 at the start of the mission.
2. Replace the first failing partition with partition 4 (prob .0806).

3. Power off the second failing partition – BOSS is now in duplex operation (prob .0024).
4. If a third, non-maskable failure occurs (prob. .0002) ARMMS will cease operating and wait for outside assistance. Retrying BOSS partitions can be done on command but will not be done automatically since this won't necessarily correct the failure and can lead to undetected erroneous computations being outputted from the computer.

Partition B is statistically very reliable but conservative design calls for providing a spare partition to be switched in automatically upon self-detected disagreement within the first partition.

BOSS Register Level Design

The BOSS microprogram read only memory organization is summarized in Figure 19. Bits have been provided to implement all BOSS micro and macro instructions discussed later in this section. This MROM would have to be modified for CPE operation. Fields are included for interrupt, scratchpad memory, ALU, hardware register, bus interface, and sequencer control functions. Each MROM word uses 36 bits plus parity and 256 words are provided reducing the memory to one eighth the size of the one in SUMC.

Figure 20 shows the BOSS instruction and data formats. Three types of instruction and one data format are recognized: Main memory reference instructions contain an address for a 2nd operand fetch including a choice of 3 index registers, 3 base/bound registers plus a no index or base register option when these fields are "0". An 8 bit op-code accesses the MROM directly with the op-code of an instruction being the MROM address of its first micro instruction. Two register addresses are provided for accessing two words from scratchpad memory during the course of the instruction. Field R_1 can access any non-privileged SPM location while Field R_2 accesses the lowest 8 accumulators. Single operand instructions have formats the same as above except that a third general SPM register may be accessed with Field R_3 rather than a main memory location. Link word instructions are used for list handling and provide two main memory address fields allowing indirect address linkage to a data item in main memory and to the next link in the list. Data words allow 32 bit signed fixed point data to be accessed by BOSS.

Figure 21 shows the organization of the BOSS scratchpad memory. It contains 23 accumulators, plus 3 base and 3 index registers directly accessible by the program. In addition a rollback program status word (RPSW) and interrupt status word (IPSW) provide for program jumps on errors and interrupts and three base registers provide for extended main memory access when summed with an instruction displacement field. The RPSW, IPSW, and Program counter are read accessible but not write accessible under normal conditions.

2-51

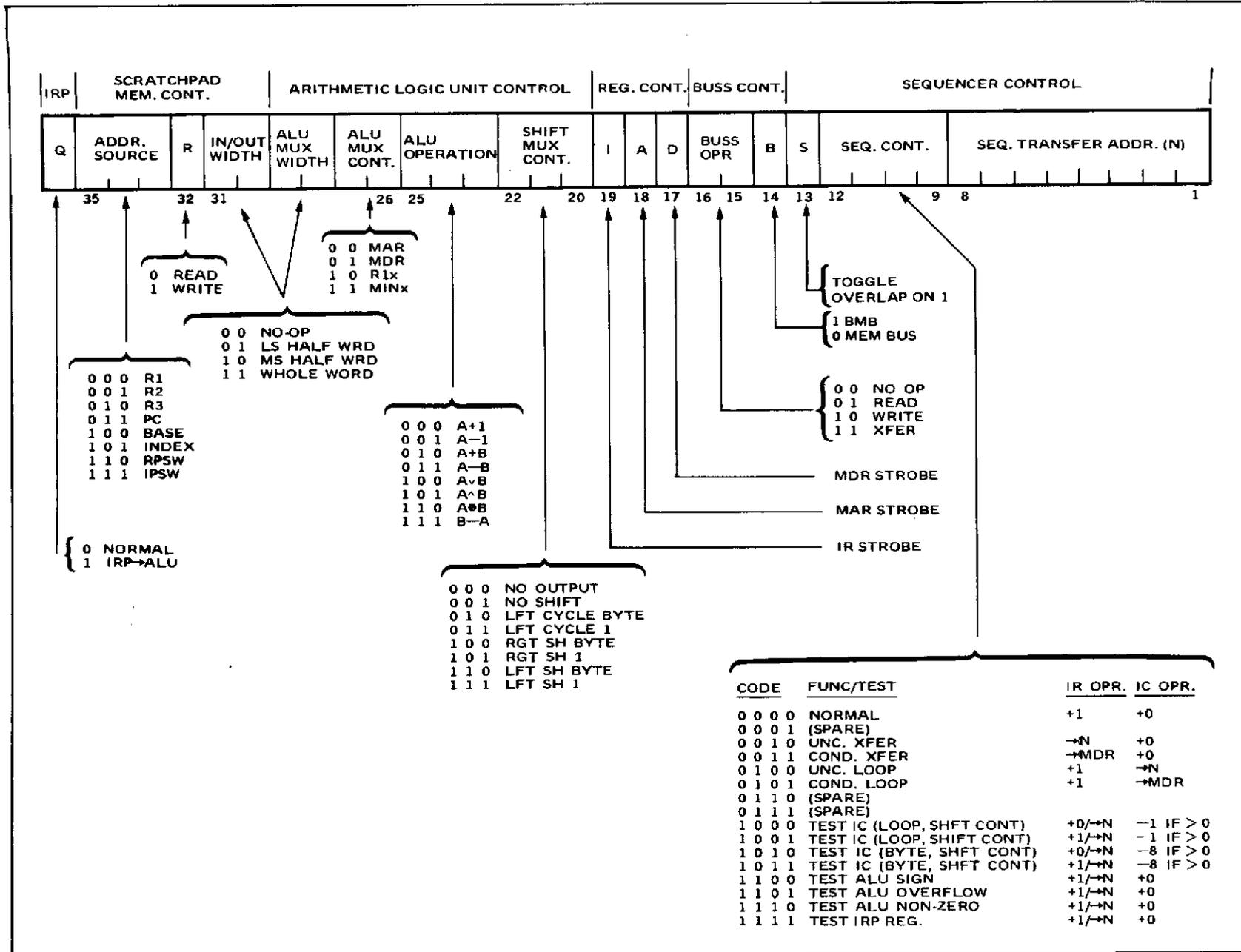


Figure 19. BOSS Microprogram Memory Organization

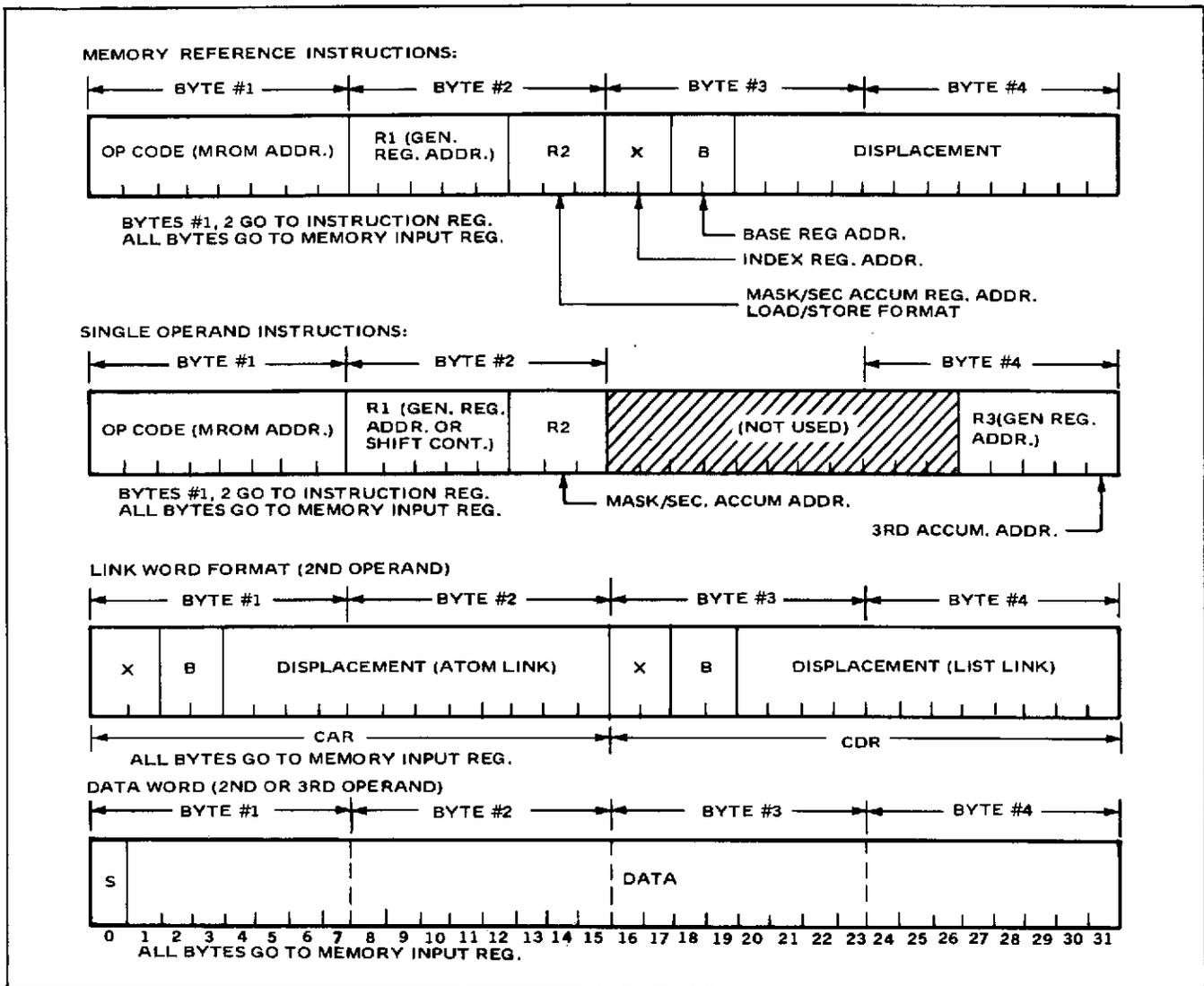


Figure 20. BOSS Instruction and Data Formats

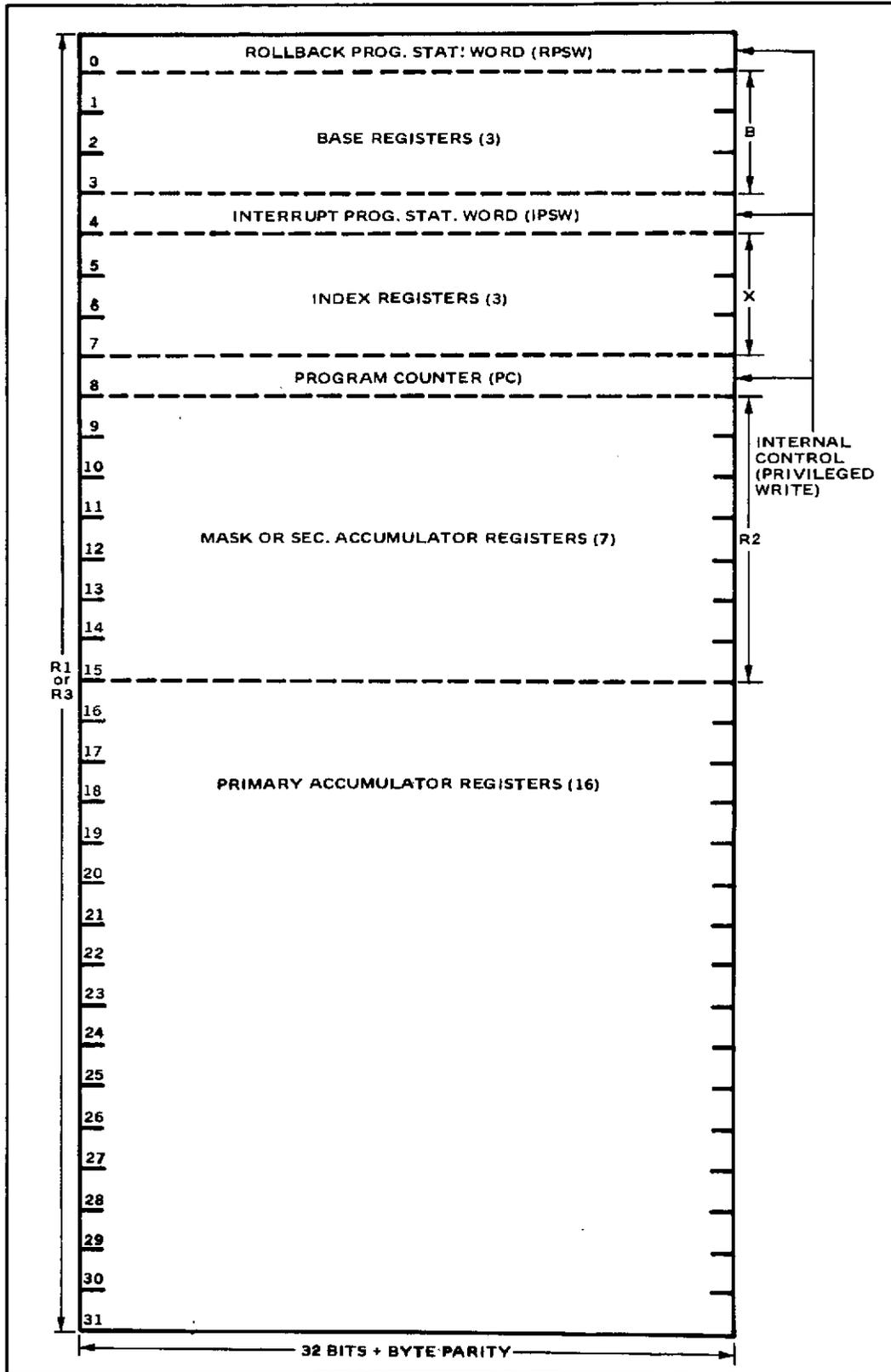


Figure 21. BOSS Scratchpad Memory Organization

BOSS Interaction with Other Modules

BOSS will command and interrogate other modules via a 2-way BOSS/Module bus (BMB). Each module will contain bus interface logic capable of decoding a unique access code for that module plus a general sync code which allows simultaneously starting several pre-primed processors working together in the same stream. The interface logic will also gate the module's status word MSW onto the BMB in response to an interrogate command from BOSS to the module. Both processor and memory MSWs would contain their BOSS assignments (memory page, processor bus access code) and in addition memories could use a one bit code to indicate failures and the CPEs would include a 4 bit status code as follows:

Error Code (2)	Termination Code (2)
00 No Error	00 Normal termination unless error code > 0
01 Memory Error	01 Memory Page available
10 Processor Error	10 Lock variable request
11 Undetermined Error	11 Unlock variable request

BOSS would then use the code to determine which subroutine to branch to in response to the processors' status. BOSS could interrogate processors periodically or in response to interrupts from them. Descriptions of, and formats for, BOSS commands to other modules are shown in Figure 22. The "save" and "restore" data commands cause the processor to store or load data respectively from an area of memory defined in the commands. This allows BOSS access to the processor's registers including privileged Base/Bound registers not accessible by general programs. Transmission on the BMB will be parity coded and a sync line is included to activate modules' access decoders. The BMB is duplicated so that modules can verify accuracy of commands through comparison of signals on the 2 buses and BOSS can likewise verify data from the modules. Further hardware details were included in the configuration "C" description section of this report.

BOSS Instruction Set

Recently M&S computing proposed a set of BOSS macroinstructions covering bit and byte testing, byte, half-word and field load and store instructions and a set of instructions for formation and manipulation of linked lists. These instructions were designed to allow rapid, efficient manipulation of various tables, lists, queues, and other data structures contained in BOSS memory. Their macroinstruction set has been analyzed and modified where necessary to fit it into the framework of a 32 bit word machine. Some of the instructions have been combined and made more powerful where doing so did not reduce BOSS speed.

<u>CODE</u>	<u>COMMAND</u>	<u>MEMORY</u>	<u>PROCESSORS</u>	<u>ARGUMENT (6)</u>
00	STOP - SAVE DATA		X	MEMORY ADDR.
01	RESTORE DATA* - PRIME FOR SYNC START		X	MEMORY ADDR.
10	TRANSMIT MSW	X	X	SUBCODE = 0
11	LOAD ASSIGNMENT REG	X	X	ASSIGNMENT
10	SYNC START		X	SUBCODE = 1

*GIVES BOSS WRITE ACCESS TO PRIVILEGED BASE/BOUND REGISTERS.

FORMAT:

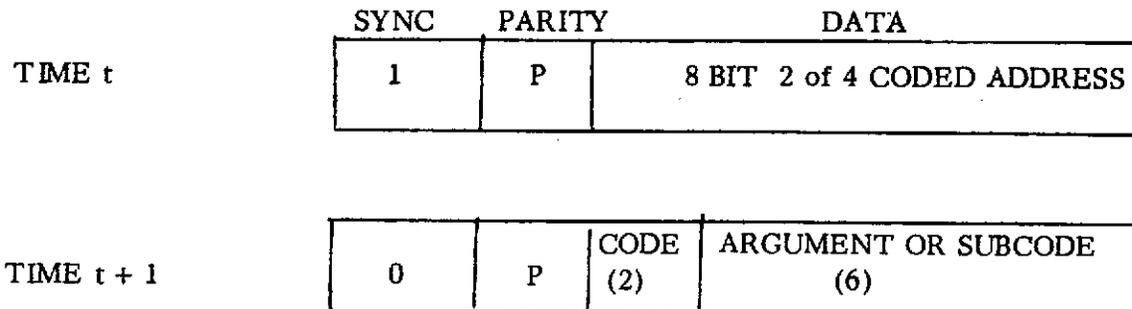


Figure 22. BOSS to Module Commands

TABLE X. BOSS MACRO-INSTRUCTION DESCRIPTIONS

1. GENERALIZED "CLEAR AND ADD" AND "STORE" INSTRUCTIONS

<u>Boss Mnemonic</u>	<u>M&S Mnemonic</u>	<u>R2 Field</u>	<u>Left Halfword Oper.</u>	<u>Right Halfword Oper.</u>
CLA, STO	LH, SH	x01	NO-OP	LOAD, STORE
CLA, STO	LH, SH	x10	LOAD, STORE	NO-OP
CLA, STO	—	x11	LOAD, STORE	LOAD, STORE
CAI, STI	XCDR, SCDR	x01	NO-OP INDIRECT ADDR.	INDIRECT ADDR. LOAD, STORE
CAI, STI	XCAR, SCAR	x10	LOAD, STORE	NO-OP
CAI, STI	—	x11	INDIRECT ADDR. LOAD, STORE	INDIRECT ADDR. LOAD, STORE

TIMING = CLA = 1.2 μ SEC STO = 1.5 μ SEC CAI = 1.8 μ SEC STI = 2.1 μ SEC

MEMORY EST: CLA = 1 STO = 1 CAI = 2 STI = 2 WORDS

2. GENERALIZED TEST INSTRUCTIONS

(ARGUMENTS ARE ASSUMED TO BE STORED IN RESPECTIVE REGISTERS PRIOR TO EXECUTION OF THESE INSTRUCTIONS)

BON	R_1, R_2, A	BRANCH IF BIT ON
BOF	R_1, R_2, A	BRANCH IF BIT OFF
TUM	R_1, R_2, A	TEST UNDER MASK, BRANCH ON EQUAL
TDM	R_1, R_2, A	TEST UNDER MASK, BRANCH ON EQUAL, ELSE DECREMENT INDEX

R_1 = BIT NO. TO BE TESTED IN BON, BOF

R_1 = GENL. REG. TO BE COMPARED WITH MEMORY
IN TUM, TDM

R_2 = BRANCH ADDR. IN ALL INSTRUCTIONS

A = ADDRESS (INCL. BASE & INDEX) OF MEMORY
LOCATION UNDER TEST

$R_2 + 1$ = ADDRESS OF 32-BIT MASK IN TUM, TDM

INDEX REGISTER TO BE DECREMENTED IN TDM IS SPECIFIED
BY THE X PORTION OF A.

TABLE X. BOSS MACRO-INSTRUCTION DESCRIPTIONS (Continued)

2. GENERALIZED TEST INSTRUCTIONS (Continued)

NOTE: THERE IS NO ROOM FOR INCLUSION OF A MASK FIELD WITHIN THE FORMAT, THEREFORE M&S TUM WAS MODIFIED AS SHOWN AND BYTE INSTRUCTIONS DELETED.

TIMING: BON, BOF = 2.0 SEC TUM = 2.2 SEC TDM = 2.4 μ SEC
 BON, BOF = 4 TUM = 5 TDM = 6 WORDS

3. GENERALIZED PARTIAL WORD INSTRUCTIONS

(ARGUMENTS ARE ASSUMED TO BE STORED IN RESPECTIVE REGISTERS PRIOR TO EXECUTION OF THESE INSTRUCTIONS)

CLF, R₁, R₂, A CLEAR AND ADD MASKED FIELD
 STF R₁, R₂, A STORE MASKED FIELD
 R₁ = GENL. REG. TO BE LOADED OR STORED FROM
 R₂ = ADDR. OF 32-BIT MASK
 A = ADDRESS (INCL BASE & INDEX) OF MEMORY
 LOCATION CONTAINING BITS IN QUESTION.

BITS OF R OR A CORRESPONDING TO MASK POSITIONS CONTAINING "1" WILL BE CHANGED, REMAINING BITS WILL NOT BE CHANGED.

NOTE: THERE IS NOT ENOUGH ROOM TO SPECIFY M&S LF AND SF INSTRUCTIONS WITHIN A 32-BIT WORD. EQUIVALENT PERFORMANCE IS OBTAINED BY SPECIFYING A MASK AND JUSTIFYING THE GENERAL REGISTER POSITION PRIOR TO STORE AND FOLLOWING LOAD WITH SEPARATE SHIFT INSTRUCTIONS.

TIMING = CLF = 2.0 μ SEC STF = 2.1 μ SEC
 MEMORY EST: CLF = 4 STF = 4 WORDS

TABLE X. BOSS MACRO-INSTRUCTION DESCRIPTIONS (Continued)

4. LIST MANIPULATION INSTRUCTIONS – NO CHANGE FROM M&S SPECIFICATION (ARGUMENTS ARE ASSUMED TO BE STORED IN RESPECTIVE REGISTERS PRIOR TO EXECUTION OF THESE INSTRUCTIONS).

<u>BOSS Specification</u>	<u>M&S Mnemonic</u>	<u>Function</u>
NXT R ₁ , -, -,	NEXT W	STEP TO NEXT ITEM
INS R ₁ , -, A	INSERT W, A	INSERT A AFTER W
RMV R ₁ , -, -	REMOVE W	REMOVE W
FND R ₁ , R ₂ , R ₃	FIND W, M, C	FIND ITEM ACCORDING TO MASK
R ₁ ↔ W		WORD OFFSET IN THE (ASSUMED) ATOM TO BE FETCHED
R ₂ ↔ M		MASK WITH "1" BITS IN BIT POSITIONS TO BE COMPARED
R ₃ ↔ C		GENL. REG. CONTAINING WORD FOR COMPARISON
A		POINTER ADDRESS

TIMING: NXT = 2.1, INS = 5.4, RMV = 4.0, FND 3.0 μSEC/ITEM

MEMORY EST: NXT = 4 INS = 7 RMV = 5 FND = 11 WORDS

TABLE XI. TENTATIVE BASIC BOSS INSTRUCTION SET

Mnemonic	Instruction	Avail in SUMC	Timing μ sec	Microprogram Storage
JRE	Jump On Register Equal to Memory	Y	1.5	4
JRG	Jump on Register Greater than Memory	Y	1.4	2
JRN	Jump on Register Not Equal to Memory	N	1.5	4
JRL	Jump on Register Less than Memory	N	1.4	2
SPJ	Store Program Counter and Jump	N	2.1	2
JMP	Jump Unconditionally	Y	1.4	1
JPI	Jump Unconditionally Immediate	Y	1.4	1
XEC	Execute	N	0.8	1
ADM	ADD Memory to Register	Y	1.2	2
SBM	Subtract Memory from Register	Y	1.2	2
ANM	AND Memory with Register	Y	1.4	2
ORM	OR Memory with Register	Y	1.4	2
XOM	Exclusive OR Memory with Register	Y	1.4	2
ADR	ADD Register to Register	Y	1.2	4
SBR	Subtract Register from Register	Y	1.2	4

TABLE XI. TENTATIVE BASIC BOSS INSTRUCTION SET (Continued)

Mnemonic	Instruction	Avail in SUMC	Timing μ sec	Microprogram Storage
ANR	AND Register with Register	Y	1.2	4
ORR	OR Register with Register	Y	1.2	4
XOR	Exclusive OR Register with Register	Y	1.2	3
ICT	Increment Memory	N	2.3	3
NOT	Complement Register	N	1.6	3
DLY	Delay N Cycles	Y	0.8	1
HLT	HALT and Wait for Interrupt	Y	0.8	1
CWM	Compare Register with Memory	N	2.2	4
CSR	Compare Register Selectively with Register	N	2.2	5
SHR	Shift Right N Bits	Y	2.0	6
CYL	Cycle Left N Bits	Y	1.8	5
SHL	Shift Left N Bits	N	2.0	5
COM	Command Module via BMB	N	1.2	1
INM	Interrogate Module via BMB	N	1.2	1
LRR	Load Rollback Reg. from Program CTR	N	2.1	2

- NOTES:
1. Load and store instructions are included in the MACRO Table X.
 2. Speeds assume 10 MHz system clocks.
 3. Microprogram storage estimates assume an additional 6 word fetch routine.

The result was a set of 14 macroinstructions listed in Table X using an estimated 60 words of microprogram read-only-memory and having an average execution time of 2.4 μ sec each, assuming 10 MHz system clock. This compares with 30 basic instructions listed in Table XI having an average execution time of 1.4 μ sec and requiring 95 words of microprogram storage. It is expected that the BOSS instruction set will contain these macros plus many from the basic set and that further discussions will be held prior to choosing a final set for implementation later this year.

V Memory Module Reliability and Register Level Design Study

It is likely that the least reliable of the ARMMS modules will be the main memories due to the large number of discrete components and small scale integrated circuits required and the power levels associated with accessing the plated wire planes. Fortunately, however, analysis has shown that due to their organization it is possible to achieve 99+% memory reliability on a system basis through judicious use of error detecting and correcting codes which are generated and checked within processor modules and stored in each memory word, internal redundancy within memory modules, spare modules, and duplex memory operation for duplex or TMR processing streams. Software read-after-write in the simplex mode and duplication of data from a good memory into a spare memory in duplex or TMR modes would also be desirable. Using these techniques the results shown in Tables XII and XIII have been obtained. Table XII summarizes probabilities of occurrence of dominant failure modes along with recommended solutions while Table XIII lists various causes of memory failures again with their contributions to the memory module's failure rate. A block diagram of the proposed memory module is shown in Figure 23. The failure rates were derived from data in a 1971 Autonetics Space Station Study. The memory is assumed to use plated wire technology in an 8192 word by 39 bit (32 data bits plus error correcting codes) organization.

Memory Module Register Level Design

Plated wire technology was chosen for the ARMMS main memory because of its low, power, weight, and volume and non-volatility in the presence of power transients. Such memories are being used extensively in space computers being designed today for these reasons. The basic organization consists of a 512 word by 628 bit structure which is accessed in a 2-1/2 D configuration requiring 512 word drivers, a 628x39 low level bit multiplexer and 39 bit switch/sense amplifier circuits allowing 32 data bits plus 7 error detecting/correcting code bits per word. The memories' cycle time is assumed to be 600 nsec for READ and 800 nsec for WRITE. The details of the memories' control and voting logic were discussed in the configuration and error correction sections respectively of this report. The remaining logic is straightforward except for noting that since the memory must sometimes output data on one bus while the address for the next cycle is being inputted on another a one word Access-Request Buffer is required to hold the current address stable until the end of the memory cycle.

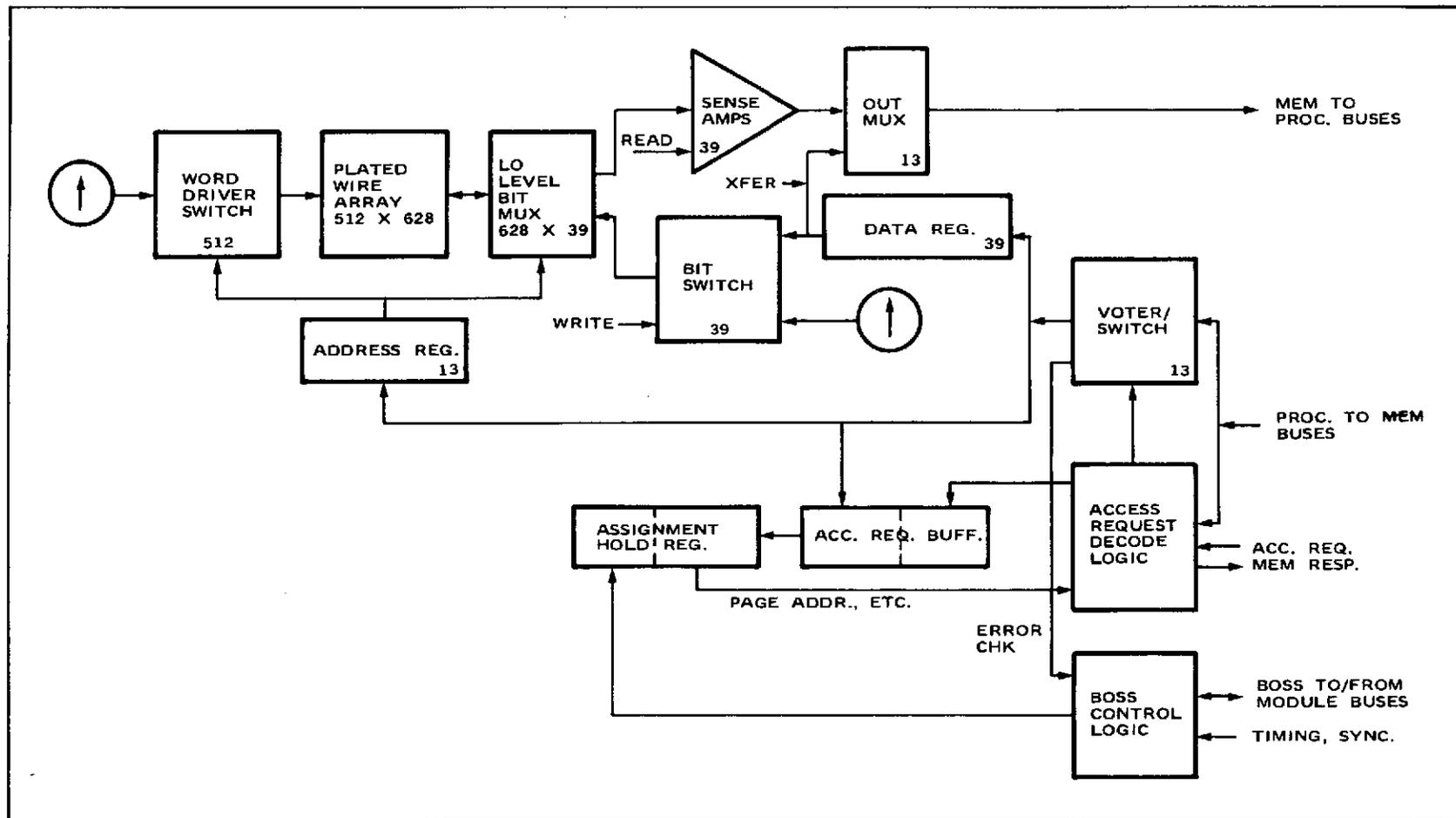


Figure 23. ARMMS Main Memory Functional Block Diagram

TABLE XII. DOMINANT MEMORY FAILURE MODES AND
RECOMMENDED SOLUTIONS

1.	Wrong Output of a Single Bit in Each of a Group of Words	
	Cond. Prob./Given Failure = $\sim .575$	
	Solution - All Modes	Hamming-Parity Error Masking Code
2.	No Output of all Bits in a Group of Words	
	Cond. Prob./Given Failure = $\sim .225$	
	Solution - Simplex	All "0" Output \rightarrow Parity Error \rightarrow Detection
	- Duplex	Voter/Switch Output "1" on Disagree-
	- TMR	ment \rightarrow Masking
		Majority Vote \rightarrow Masking
3.	Selection of Two Words in Memory at Once	
	Cond. Prob./Given Failure = $\sim .200$	
	Solution - All Modes	Employ Series Redundant Word Drivers
		to reduce this prob. to .0004
4.	Improper Memory Output Synchronization	
	Cond. Prob./Given Failure = .601	
	Solution - Simplex	None
	- Duplex	Detect Disagreement at Voter/Switch
	- TMR	Vote and Mask at Voter/Switch

TABLE XIII. ARMMS MEMORY FAILURE MODES

Component Failing	Result	Failures/10 ⁶ Hours		Corrective Action
		Basic Memory	Enhanced Reliability Memory	
Word switch or current source open, power supply failed	No output (whole words)	2.4	4.8(.005)*	Detect with Inv. Hamming-Parity code
Word switch shorted	Select 2 words at once	2.4	.005	Not always detectable
Plated wire or sense amp failed	Single bit failed	6.6	7.8/ 1.07**	Correct with Hamming-Parity code.
Mux or bit current switch open or short				
Control logic failures	Select wrong address	.005	.005	Detect and inhibit with parity code
	No response to access request	.01	.01	Processor timing check
	No output (whole word)	.06	.06	Detect with inv. Hamming-parity code
	Single bit failure	.06	.06/ NIL**	Correct with inv. Hamming-parity code
	Detectable garbled output	.06	.06	Detect with inv. Hamming-parity code
	Parity checker failure	.005	.005	Detect with Software
Total Failing Rate		11.6	12.8/6.0**	

*Number in () assumes quadded word drivers - not recommended due to excessive hardware involved.

**First number is probability of correctable failure, second number is probability of detectable but not correctable failure.

Memory Reliability Analysis

The dominant failure mode of Table XII can be masked by a single error correcting code. The second mode can be detected by such a code if the code bits are inverted prior to storage so that a code check on a word consisting of all "0" will fail. The third mode is the most serious because it can cause properly coded words to be written or read from the wrong location in memory undetected. It is caused by a stuck-in "1" condition in one of the hundreds of plated wire word line drivers. By employing series redundancy in these drivers, the conditional probability of occurrence of this condition can be reduced to a negligible .005. Series parallel redundancy (quadding) in these drivers will also eliminate the principal cause of the second failure mode. However, it is probably preferable to provide additional spare memories rather than to resort to quadded word drivers due to the large hardware increase involved in quadded word drivers.

The mean failure rate of a memory employing single error correction coding and serial redundant word drivers is half that of a memory without these features. What is more, undetectable failures make up less than 0.1% of the total failure modes yielding a coverage in excess of .999. Thus, use of triplicated memories in TMR processing stream does not seem to be justified and use of duplex memories for duplex or TMR stream and simplex memories for simplex streams is recommended to make most effective use of ARMMS hardware. Duplex operation is required when it is desirable to avoid program rollbacks in the event of a non-correctable memory failure.

In duplex operation the contents of the good memory can be written into a spare module or used in simplex for the duration of the program. In simplex operation it is essential to avoid writing bad data into the memory, or good data into the wrong location. The former condition can be protected against by immediate verification of all written data by reading out the same location immediately after writing into it in a simplex program. If the data is wrong the procedure can be repeated until the WRITE is accomplished successfully. The latter condition can be protected against by employing an address parity check code at the memory and inhibiting WRITE operations any time address parity is violated. Parity checkers can be provided in each memory at a small hardware cost.

Another result worth noting is that as the number of memory modules required goes up, the ratio of operating modules to required spares decreases, making the use of spares vs internal redundancy more attractive for larger numbers of modules required. A memory module incorporating a single error correcting code and series redundant word drivers should have a probability of surviving a 5-year mission of .869 (compared to .603 for a memory without these features). This means that if 5 modules are used there will be a .9884 probability that 1 will be operating after 5 years. If 10 modules are flown there will be a .9870 probability that 5 survive and if 25 modules are flown there will be a .9832 probability that 15 survive, and a .9945 probability that 14 will survive. This means that if 25 modules were flown allowing 12 duplex pages at the beginning of a mission, that there would be a better than .99 probability of having an operational simplex memory of the same size available at the end of the mission.

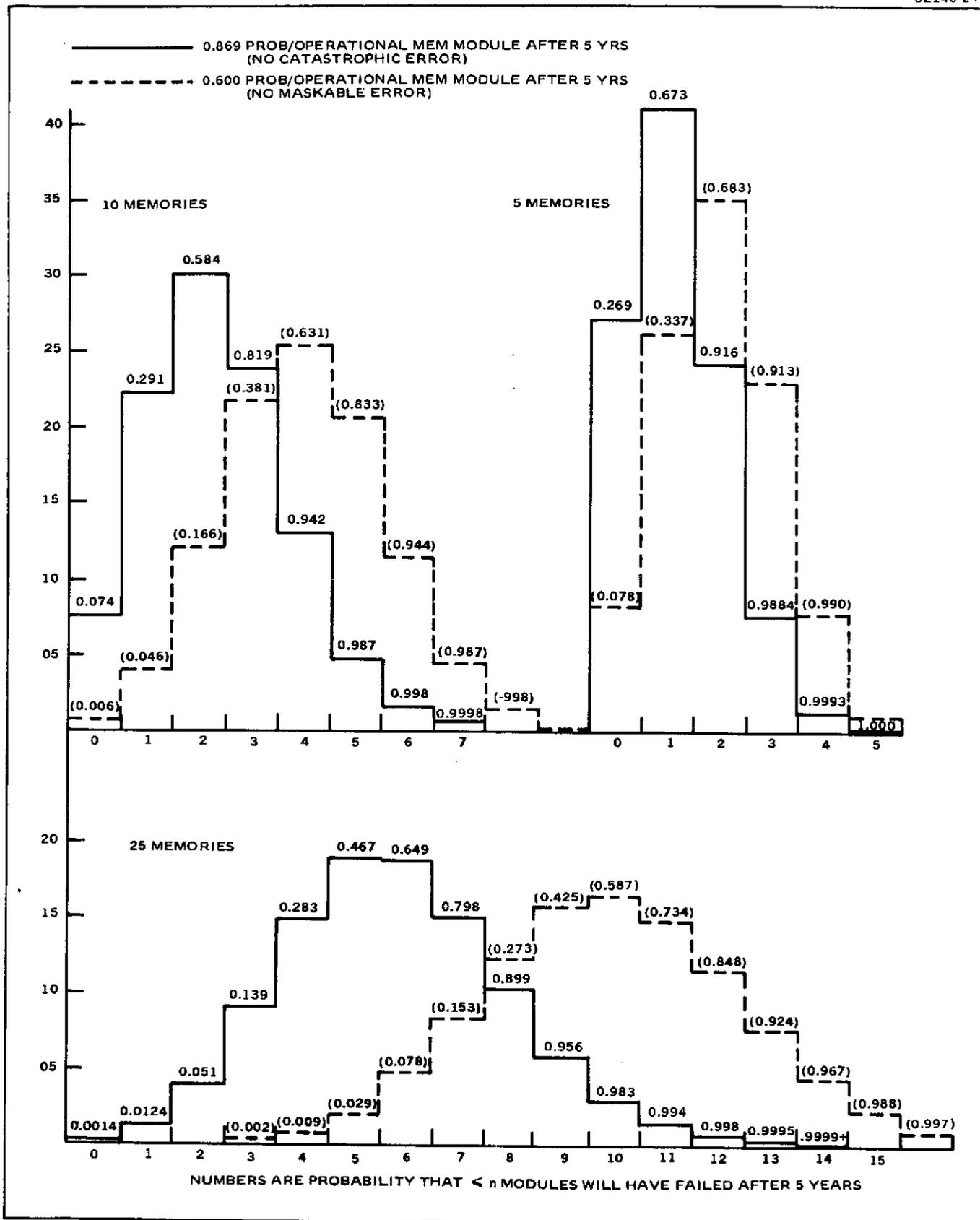


Figure 24. Effects of a Spare Pool on Memory Reliability. Probability Distribution of Memories Failing After 5 Years

Conversely, if fewer than 10 memory modules are flown this will probably not be possible. Figure 24 shows the probability distribution and the cumulative probability of various numbers of memories failing after a 5 year period. It should be noted that it is probable that several memory modules will fail during a 5 year mission, even with redundancy included, and a good system design must be able to cope with such failures gracefully. It is possible to detect or mask 99.9% of all memory failures in all ARMMS modes with the redundancy recommended at a power increase of 20% and a negligible weight and volume increase. Incorporating less redundancy than this would require TMR memory operation to achieve ARMMS system reliability goals and would thus require more weight, power and volume than the recommended approach.

SECTION 3

ARMMS CONTROL EXECUTIVE SYSTEM DESIGN

ARMMS software design of the control executive system was performed by Hughes' subcontractor M&S Computing, Huntsville, Alabama, concurrently with much of the hardware design described in the previous section. Hence some detailed discrepancies exist which will be reconciled during Phase III. None are considered to be severe problems at this time. This section consists of a discussion of software design philosophy, executive conceptual design including task control, event recognition and response, resource allocation and control, fault detection and diagnostic processing, information protection and input-output control. In addition a discussion of ARMMS impacts on the hardware SUMC processor is included. This SUMC study compliments but was independent of the one in Section 2.

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
LIST OF ABBREVIATIONS	3-iv
LIST OF FIGURES	3-vi
1. INTRODUCTION	3-1
1.1 Purpose	3-1
1.2 Project Task Summaries	3-1
1.3 ARMMS Baseline Configuration	3-2
2. CONTROL EXECUTIVE SYSTEM DESIGN OBJECTIVES	3-4
3. CONTROL EXECUTIVE SYSTEM SERVICES	3-6
3.1 Task Scheduling	3-7
3.2 Task Dispatching	3-9
3.3 Task Resource Allocation	3-10
3.4 Event Processing	3-13
3.4.1 Introduction	3-13
3.4.2 Event Definition	3-13
3.4.3 Alerted Events	3-14
3.4.4 Event Processing Commands	3-15
3.5 Fault Detection Progressing	3-16
3.5.1 Basic Concepts	3-16
3.5.2 Fault Categories	3-17
3.6 Information Protection	3-22
3.6.1 General Description	3-22
3.6.2 Storage Access Protection	3-22
3.6.3 Data Access Synchronization	3-23
3.7 Graceful Degradation	3-25
4. CONTROL EXECUTIVE CONCEPTUAL DESIGN	3-27
4.1 Definition of Terminology	3-27
4.2 Design Groundrules and Assumptions	3-29
4.3 System Design Overview	3-31
4.4 Task Control	3-35
4.4.1 Task Control Overview	3-35
4.4.2 Task Control Components	3-41

TABLE OF CONTENTS
(continued)

<u>Section</u>	<u>Page</u>	
4.5	Event Recognition and Response	3-65
4.5.1	Event Processing Overview	3-65
4.5.2	Event Processing Components	3-74
4.6	Resource Allocation and Control	3-87
4.6.1	Resource Control Overview	3-87
4.6.2	Stream Resource Allocation and Control	3-87
4.6.3	Diagnostic Resource Reservation	3-88
4.6.4	Memory Management	3-92
4.6.5	Locked Logical Pages	3-94
4.7	Fault Detection and Diagnostic Processing	3-107
4.7.1	Diagnostic Overview	3-107
4.7.2	Diagnostic Processing Components	3-108
4.8	Information Protection	3-128
4.8.1	General Description	3-128
4.8.2	Base/Bound Registers	3-128
4.8.3	Processor Architecture Implications	3-130
4.8.4	Shared Data Locks	3-131
4.8.5	Lock-Variable Contents	3-134
4.8.6	Subtask Accessing	3-138
4.9	Input/Output Control	3-139
4.9.1	Types of I/O Transmissions	3-139
4.9.2	Types of Input/Output Streams	3-139
4.9.3	Residency of Input/Output Control	3-141
4.9.4	I/O Summary	3-142
5.	MAJOR IMPACTS ON BASELINE SUMC	3-143
5.1	Processor Speed	3-143
5.2	Basic Instruction Set	3-144
5.3	SUMC/BOSS Communications	3-146
5.3.1	General Description	3-146
5.3.2	Hardware Modifications to SUMC	3-149
5.4	SUMC/BOSS Control Commands	3-153
5.4.1	Fetch Cycle	3-153
5.4.2	Task to Control Executive Commands	3-153
5.4.3	Control Executive to Processor Commands	3-155

TABLE OF CONTENTS
(continued)

<u>Section</u>		<u>Page</u>
5.5	SUMC/Memory Communications	3-163
	5.5.1 Communication Paths	3-163
	5.5.2 Error Conditions	3-163
	5.5.3 Hardware Modifications to SUMC	3-165
5.6	SUMC/I/O Communications	3-167
	5.6.1 I/O Commands	3-167
	5.6.2 Hardware Modifications to SUMC	3-169
5.7	Summary of SUMC Modifications	3-171

ABBREVIATIONS

ABEND -	Abnormal Ending or Terminating
ACES -	ARMMS Control Executive System
AFI -	Alert File Item
AFM -	Alert File Memory
ARMMS -	Automatically Reconfigurable Modular Multiprocessing System
BOSS -	Block Organizer and System Scheduler
BSW -	Bus Status Word
CPE -	Central Processing Element
CSRW -	Configuration Stream Request Word
DP -	Diagnostic Processor
FBSM -	File Block Status Matrix
FD -	Fault Detector
FM -	File Memory
FPS -	Full Processing Stream
I/O -	Input/Output
IOP -	Input/Output Processor
IOPS -	I/O Processing Stream
IP -	Input to (CPE) Processor (Bus)
LA -	Logical Address
LAAT -	Logical Address Assignment Table
LM -	Logical Module
LP -	Logical Page
LPS -	Limited Processing Stream
LSI -	Large Scale Integration
LU -	Logical Unit
MET -	Master Execution Table
MFW -	Module Fail Word
MI -	Memory Input (Bus)
MIC -	Memory Input (Bus from) CPE
MIP -	Memory Input (Bus from) IOP
MO -	Memory Output (Bus)
MOC -	Memory Output (Bus to) CPE
MOP -	Memory Output (Bus to) IOP
MSW -	Module Status Word

ABBREVIATIONS
(continued)

CB -	Output Bus (IOP to VS)
PO -	(CPE) Processor Output (Bus)
PSW -	Program Status Word
Q -	Queue - (Timer Queue or Priority Queue)
RPC -	Resource Pool Counters
TD -	Task Dictionary
TDIB -	Task Dictionary Information Block
TDIF -	TMR Dispatcher Inhibit Flag
TMR -	Triple Modular Redundancy
TQI -	Task Queue Item
TQM -	Task Queue Memory
TTE -	Time to Execute
UST -	Unit Status Table
VS -	Voter Switch
WF -	Weighting Factor
WFM -	Wait File Memory
WFP -	Wait File Pointer
WI -	Wait Item
WIQ -	Wait Item Queue

LIST OF FIGURES

<u>No.</u>	<u>Title</u>	<u>Page</u>
1-1	ARMMS Baseline Configuration	3-3
4-1	ACES Overview	3-32
4-2	Task Control Overview	3-36
4-3	Task Queue Item (TQI)	3-37
4-4	Master Execution Table (MET)	3-39
4-5	Task Scheduler	3-42
4-6	Schedule Task Call (Functional Definition)	3-45
4-7	Task Request Dictionary Entry (Functional Definition)	3-46
4-8	Timer Scheduler	3-49
4-9	Priority Scheduler	3-50
4-10	Dispatcher	3-52
4-11	Initiator	3-60
4-12	Timer Processor	3-62
4-13	Task Terminator	3-63
4-14	Event Processing Overview	3-67
4-15	Wait Call (Functional Definition)	3-68
4-16	Wait Item (Functional Definition)	3-69
4-17	Alert Calls (Functional Definition)	3-71
4-18	Alert File Item (Functional Definition)	3-72
4-19	Wait Call Processor	3-75
4-20	Wait File Processor	3-78
4-21	Wait Event Processor	3-79
4-22	Alert Call Processor	3-81
4-23	Alert File Scan	3-83
4-24	Alert Event Processor	3-85
4-25	System Wait Call	3-86
4-26	Reservation Processing	3-89
4-27	Reservation Calls	3-91
4-28	Configurator	3-96
4-29	Stream Search Subroutine	3-97
4-30	Memory Reconfigurator	3-100
4-31	Reservation Call Processor	3-102
4-32	Reservation Monitor	3-105
4-33	Fault Detector	3-109
4-34	Search Pageable Units	3-111
4-35	Page Routine	3-112
4-36	New Task Dictionary	3-114

LIST OF FIGURES
(continued)

<u>No.</u>	<u>Title</u>	<u>Page</u>
4-37	Memory Fail	3-115
4-38	Diagnostic Processor	3-117
4-39	Fail 1	3-120
4-40	Fail 2	3-123
4-41	Fail 3	3-125
4-42	Log Intermittent	3-127
4-43	Lock Variable Usage	3-132
4-44	Lock Request Logic	3-133
4-45	Unlock Request Logic	3-135
4-46	Lock Variable	3-136
4-47	Types of I/O Streams	3-140
5-1	SUMC/BOSS Communications	3-147
5-2	Interrupt Decoder	3-148
5-3	Processor to BOSS Communications	3-150
5-4	Memory/BOSS/Processor Communications	3-151
5-5	Hardware Modification to SUMC for BOSS Com- munications	3-152
5-6	Fetch Cycle	3-154
5-7	Call Instruction	3-156
5-8	Microprogram Abbreviations	3-157
5-9	Save to Main Memory	3-159
5-10	Stop Instruction	3-160
5-11	Restore from Main Memory	3-161
5-12	Start from BOSS	3-162
5-13	Processor/Memory Communication Paths	3-164
5-14	Memory/Processor Communications	3-166
5-15	Processor/I/O Communications	3-168
5-16	SUMC Modifications	3-170

1. INTRODUCTION

1.1 Purpose

The purpose of this contract was to conceptually design the ARMMS Control Executive System (ACES) functions pertinent to the reliability and reconfiguration aspects of ARMMS.

The design was preceded by a requirements analysis which resulted in various system concepts and design groundrules.

1.2 Project Task Summaries

The project was performed in a logical sequence of tasks.

Task I: Mission Analysis Profile

During this task historical data from previous airborne/spaceborne flights and future projections for such flights were analyzed to establish a baseline mission profile. This profile determined the range of computational and reliability capabilities required by the onboard computer during various modes of operation. In addition, it provided information concerning the basic operational characteristics of the application software necessary to perform the various missions.

The results of this task are documented in "Design of a Modular Digital Computer System DRL, Phase 1 Report (U)," Contract No. NAS8-27926, Hughes Aircraft Company.

Task II: Review of MSFC Processor

The MSFC processor (SUMC) is to be the basic processor module in ARMMS. The purpose of this task was therefore to review the SUMC architecture and recommend modifications necessary to support the necessary Control Executive functions.

During this task, however, it became exceedingly clear that the majority of the Control Executive functions had to be performed in an ultra-reliable mode to retain systems control under any and all failure conditions. Therefore most Control Executive functions were selected to reside in BOSS, the special-purpose controlling processor of ARMMS. This task therefore emphasized Executive Control System design concepts and groundrules necessary for BOSS. These are documented in Section 4 of this document.

Modifications to the SUMC identified during this task were mainly concerned with the support of BOSS-SUMC communications. These are documented in Section 5 of this document.

Task III: Preliminary Design of Control Executive Functions

Based on the results of the previous tasks a preliminary conceptual design was performed for the ARMMS Control Executive System (ACES). A design review was held for this design that resulted in various modifications and refinements of the proposed design.

Task IV: Final Design of Control Executive Functions

This task then completed the conceptual design. The design establishes a great degree of confidence in the feasibility of the basic objectives of ARMMS. This design provides the basis for a complete detailed functional design of ACES.

1.3 ARMMS Baseline Configuration

The systems configuration to which ACES was designed is characterized in Figure 1-1.

BOSS is a single ultra-reliable special-purpose processor providing central control of the system. Several copies of Memory Modules, Central Processors, and I/O Processors may exist in a particular ARMMS configuration. Any of these modules can be combined with other identical modules to run in a (triple or dual) redundancy mode, if so desired. Interconnections between modules are switchable, such that any logical combination of operational modules can be used to process a task.

An I/O Processor can be connected directly to Memory and thus be shared by several Central Processors. An I/O Processor can also be connected to a Central Processor and thus be dedicated to it, if so desired

ARMMS BASELINE CONFIGURATION

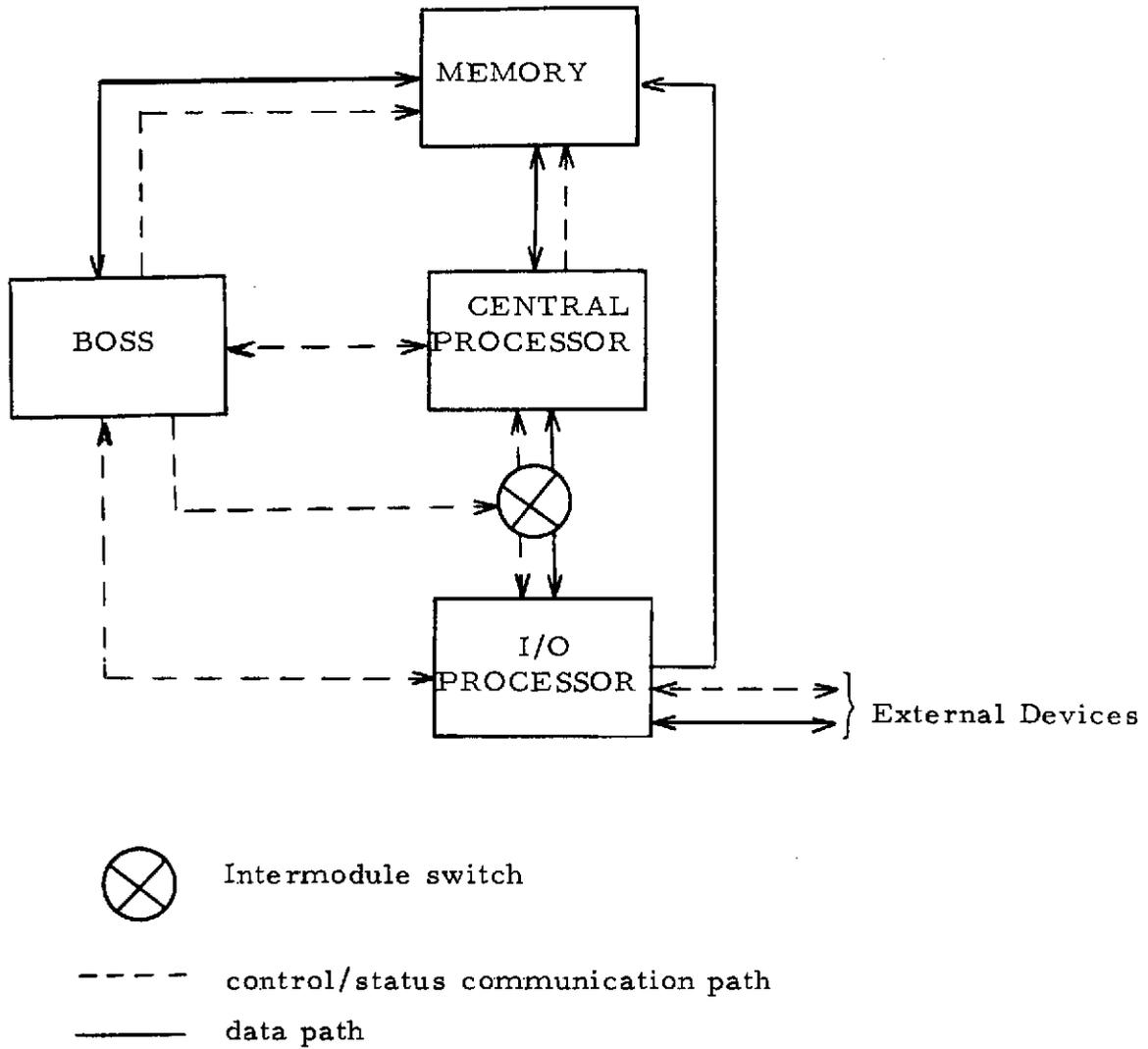


Figure 1-1

2. CONTROL EXECUTIVE SYSTEM DESIGN OBJECTIVES

A primary objective of ARMMS is to provide the ability to support a long life mission with a high probability of success. ARMMS can therefore, for example, be configured as a TMR System with standby spares for each module.

ACES, therefore, must first of all be able to react to error indications from the hardware, isolate a failing module, switch in a spare module, and allow the system to continue successfully. This has to be accomplished without any human assistance. ACES must further be able to allow the systems to degrade gracefully until the point that all of a particular type of module have failed. In addition, ACES must provide the application designers with as many aids as possible to prevent the propagation of software errors. That is, the effect of undetected software bugs must be contained within the software module containing the error. This may allow the system, in most instances, to continue its most critical functions regardless of software failures.

ARMMS can be selected to be configured as a high-performance system consisting of modules identical to those used in the high reliability mode described above. To accomplish this the system can be configured into a multiprocessing system.

ACES must therefore be able to schedule execution of programs on a varying number of independently operating modules. It must allow an application to be designed such that it can be divided in concurrently executing modules. It must not, however, force an application into a special design when multiprocessing is not necessary. Program modules, executing concurrently, must, of course, be prevented from interfering with each other's operation.

The primary types of applications, which ARMMS is anticipated to support, are real-time applications such as vehicle control, experiment control, etc. ACES is, therefore, primarily designed to support "process-control" type applications. This does not imply that "batch-processing" will not or cannot be performed. It implies that many support services characteristics of "batch-processing" (such as File Management) are not a standard service within ACES, but many "real-time control" services are. It is anticipated that, where batch-processing is required, that particular job and its support service routines are run as a single task under control of ACES. Batch-processing is thus considered incidental to the ACES design.

Finally, it is necessary to keep the ACES system as small and simple as possible. ACES directly influences BOSS and its interfaces with the ARMMS Modules. The complexity of BOSS and its interfaces directly affect the overall reliability and cost of the system. In addition, the Control Executive itself must not fail, for obvious reasons. It should therefore be possible to test ACES (nearly) exhaustively. The ACES design must, therefore, lend itself to a true modular design; that is, a design with simple interfaces between modules, resulting in a finite number of combinations of inputs and outputs for each module.

3. CONTROL EXECUTIVE SYSTEM SERVICES

This section describes the functions performed by ACES from the viewpoint of the systems user, such that the design of the functions provided in Section 4 will be more meaningful to the reader.

The functions described here are primarily executed by BOSS. They were selected for execution by BOSS on the basis that hardware failures during execution of these functions would cause loss of operational systems control. BOSS is the only module in the system that is assumed not to fail.

The ACES services are described in this section in slightly different categories than those used in the component design descriptions in Section 4. Once this section is understood, however, the correlation between ACES services and the design components is relatively easy to understand.

3.1 Task Scheduling

Key to the ability to multiprocess an application, is the ability to identify pieces of the application for concurrent processing on more than one processor. These pieces of an application are (in ACES as well as other systems) called Tasks.

Task Scheduling controls the potential start of execution of a Task and can be based on various conditions. The actual assignment of a task to a processor for active execution is called Task Dispatching and is described in Paragraph 3.2.

Task Scheduling is explicitly controlled through commands from the application programs. The commands available to the application programs are described below.

SCHEDULE (PARAMETERS)

This command requests the start of execution of a task based on the associated parameters. Two types of parameters can be associated with this command: Execution Type and Dispatch Conditions.

Execution Type specifies that a task is to be executed once, or that it is to be executed periodically at specific time intervals.

Dispatch Conditions specify the conditions to be met before a task is released for dispatching. Conditions can be time (at real-time or after timed interval) or events. The latter specifies that certain events have to be satisfied prior to dispatching. The exact definition and use of events is described in Paragraph 3.4. If time as well as events is specified as a dispatch condition, the time requirement will be satisfied before the status of the events is interrogated.

Dispatch Conditions associated with a periodically executing Task can only gate the first execution, subsequent executions can only be gated by the (periodic) time interval.

WAIT (PARAMETERS)

This command requests that execution of the Task, executing this command, is suspended unless/until the dispatch conditions specified as parameters are met. These dispatch conditions are identical to those listed under the SCHEDULE command.

TERMINATE

Obviously once a Task has started execution it requires a command to signal the end of its execution.

DELETE

This command causes any scheduling requests previously initiated to be negated. It is used, for example, to terminate scheduling of periodic tasks. It is also used under abort conditions or at any other time that currently scheduled tasks have to be prevented from being executed.

3.2 Task Dispatching

After the conditions (SCHEDULE PARAMETERS) described in the previous paragraph have been met, the task is ready to be executed. The function associated with putting the task into active execution is called Task Dispatching.

Task Dispatching, first of all, selects one task from all tasks that are ready for execution. This selection is based on a (dispatch) priority which has been pre-assigned by the user to a task. This priority is called the Dispatch Priority. This Dispatch Priority is selected by the user to obtain near optimum use of the system, considering task execution deadlines, task execution times, task precedence relations, etc.

Subsequently, Task Dispatching attempts to find the appropriate resources (reference Paragraph 3.3, Task Resource Allocation) for the execution of this selected task. If successful, the task execution is initiated.

Different tasks may have different resource requirements. Therefore if a Task cannot be executed because of unavailable resources, Task Dispatching will attempt to dispatch the next highest priority task.

As will be noted in the next paragraph, Task Resource Allocation, a Task may preempt resources from an actively executing Task, based on their assigned dispatch priorities. Once a Task is put into active execution, it is assigned an Execution Priority by ACES. The value of the Execution Priority is normally identical to the Dispatch Priority. Under certain conditions it is necessary to raise the Execution Priority to the extent that the resources of a Task will not be preempted by another Task. For example, it is not normally desirable to suspend an I/O Task (i. e., data transmissions). The Execution Priority cannot be controlled by the application, but is solely controlled by ACES.

3.3 Task Resource Allocation

To execute a Task various combinations of modules and buses are required. These combinations are called Processing Streams. Task Resource Allocation is concerned with keeping track of available resources, selecting available Processing Streams, or selecting available modules and connecting them into a Processing Stream, for task dispatching.

The types of Processing Streams required for different types of Tasks are the following:

Full Processing Stream:

This consists of a CPE, an IOP, and associated interconnecting buses.

Limited Processing Stream:

This consists of a CPE only and an interconnection with main memory.

Input/Output Stream:

This consists of an IOP only with a connecting bus to memory.

Note that any Processing Stream is connected to all of memory. Memory modules are shared resources and are not dedicated to a particular stream. Any stream, of course, can be TMR'ed, or Duplexed.

Any of the ARMMS modules are marked to be in any one of the states described below. These states are meaningful to Task Resource Allocation to perform various "levels" of resource searches described further on in this paragraph.

Fully Operational:

Such a module has no limitations as to use. It has two submodes: spare or active (i. e. actively being used).

Partially Operational:

Such a module has limitations as to its use (e. g., one of its parts is non-operational), but is not excluded from use. It has the same two submodes described above.

Non-Operational:

Such a module has failed to the extent that it cannot be used under any circumstances until it has been repaired.

Powered Down:

This is essentially a spare module which has been powered down until its use is mandatory. When powered up it may become either Fully Operational or Partially Operational, depending on its state prior to Power Down. Non-operational units can, of course, also be powered down.

Reserved:

Such a module is set aside, or to be set aside, for use by diagnostic routines. When a module is suspected of failure it is frequently necessary to run diagnostics to, positively, determine the existence and extent of the failure. To perform these diagnostics it is often necessary to use other operational modules. Those modules are thus "Reserved" for diagnostic processing. Operational modules may become reserved even though they are in active use. This means that they will not be reassigned to a processing stream after the current task has completed execution. It does not mean that the currently executing task will be suspended.

When a Task has been selected for dispatching a search is performed for the required resources. A table is kept where the available modules, their status, and existing bus connections are noted. This search is one of the more time-consuming functions performed by the Control Executive. To keep this search as short as possible, under normal conditions, several "levels" of searches have been defined.

First of all (level 1) a search is made for an existing (i. e., interconnected) spare processing stream with the appropriate configuration. If that search fails, a search (level 2) is made for spare and operational modules that can be interconnected. If this level 2 search is successful, the modules are interconnected and the task is initiated.

If the level 2 search fails, however, a level 3 search is made for spare modules that are either fully or partially operational. This search is more time-consuming than a level 2 search because of the complexities of interconnecting partially operational modules into a fully operational stream.

Only if the level 3 search fails, are the priorities of the executing tasks interrogated to determine if any of the modules can be preempted. This is only a last resort, because it is desirable to minimize forced suspensions of executing tasks and because the search is again a bit more complicated and thus more time-consuming than the previous levels.

3.4 Event Processing

3.4.1 Introduction

The real-time application of ARMMS emphasizes the controlled interaction of concurrently but asynchronously, progressing, processes. These processes include external processes (such as vehicle control and experiments) as well as processes internal to ARMMS (i. e., Tasks). These processes interact with each other through data and control signals. It is the control signals which are of interest here. Control signals include such things as lines into discrete registers or interrupt registers, program flags set by tasks, or any state change desired to be signalled.

ACES includes a unified control signalling concept, to provide a standard and centralized mechanism for use by the applications programmer. This mechanism is called Event Processing. It provides a simple, reliable and consistent mechanism to provide interprocess control communication.

3.4.2 Event Definition

Central to Event Processing is the definition of events. In ACES an event is an occurrence which:

- 1) The system has been designed to recognize.
- 2) The system has been requested to act upon.

An occurrence can be anything from an external signal, coming on, to the termination of execution of a task. Obviously, unless the software is designed to note the occurrence and potentially handle it as an event, the occurrence is meaningless to Event Processing. Additionally, unless the system has been requested (by a Task) to take some specific action as the result of the occurrence, the occurrence will appear and disappear without any effect, and is therefore meaningless to the system, and thus not an event.

Occurrences which may be used as events by the applications programmer are:

- o Task Termination
- o External (to ARMMS) attention requests

- o Input/Output Completion
- o Setting/Resetting of special events called Program Flags
- o Detected Software Faults (reference Paragraph 3.5).

Other events have been defined for use internal to ACES. These are described in Section 4.

An Event's status is normally unsatisfied (off). When ACES recognizes an occurrence it considers the associated Event satisfied for as long as it is necessary to perform the actions requested (by a Task). Thus, actions specified after an occurrence has been recognized will not be executed until an occurrence is recognized again. An Event is therefore a pulse and is never explicitly reset by a Task.

An Event can be created (i. e., an action specified upon recognition of an occurrence) by a Task through a SCHEDULE, WAIT, or ALERT call. The SCHEDULE and WAIT calls were previously discussed in Paragraph 3.1. The ALERT call is discussed in the following paragraph.

3.4.3 Alerted Events

Note that the SCHEDULE/WAIT upon event, depends on the recognition of an occurrence from that point on. It is sometimes desirable to recognize an occurrence during a time period prior the point that it is necessary to perform the SCHEDULE/WAIT call. It is also true that sometimes different tasks are interested in recognizing occurrences during different but overlapping time periods. To allow the Tasks to accomplish this, the concept of the Alerted Event is introduced. An Alerted Event is simply equivalent to a "flag" that is set on when the associated event is satisfied and remains on. Alerted Events are created by an ALERT call. An ALERT also creates an event: it specifies that an Alerted Event is to be set when the underlying occurrence is recognized. Once an Alerted Event has been set, it is not reset until a new ALERT is issued. Note that multiple Alerted Events can be associated with a single Event. Parameters for SCHEDULE and WAIT calls can include Alerted Events as well as Events.

3.4.4 Event Processing Commands

The total set of commands available to the Task to control Event Processing includes the SCHEDULE/WAIT, and ALERT calls as well as the commands summarized below.

CANCEL ALERTED EVENT

This deletes the identified Alerted Event from further participation.

TEST ALERTED EVENT

This provides a test of an Alerted Event status to base program decision logic upon.

SET/RESET EVENT

This allows the Task to stimulate a specific class of Events called Program Flags to perform intertask coordination.

Any occurrence may in fact be simulated by SET Event. This is not normally allowed, however, to be used by the application programmer. The application programmer's main intertask coordination mechanism is the Program Flag.

3.5 Fault Detection Processing

A significant part of the ARMMS design deals with fault tolerant aspects of uni- and multiprocessing. The following explains the basic design philosophy used in ACES, to support a high degree of fault-tolerance.

3.5.1 Basic Concepts

ACES uses two broad, general classifications into which computer faults can be divided: hardware faults and software faults.

Hardware faults are defined as faults which are the result of component failures. These component failures may be due to fatigue, broken wires, insufficient power, etc. Any particular component of a system, even though properly designed, manufactured, and installed, may fail.

Software faults are all other faults which are not clearly caused by component failures. Typical examples includes divide overflow, addressing error, non-existent operation code, etc. Past experience has shown that even thorough testing of a complex program will not completely eliminate all program faults. These bugs may result either because the programmer failed to comprehend the full magnitude of the program and improperly designed it, or because the programmer made a mistake in the actual coding of the program even though it was properly designed.

Software faults are initially assumed to be the result of these types of software bugs. However, a hardware failure could yield a fault which at first appears to be due to a software bug. It is for this reason that as part of the software fault recovery procedures, module self test-diagnostic routines are executed. If the fault is found to be due to a hardware failure, the self-test diagnostics will be responsible for communicating this to BOSS (through ACES).

In ARMMS, all faults, hardware and software, are communicated to BOSS via the Module Status Word (MSW). Each module contains an MSW which indicates the status of that module. A section of the MSW contains information concerning the faults detected by the module. This section of the MSW is divided into two subsections, one for hardware faults and the other for software faults.

It is expected that whenever a hardware fault occurs, a bit is set in the hardware subsection of the MSW (by the hardware), and if the bit was not previously set, an interrupt notification is sent to BOSS along with the new MSW. For these types of faults, BOSS (through ACES) will be responsible to perform fault diagnostics and to take corrective action.

Whenever a software fault occurs, it is expected that a bit is set in the software subsection of the MSW to indicate the type of software fault. The CPE (or IOP) will then execute a predefined instruction sequence to process the software fault. A task will have the capability to mask any software faults such that they are not processed. (Hardware faults cannot be masked.) The software fault processing will vary dependent upon the type of error. However, it is anticipated that most processing will result in abnormally terminating the current task utilizing the processor, and executing a simple processor self-test program to insure the processor hardware is not at fault.

BOSS periodically will poll each MSW and determine if it has been updated since its last polling. It is by this means that BOSS has knowledge of the software error. However, BOSS does not actively participate in the fault processing of software faults unless a particular module continues to experience abnormal terminations of tasks, or a self-test indicates a component failure.

3.5.2 Fault Categories

Tables 3-1 through 3-4 define the fault types applicable to each module class. The fault types are defined by an example, definition, etc. The module which is expected to process each type of fault is identified along with the possible sources of the faults. These fault classifications are preliminary and may be modified as future requirements dictate.

CPE FAULT CLASSIFICATIONS

UNIT	FAULT TYPE	REASON, EXAMPLE, DEFINITION	FAULT HANDLER	POSSIBLE FAULT SOURCES
CPE	Illegal Operation	1) Non-Existent Operation Code 2) Illegal Register Designation 3) Addressing Error - (Displacement Out - of-Range) 4) Privileged Instructions 5) Boundary Alignment	CPE	CPE, Programming
	Data Error	1) Overflow 2) Underflow 3) Improper Divide 4) Sum Check	CPE	CPE, Programming
	Input Error	1) Parity 2) Correction Codes	BOSS	Bus, Memory, IOP, Error Detection Logic in CPE
	Communication	1) Timers Time-Out 2) Illegal IOP Busy	BOSS	Bus, Memory IOP, Timers, Logical Addr. Not Available

3-18

Table 3-1

IOP FAULT CLASSIFICATIONS

UNIT	FAULT TYPE	REASON, EXAMPLE, DEFINITION	FAULT HANDLER	POSSIBLE FAULT SOURCES
IOP	Illegal Operation	1) Non-Existent I/O Operation Code 2) Addressing Error (Out-of-Range) 3) Privileged Instructions	IOP	Programming, IOP
	Input Error	1) Parity	BOSS	Bus, Memory, CPE, Error Detection Logic in IOP
	Communication	1) Timers Time-Out	BOSS	Bus, Memory, CPE, Timers, Logical Address Not Available

Table 3-2

MEMORY FAULT CLASSIFICATIONS

UNIT	FAULT TYPE	REASON, EXAMPLE, DEFINITION	FAULT HANDLER	POSSIBLE FAULT SOURCES
MEMORY	Input	1) Parity 2) Correction Codes	BOSS	Bus, CPE, IOP, Internal Fault Detection Logic
	Memory Read	1) Error Correction Code Detected for Memory Read	BOSS	Memory
	Protection Violation	1) Read Attempt 2) Write Attempt	Requesting Processor	Programming, Memory Addr. Decode Logic, Invalid Key, Read/Write Locks

3-20

Table 3-3

VOTER/SWITCH FAULT CLASSIFICATIONS

UNIT	FAULT TYPE	REASON, EXAMPLE, DEFINITION	FAULT HANDLER	POSSIBLE FAULT SOURCES
Voter/ Switch	Voter Non-Compare	1) Input Disagreed With Other Input(s)	BOSS	Voter, Switch, Memory, CPE, IOP, Bus
	Illegal Configuration Command	1) Not Legal Configuration From BOSS	BOSS	BOSS Software, BOSS Firmware, BOSS Hardware, Switch, Bus
	Input	1) Parity From BOSS	BOSS	BOSS, Detection Logic, Bus

Table 3-4

3.6 Information Protection

3.6.1 General Description

Information Protection is concerned with the protection of information, used by a Task (i. e. , its instructions and accessible data), against inadvertent modification by other Tasks.

A software or hardware failure may cause a task to address the wrong storage location. Protection against this is called storage access protection.

Concurrently executing Tasks, accessing a common data base, may interfere with each other's proper execution in, for example, the following manner. Assume that Task A uses input data, which is computed and updated, at appropriate times, by Task B. There is not necessarily a sequential dependency between the two tasks. Task B may, for example, be executed more often than Task A. These Tasks will obviously execute properly without any sequential dependencies, as long as the Task A input data is consistent whenever Task A uses that data. Thus, Task B must be prevented from updating the data whenever Task A is actively using it to prevent Task A from using data that is partially updated and therefore not consistent. Protection against this type of interference is called Data Access Synchronization.

The mechanisms designed into ACES to provide information protection are described below.

3.6.2 Storage Access Protection

This is a mandatory requirement in a fault tolerant system such as ARMMS. Without it, a single software/hardware failure during task execution could cause total loss of system control.

Three basic schemes exist to accomplish Storage Access Protection: Segmenting, Storage Protect Keys, and Base-Bound Registers.

Segmenting is easily the most flexible and elegant solution. It was implemented in full glory in MIT's Multics System. The sole drawback is that information accesses frequently require multiple storage accesses. This would prohibit ARMMS from meeting its performance objectives and was therefore discarded.

Storage Protect Keys are, for example, used in the IBM 360/370 systems. The main drawback to the scheme is that the protected storage block has to have a predetermined fixed size and has to start on fixed boundaries. The advantages are that it does not affect performance, and is relatively simple and inexpensive to implement. Although discarded for ACES, it is a valid backup in case the selected ACES scheme turns out to be impractical for one reason or the other.

Base-Bound Registers allow the protected storage blocks to vary in size, allowing storage allocation to be more efficient than with storage protect keys. Although less elegant, and less of a unified concept than Segmenting, it does not have the significant performance disadvantages. Its disadvantage is that it only detects failures up to the Storage Address Register. Any hardware failures, occurring beyond that point, must be detected in a different manner. Storage Protect keys, on the other hand, are checked at the memory module during memory access and therefore do check the complete path to the memory modules. ARMMS does provide an alternate method to check the Storage Address Register - Memory Module path (TMR or Duplex), therefore this disadvantage is negligible in ARMMS. Base-Bound Register protection was therefore the compromise solution selected for ARMMS. It is described in detail in Section 4 .

3.6.3 Data Access Synchronization

As briefly described in Paragraph 3.6.1, a basic mechanism is required to enable Tasks to obtain exclusive use of common sets of data.

The most common, current, mechanism provided is the "Test and Set" instruction. This instruction allows a flag to be tested and to be set on, if it was off. This needs to be done within one instruction to insure that no Task can access the flag between testing of the flag and setting of the flag by another Task. Each set of data, which needs access synchronization, can thus be assigned a flag. Any Task requiring exclusive use, performs the "Test and Set" and gains control of the data, or waits until it can gain control. The mechanism is simple and inexpensive but has some basic shortcomings. First of all, proper operation is totally dependent on the programmer. It is not detectable, at execution time, whether the programmer has properly protected the data before accessing it. There is also significant opportunity to cause deadlocks. Both these problems can be largely overcome, however, by static checking during compilation/assembly time. A more significant disadvantage is, that multiple Tasks

cannot be allowed simultaneous read access to a set of data. That is, a Task which sets the flag to be protected from other Tasks writing into the data, locks out other Tasks from reading the protected data as well.

In most batch processing and time sharing systems the limitations discussed above are not of major magnitude. In a system such as ARMMS, the real-time multitasking and multiprocessing characteristics are such that the access of common data is far more common, and it is far more critical that synchronization is assured and performed efficiently.

The ARMMS Data Access Synchronization scheme therefore discards the Test and Set Instructions for a scheme based on the following groundrules:

1. All common data is under access protection of the Control Executive. That is, a Task cannot access a set of common data before it has explicitly requested access to it from the Central Executive.
2. The Access request has to specify explicitly the type of lock needed. A Read Lock will prevent other Tasks from writing into this set of data until the lock has been removed, but will not prevent other Tasks from concurrently reading it. A Write-Lock will prevent other Tasks from reading the data concurrently and may (system option) prevent other Tasks from writing into the data concurrently.
3. To prevent potential deadlocks, a Task must request all its data-locks at the same time. No additional locks may be requested until the existing ones have been removed.

As it later described in Section 4 , Data Access Synchronization makes use of the existing Base-Bound scheme defined for storage protection in Paragraph 3.5. It thus requires very little additional logic over the basic Storage Access Protection available.

3.7 Graceful Degradation

An integral part of any fault-tolerant system is graceful degradation. That is the system must be able to experience loss of modules without losing control of the system.

Thus, for ARMMS, the objective is that the system must be able to continue operation as long as there is one module of each kind; that is, one CPE, one IOP, and one Memory Module. BOSS is assumed to be fully operational at all times.

There are two main facets to graceful degradation. First of all, the system must be independent of which modules fail. That is, scheduling and dispatching algorithms must be independent of the number and types of modules usable in the system. Secondly, some means must be provided to adjust the load of the system to the loss of processing power.

Loss of CPEs, IOPs, and buses is easily handled during dispatching. ACES performs its resource allocation, during dispatching, based on a table of available resources. The numbers and types of resources noted as available have no effect on the normal operation of the dispatcher.

Loss of memory modules (i. e. directly addressable storage) is handled through a simple paging scheme. Page size for ACES is equivalent to the size of a memory module. When an addressing exception occurs, the required page is brought from back-up storage (drum or disk) and swapped with the contents of one of the operational memory modules.

To adjust the load on a degraded system it is first of all necessary to detect that the system has degraded to a point where it cannot process its normal load.

In this preliminary design there are two points at which this can be detected. When the system during diagnostic processing has to remove a failed module from further participation, it can scan the total set of available modules, and evaluate this against a predetermined set of criteria. Also when the system tries to allocate resources prior to dispatching, it may detect conditions that signal a potential overload due to excessive degradation.

Once it has been detected that the system has lost an excessive percentage of its performance, the processing load has to be readjusted. At first glance it would seem that this can be accomplished by interrogating

the available Task priorities. However, the normally used Task priorities are dispatch priorities established to obtain efficient performance and meet required deadlines. A Task that has a low dispatch priority may be more critical (i. e., necessary) to a mission than a Task with a higher dispatch priority.

An approach to this problem may be to assign a second priority to each Task, the mission priority. During degraded operation, the dispatcher could then be influenced by the mission priorities. This scheme has two main disadvantages. First of all the Control Executive would rapidly run out of storage necessary to keep track of the Tasks waiting for dispatching. Secondly, a significant amount of Control Executive overhead is used to handle Tasks that may never go into execution.

The solution used in ACES is to allow the user to predefine several different Task loads for several levels of degradation. When a certain level of degradation is reached, ACES will access the (user-provided) corresponding Task load definition and reload the system, or flush non-critical tasks from the system as appropriate.

4. CONTROL EXECUTIVE CONCEPTUAL DESIGN

4.1 Definition of Terminology

Prior to describing the ARMMS Control Executive System (ACES), a number of terms used in the remainder of this report must be defined. These definitions are listed below in alphabetical order. Other definitions are provided in the appropriate paragraphs.

- 1) Configuration Wait - An executing task is placed in a configuration wait when the resources utilized by that task are required by a higher priority task, or when the diagnostic software/hardware has identified a failure in a stream component.

- 2) Criticality - Criticality is equivalent to the required mode of operation; TMR, duplex, or simplex. The terms "stream weight" or "stream weighting factor" are often used as a measure of criticality where a stream weight or weighting factor of three (3) is equivalent to TMR operation, a stream weight of two (2) is equivalent to duplex operation, and a stream weight of one (1) is equivalent to simplex operation.

Criticality is considered to be a characteristic of both streams and individual tasks.

- 3) Dictionary Period - The period of time during which a given task dictionary is in effect.

- 4) Logical Memory Module - A logical memory module is defined to be a set of physical memory modules with a common logical memory address. A logical memory module can be a triad with a criticality of TMR, a pair with a criticality of duplex, or a single module with a criticality of simplex.

- 5) Module - A module is a specific ARMMS module such as a memory module, a CPE module, or an IOP module. Buses are not considered to be modules.

- 6) Module Number - A module number or address is an identifier defining a particular module port. It is likely that multi-port modules will have multiple module numbers.

- 7) Overload Condition - An overload condition exists when the total number of components is sufficient to build the desired stream but (due to one or more higher priority tasks currently utilizing some or all of the operational components) the necessary resources are not currently available to build the desired stream. The overload condition is temporary and is synonymous with scheduling overload and not to be confused with the, much more critical, processing overload.
- 8) Port - A module port is the capability of a device to support and control an input and an output channel. A four-port module would therefore be capable of supporting four independent input and output channels.
- 9) Process - A process is the total set of operations required to perform a defined application effort, excluding those operations performed by the control system. The execution of a process is normally implemented as a set of related programs responsible for performing the process operations. A process can be implemented as a single task or as a set of related tasks.
- 10) Processing Overload - A processing overload exists when the available resources are insufficient to process the tasks within a required time period. It is a permanent condition caused by failures of the modules.
- 11) Resource Problem - A resource problem exists when a new task is potentially the highest priority executing task, and it cannot be put into execution on existing operational resources. In other words, if all processing streams were halted and all operational components were made available, the total number of necessary components would be insufficient to build the desired stream.
- 12) Stream - A stream is the interconnection of devices required to support the execution of a task. A stream consists of the IOPs, CPEs, buses, output switches, and voter configurations required to support task execution.

A stream's criticality can be TMR, duplex or simplex, and three types of streams will be supported by ARMMS:

- a) A Full Processing Stream consisting of CPEs, IOPs, and associated buses with the IOPs connected to the CPEs.
- b) A Limited Processing Stream consisting of CPEs and associated buses (no IOPs).
- c) An I/O Stream consisting of IOPs and associated buses with no CPEs.

Memory modules are not members of a stream. Memory modules are shared devices and are not dedicated to any specific stream.

- 13) Task - A task is a unit of work. That is, a set of instructions, data and control information capable of being executed by a single Central Processing Element (CPE) and/or by an Input/Output Processor (IOP). It can be a program to be executed on a processing stream or a block of data to be received or transmitted via an I/O stream.
- 14) Task Dictionary of Lower Level - A task dictionary of lower level is a task dictionary dependent on fewer resource components for successful and timely execution than the current task dictionary. (See Section 4.3 for Task Dictionary.)

4.2 Design Groundrules and Assumptions

To bound the operational characteristics of the ARMMS Control Executive System (ACES), a number of basic design groundrules were established. Also, a number of design assumptions were made based on engineering judgement and experience concerning desired system capabilities and services. All of these groundrules and assumptions were reviewed to ensure that no serious design incompatibilities exist between the current software and hardware design concepts.

The following paragraphs define the groundrules and assumptions which governed the definition of the ACES philosophy.

- 1) The TMR full processing stream consists of three IOPs, three CPEs, and a logical memory module of multiples of three physical memory modules. A TMR Logical Memory Module (LMM) is called a TMR triad. The TMR limited processing stream or I/O stream configuration is assumed to consist of three CPEs and a TMR LMM, or three IOPs and a TMR LMM, respectively.
- 2) The duplex full processing stream configuration consists of two IOPs, two CPEs and an LMM made up of multiples of two physical memory modules (a duplex pair). The duplex limited processing stream or I/O stream configuration consists of two CPEs and a duplex LMM, or two IOPs and a duplex LMM, respectively.
- 3) The simplex full processing stream configuration consists of an IOP, a CPE, and an LMM consisting of one or more single physical memory modules. The simplex limited processing stream or I/O stream consists of a CPE and a simplex LMM, or an IOP and an LMM, respectively.
- 4) Stream to memory interconnection is task independent. That is, memory is not a dedicated resource. The ACES configurator will attempt to connect all defined streams to all defined LMMs with a criticality or stream weight equal to or greater than that of the defined stream.
- 5) The Dispatcher will not impose any limitations on the number of simultaneously executing streams. The Dispatcher will attempt to build as many simultaneous streams as it can use so long as resources are available.
- 6) A real-time clock and interval timer(s) are available to BOSS and to the individual CPEs. Any such clocks and timers will be accurate to at least 100 μ s.
- 7) Certain error conditions within streams will result in alerts (interrupts) to BOSS. Such failures will automatically halt the failing stream (all modes except TMR) until BOSS takes corrective action. TMR streams will continue to execute until task completion regardless of the occurrence of failures, unless two or more failure locations are identified.

- 8) The memory input and output buses will be multiplexed such that an IOP and a CPE can be simultaneously connected to the same memory buses. However, multiple CPEs or multiple IOPs cannot be connected simultaneously to the same memory buses. Also, multiplexed IOP buses may be treated as completely independent of multiplexed CPE buses.
- 9) The ACES scheduling algorithms are based on asynchronous executing concepts in which the Executive software is primarily concerned with dispatching application program tasks as a function of task priority and time to initiate execution.
- 10) All streams will be constructed from available resources maintained in a centralized resource pool. Failing streams will be disassembled and all operational components will be returned to the resource pool. Terminating streams will remain connected until individual components of that stream are required for a different type stream.

4.3 System Design Overview

This subsection gives an overview of the ARMMS Control Executive System functions, as indicated in Figure 4-1 which presents a simplified functional overview of ACES.

The Task Dictionary is a list of all possible task requests for a given phase of the mission. In addition, the Task Dictionary contains, for each task request, a definition of static execution parameters of concern to ACES. A separate Task Dictionary is loaded into BOSS memory for each mission phase, or whenever resource failures dictate the need to limit processing to a degraded set of mission tasks.

Task Dictionary entries are utilized by the Task Scheduler to build a Task Queue Item (TQI) for each task execution request which contains the execution control parameters of the request.

The Task Scheduler accepts requests for task execution from currently executing processing streams. The request parameters from the request call are combined with the static parameters from the Task Dictionary by the Task Scheduler which builds the TQI for that request. The TQI is placed either in the Timer Queue or in the Pending Task Queues depending on the request parameters.

ACES OVER VIEW

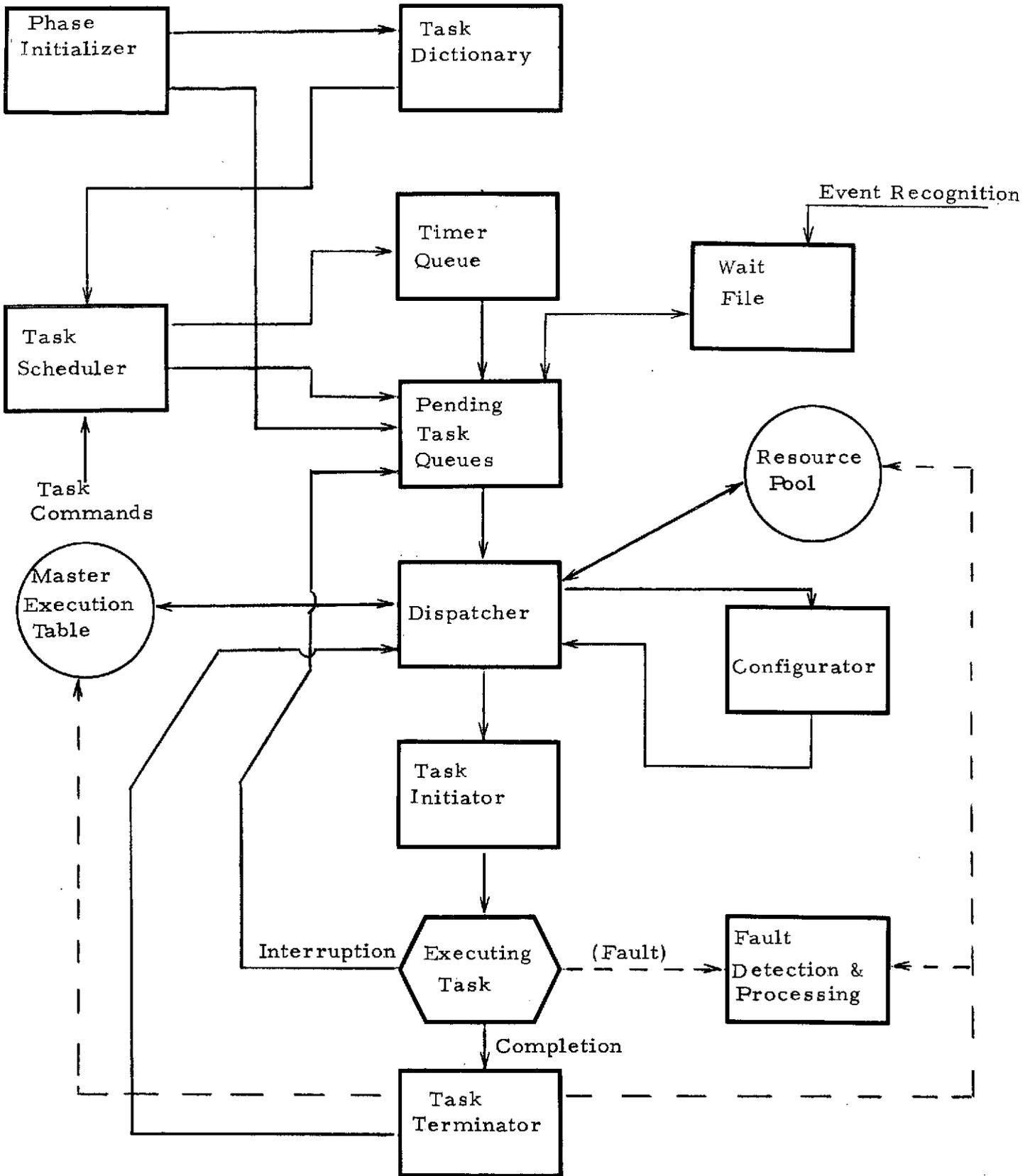


Figure 4-1

The Timer Queue contains a list of all TQI requests where task execution is not to commence prior to some specific future time.

Once the time requirement has expired, or if no time requirement was specified in the call, a task's TQI is placed in the Pending Task Queues. There are as many Pending Task Queues as there are defined levels of task priority and, within each Pending Task Queue, TQIs are treated by the Dispatcher on a First-In, First-Out (FIFO) basis. TQIs within the Pending Task Queues may have specified events which must occur before they can be eligible for execution. The definition of such events resides in the Wait File and the TQI is marked waiting in the Pending Task Queues.

The Dispatcher is responsible for sequencing the execution of tasks eligible for execution; i. e., those tasks having requests in the Pending Task Queues which are not waiting for an event to occur. The Dispatcher always attempts to put the task with the oldest, highest priority task request into execution next.

If sufficient resources are available in the resource pool to build a new stream with the task criticality required by the TQI, the stream is identified and established in the Master Execution Table (MET), the Dispatcher uses the Configurator to interconnect the stream components, and the task is initiated by the Task Initiator.

If sufficient resources are not available in the Resource Pool to build a new stream as required, the Dispatcher determines whether a resource problem exists or whether the system is just in a temporary overload condition. A resource problem exists when a new task is potentially the highest priority executing task and it cannot be put into execution on existing operational resources. Such a situation either requires the Dispatcher to modify the criticality of the requested task so that it can make use of available resources, or that a new Task Dictionary be brought into the system which is tailored for the degraded configuration of operational resources.

If an overload condition exists (i. e., sufficient resources are operational to handle the current task request criticality but higher priority tasks are now executing), the current task request will be left in the Pending Task Queues and no further attempt will be made to dispatch it until sufficient resources are available and it is the highest priority pending task request.

Once a task request is dispatched (that is, sufficient resources have been identified to support its execution and a stream has been reserved in the MET and wired by the Configurator), the Task Initiator is given control.

The Task Initiator is responsible for initiating task execution on the prepared stream hardware. Three types of streams are to be supported by ACES: full processing, limited processing, and I/O streams. These types of processing streams can either be entering execution for the first time or resuming an execution which was previously interrupted by a higher priority task or relinquishing control until a specific event(s) occurs. In any event, the Task Initiator must be able to distinguish between a start operation and a restart since different initiation procedures are required. Similarly, the initiation procedures for an I/O stream are different from those required by a processing stream.

Once a task is executing on a stream, it will run until completion, or until the task places itself in the wait state, or a fault is detected by ACES, or the task is interrupted by the Dispatcher which requires its stream resources for a higher priority task.

An executing task interrupted by the Dispatcher is said to be in a configuration wait and its TQI remains in the Pending Task Queue. It is eligible for restart under the normal dispatching algorithm; that is, as soon as resources become available and it is the highest priority pending task request.

An executing task interrupted by the detection of a fault also is placed in a configuration wait. However, in this situation, certain resources may be reserved by the Fault Detection and Processing software in order to isolate the cause of the fault and remove the failing module from the resource pool.

If an executing task runs to completion and terminates properly, the Task Terminator will free the available resources, update the Resource Pool (unless the stream can be used in its present configuration), update the MET, and return control to the Dispatcher. This allows the Dispatcher to make immediate use of the available resources if task requests are currently stacked in the Pending Task Queues.

The TQI associated with a task that requests to be placed in an event-dependent wait state is marked waiting in the Priority Execution Queues. When the event(s) that the task is waiting upon occurs, the TQI's wait restriction is removed and it is eligible for restart under the normal dispatching algorithms.

4.4 Task Control

4.4.1 Task Control Overview

Task control consists of the algorithms and design concepts required to schedule, dispatch, initiate, interrupt, and terminate application program tasks. Figure 4-2 presents an overview of the task control components and defines their cross communication linkages at the functional level.

BOSS memory as presented in Figure 4-2 is the central communication area for controlling all task control operations. Task commands accepted by the Task Scheduler result in the definition of a TQI which is placed in Task Queue Memory. The TQI is linked into either the Timer Queue or one of the Priority Execution Queues (pending task queues), depending upon whether time parameters are present. Figure 4-3 defines the functional components of the TQI.

The key parameters that determine the queue the TQI initially enters is the request type parameter. The request type specifies that this request is either a priority request or a time request with an associated time-to-execute parameter.

Any TQIs with a request type specifying time, and an unexpired time-to-execute parameter will be attached to the Timer Queue by the Timer Scheduler. All other TQIs will be attached to the Priority Execution Queues. The Priority Execution Queues contain TQIs which have specific events (wait items) which must occur prior to making the TQI eligible for execution as well as TQIs which are eligible for immediate execution.

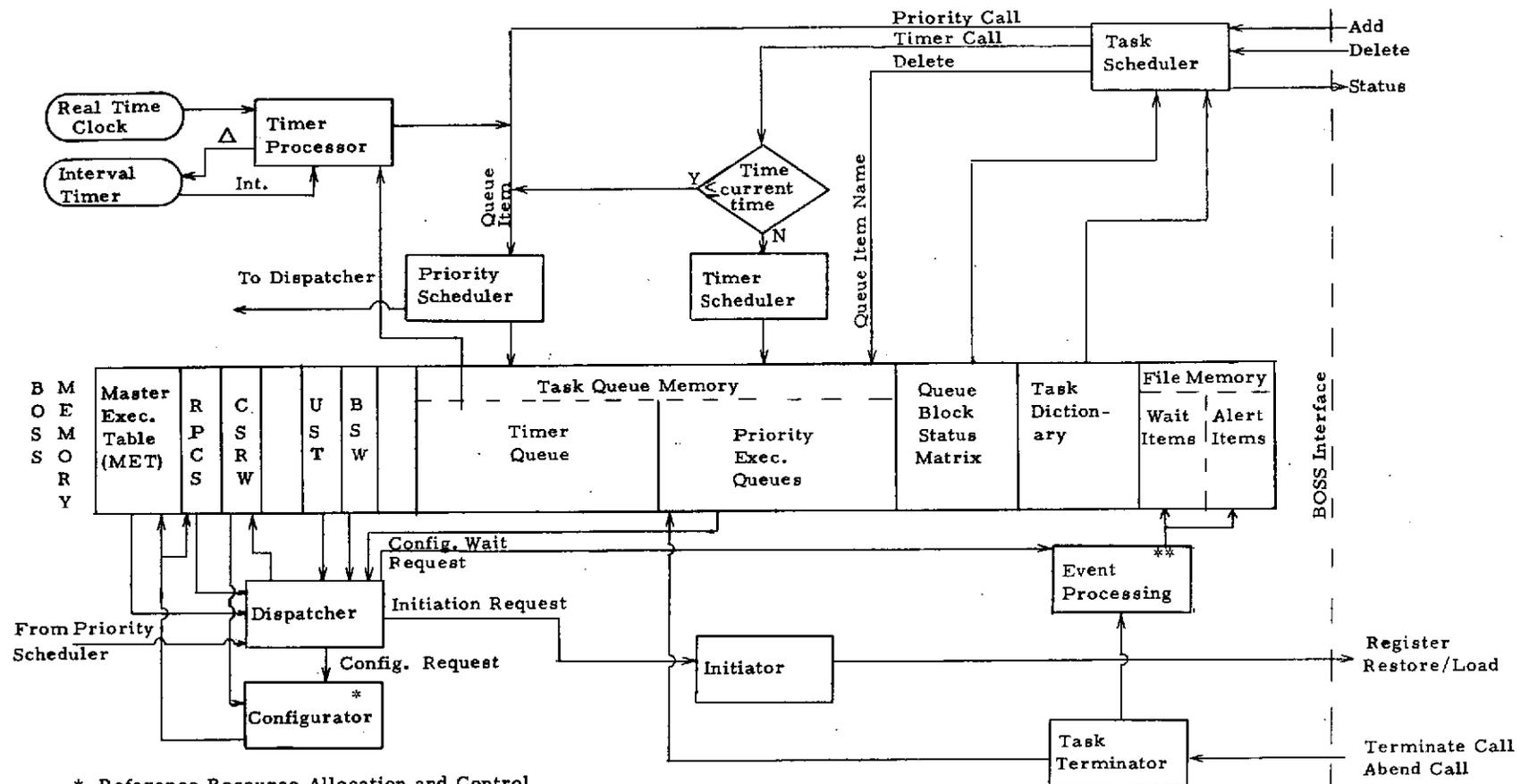
TQIs resident in the Timer Queue, are ordered according to time-to-execute and are processed by the Timer Processor. The Timer Processor accumulates phase and/or mission real time and moves TQIs from the Timer Queue to the Priority Execution Queues once the time-to-execute parameter of the TQI has been satisfied.

A different Priority Execution Queue exists for each defined task dispatch priority level. TQIs resident in the Priority Execution Queues are stacked in the order in which they are received. A TQI remains in its Priority Execution Queue until its task either terminates or until the queues are flushed by the system during a phase change.

Since a TQI can functionally move from queue to queue, considerable system overhead would be experienced if TQIs were required

TASK CONTROL OVERVIEW

93-3



* Reference Resource Allocation and Control
 ** Reference Event Recognition and Response

Figure 4-2

TASK QUEUE ITEM (TQI)

Task Q Item Number	Request Type*	Request Dispatch Priority	Task Criticality (S, D, T)	Stream Type	Time to Execute	Pointer to Parameter List	Status Flags	Pointer to Next Q Item	Task Name (Number)	Pointer to Wait Save Area
--------------------------	------------------	---------------------------------	----------------------------------	----------------	--------------------	---------------------------------	-----------------	------------------------------	--------------------------	---------------------------------

3-37

* Request Type

- 1) Priority
- 2) Time (Single Execution or Periodic)

Figure 4-3

to move physically in BOSS memory. Therefore, each TQI contains pointers with which it can be linked to any queue. Since the TQIs do not move about in Task Queue Memory, it becomes somewhat more difficult to control the utilization of Task Queue Memory efficiently. Holes or gaps in the Task Queue Memory accumulate as TQIs are added and deleted. Efficient utilization of Task Queue Memory is aided by subdividing it into fixed length segments, each equal to the fixed length of a TQI. The Queue Block Status Matrix (QBSM) is then a map of Task Queue Memory where each bit in a QBSM word represents a TQI block and identifies whether it is utilized or empty. The QBSM is used by the Task Scheduler constructing the TQI and inserting it into an available slot in the Task Queue Memory.

The Dispatcher utilizes the contents of the Priority Execution Queues to determine which task to place into execution next. The Dispatcher is entered whenever a change occurs in the number of available resources, whenever a change occurs in the status of operational resources, or whenever the status of the Priority Execution Queues is modified.

Once entered, the Dispatcher will attempt to put into execution as many pending task requests as it can support with available resources. The resource pool is defined by the contents of the Unit Status Table (UST), the Bus Status Word (BSW), and the Resource Pool Counters (RPCs).

The UST defines the status of all CPEs, IOPs, and output voter/switches. It defines whether each unit is utilized, available, failed, reserved, or off-line, and whether it is fully operational or only partially operational.

The BSW defines the status of all system buses and, for each bus, defines whether it is utilized, available, failed, reserved, or off-line. A bus cannot be partially operational; it is either capable of transmitting data or not.

The RPCs keep track of the total number of available resources of each type. The RPCs are used by the Dispatcher to quickly evaluate the possibility of constructing the desired stream, whereas the UST and BSW are used to identify specific stream components.

Once the Dispatcher has identified the specific modules to be utilized in the new stream, it requests the Configurator to interconnect the identified resources and establishes an entry in the Master Execution Table (MET). The functional layout of the MET is presented in Figure 4-4.

In order to identify to the Configurator the resources to be interconnected and the criticality of the desired stream, the Dispatcher builds a Configuration Stream Request Word (CSRW) which identifies the function to be performed and the particular MET entry which identifies the stream components.

The MET defines all potential streams associated with a particular ARMMS mission, and provides entries for recording the resources dedicated to each defined stream. The MET also defines the TQI assigned to a given stream and its associated dispatch and execution priorities.

Normally, the execution priority equals the dispatch priority for all processing streams. However, if a failure is detected in a TMR processing stream, the execution priority will be bumped to a higher level to prevent any other application tasks from interrupting the failing stream before that TMR task has a chance to complete. I/O streams will also have an execution priority which differs from the dispatch priority since, by definition, I/O streams, once dispatched, cannot be interrupted by higher priority requests.

If the Dispatcher cannot obtain sufficient resources from the resource pool for the new task, it will identify from the MET the executing stream of lowest lower executing priority. Any such stream will be placed in a configuration wait and its stream components made available to the resource pool. This search operation will be continued by the Dispatcher until either sufficient resources are identified or no more streams of lower priority are currently executing. No executing streams are actually stopped by the Dispatcher until it is certain that such action will result in sufficient resources for the new stream. If the stream cannot be put into execution, it will be treated as either a resource problem or as a temporary overload condition which leaves the TQI pending in the Priority Execution Queues.

Once the stream has been identified and constructed, the Initiator is called to start the requested task executing on the defined stream. When the task completes, the Task Terminator is called.

The Task Terminator is responsible for deleting the terminating task's TQI from the system unless the task is periodic. If the task is periodic, it calculates the next execution time and reschedules the task for execution. The Task Terminator also initiates the event processing logic which is responsible for determining if any other task is awaiting the completion of the terminating task.

4.4.2 Task Control Components

The ACES task control logic has been subdivided into the seven functional areas listed below:

- o Task Scheduler
- o Timer Scheduler
- o Priority Scheduler
- o Dispatcher
- o Timer Processor
- o Task Terminator
- o Initiator

A functional description of each is presented in the remaining subsections of Section 4.

4.4.2.1 Task Scheduler

The Task Scheduler is responsible for accepting and processing task control requests from application tasks. Figure 4-5 presents a functional flow diagram of the Task Scheduler logic.

The Task Scheduler will accept two (2) types of task calls:

- 1) Task Schedule calls;
- 2) Delete Task calls.

The Task Schedule call allows any application task to request the execution of another specific task. Figure 4-6 defines the functional parameters of the Task Schedule call. The Task Scheduler combines the functional parameters of the Task Schedule call with the related parameters defined in the Task Dictionary to build a TQI for the requested task. Figure 4-7 defines the functional parameters of a typical task dictionary entry.

Two primary types of Task Schedule requests can be accepted by the Task Scheduler: requests for priority execution with no associated time constraint, and requests for a timed execution which require the definition of a time-to-execute parameter and a repetition period parameter for a periodic task. The request for priority execution with no associated time constraints, and the request for a timed execution

TASK SCHEDULER

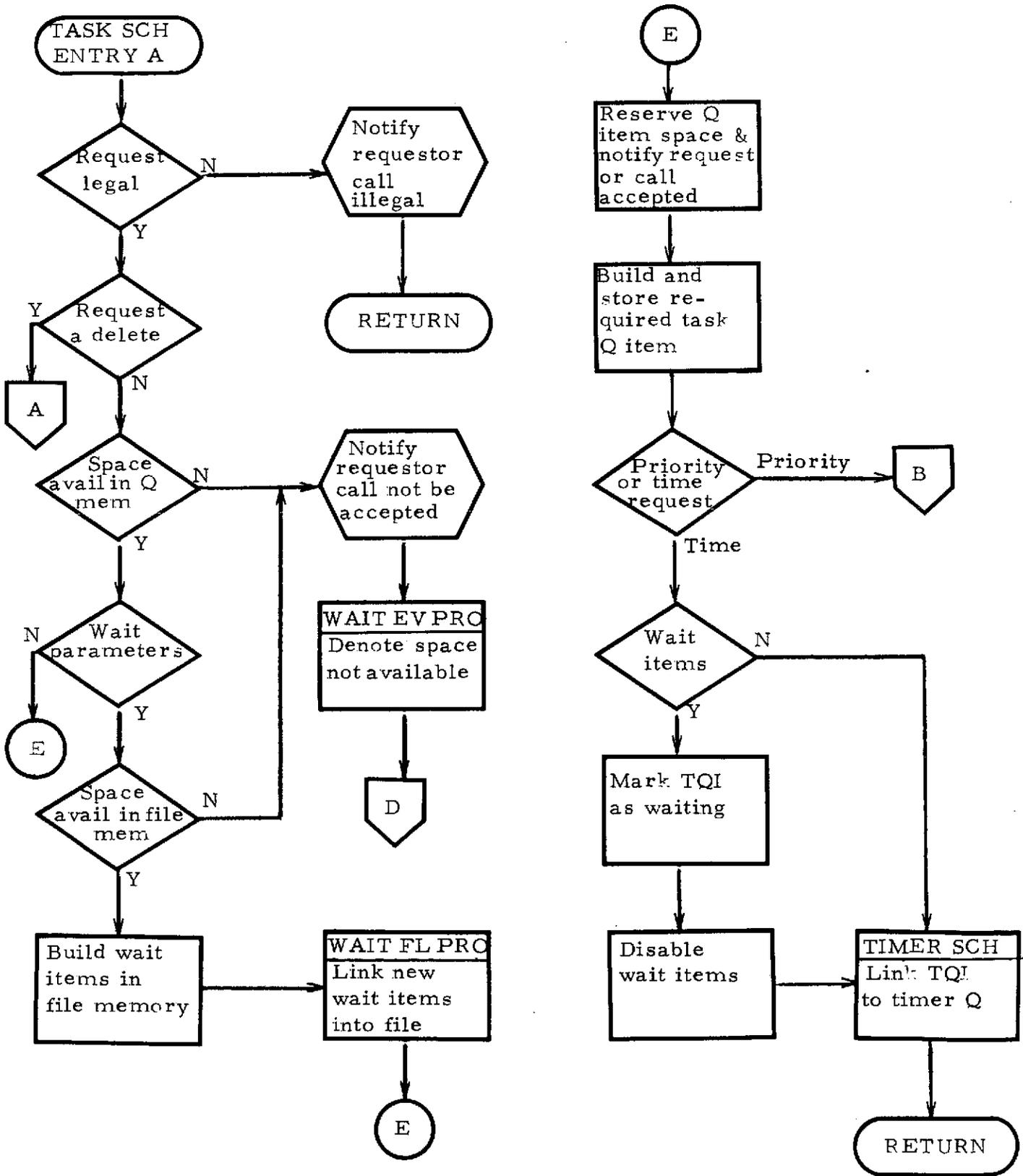


Figure 4-5

TASK SCHEDULER
(continued)

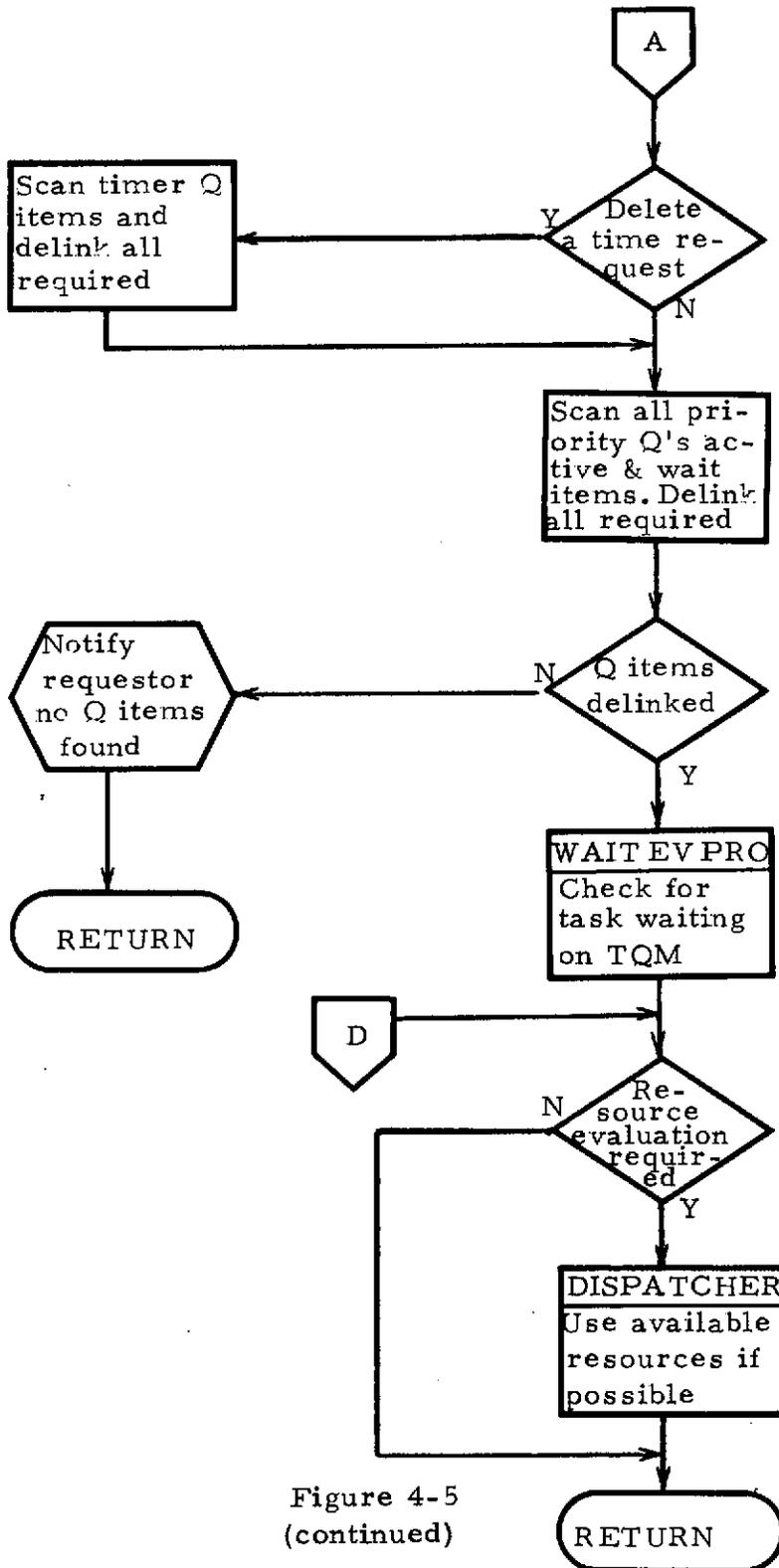


Figure 4-5
(continued)

TASK SCHEDULER
(continued)

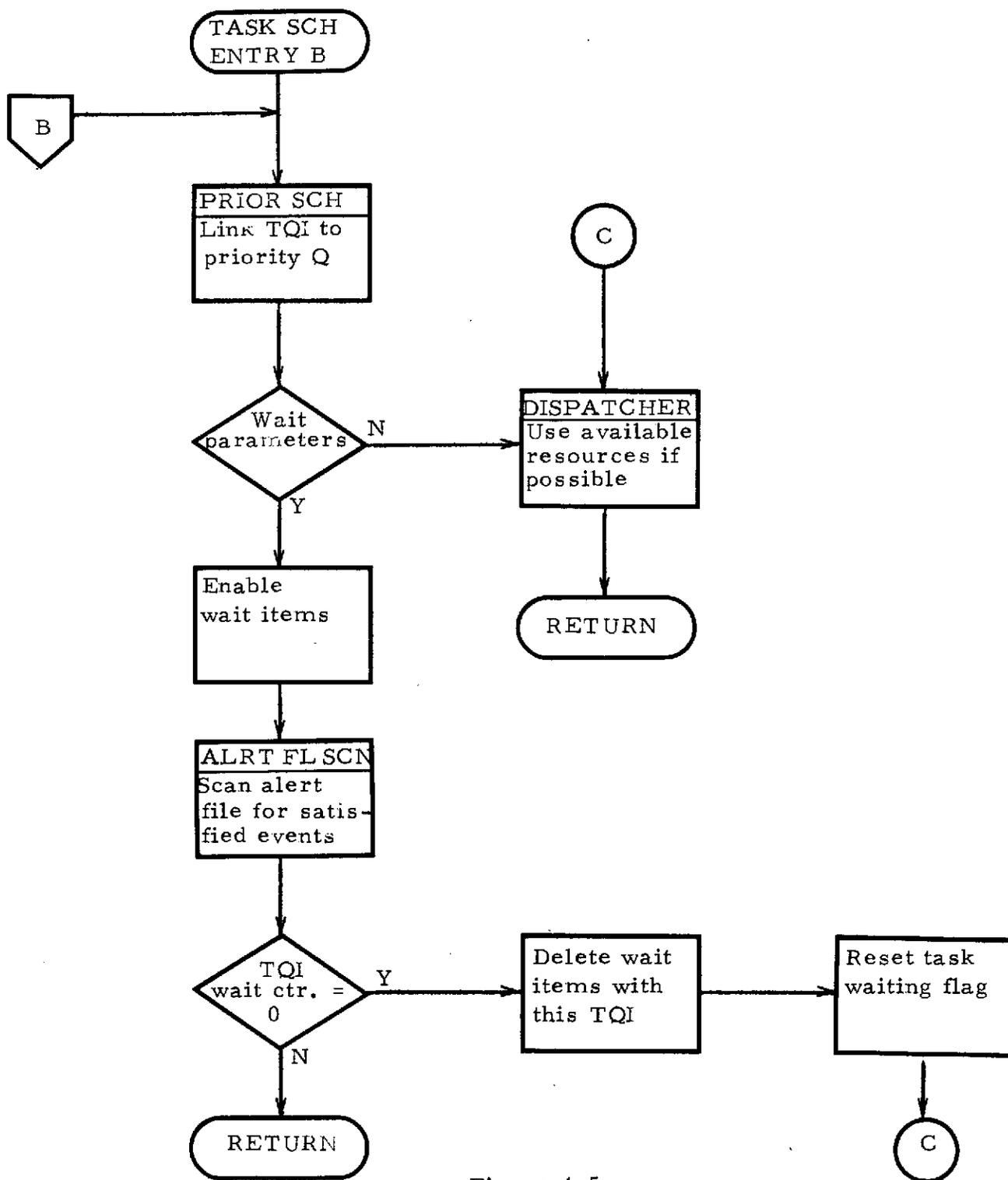
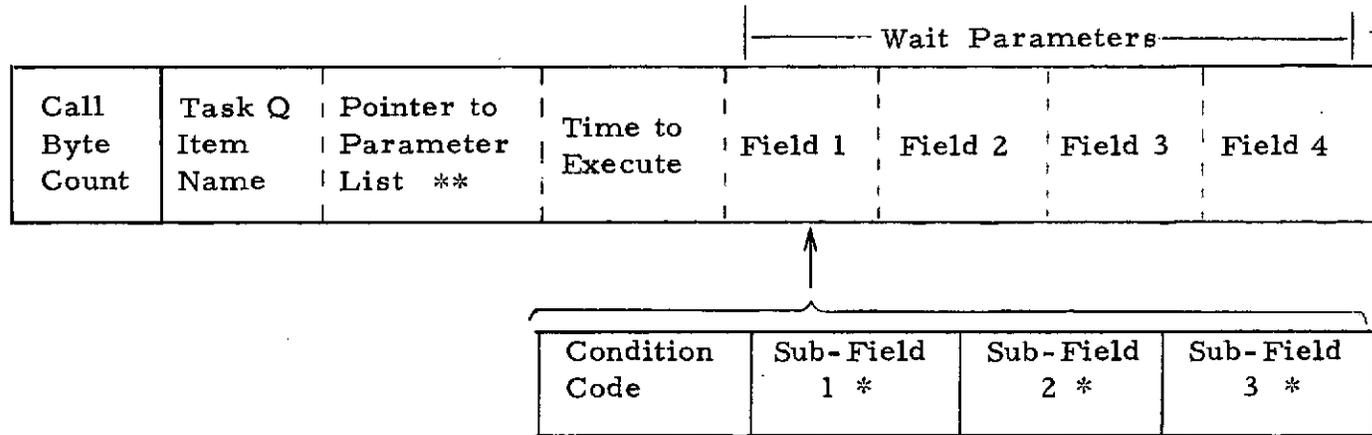


Figure 4-5
(continued)

SCHEDULE TASK CALL (FUNCTIONAL DEFINITION)



3-45

* Sub-Field Usage Event Dependent

** Pointer To Data Set To Be Passed From Calling To Receiving Task

Figure 4-6

TASK REQUEST DICTIONARY ENTRY (FUNCTIONAL DEFINITION)

Task Q Item Name	Request Type	Dispatch Priority	Task Criticality (T, D, S)	Pointer to Routine to Execute	Period of Periodic Task	Stream Type	Task Name (Number)
------------------------	-----------------	----------------------	----------------------------------	-------------------------------------	-------------------------------	----------------	--------------------------

Program Base/ Bound (B/B)	Local Data B/B	Temporary Storage B/B
------------------------------------	----------------------	-----------------------------

3-46

Request Types

1. Schedule Priority
2. Schedule Time (Periodic & Single Execution)

Stream Type

- I/O
 Processing (Full)
 Processing (Limited)

B/B - Base Bound
 Registers

Figure 4-7

with a time-to-execute parameter, but no repetition period (single execution), may have Wait Items as part of the request. A request for a timed execution with a time-to-execute parameter and a repetition period parameter (periodic execution) cannot have Wait Items associated with the request. It should be noted that the request type parameter resides in the dictionary and is not passed to the scheduler in the task call. In fact, the task's dictionary entry contains all of the major parameters which govern its mode of execution.

This may initially appear to limit the flexibility of the task call, since in some instances it may be desirable to dynamically alter such execution parameters as priority, criticality, etc., as conditions change during a mission phase. No such limitations exist, however, since the common parameter which links a Task Schedule call to a specific task dictionary entry is a Task Queue Item name, not the task name. This means that a dictionary may contain as many distinct entries referencing a common task as necessary to handle any variations required in the execution parameters.

This approach was adopted for two reasons. First, it minimizes the amount of information which must be transmitted to BOSS dynamically by an application program task. Secondly, it forces the system users to preplan the required modes of execution for each phase task, which is compatible with the overall philosophy of requiring the user to preplan worst case phase task scheduling in order to insure that all critical time deadlines can be satisfied.

The Delete Task call allows any application task to request that a previously requested task's TQI be deleted from the system. Such deletion requests will not be honored if the task associated with the referenced TQI has entered execution.

When the Task Scheduler is entered, it first confirms the legality of the task call and then determines whether the call was a Schedule or Delete call.

If the call was a Schedule call and space is available in Task Queue Memory (TQM) for an additional TQI, a check is made to determine if wait parameters are present, which must be satisfied before the task is eligible for execution. If Wait Items are present and space is available, the Wait Items are moved to File Memory and linked into the Wait Item file. The TQI is then constructed and placed in the TQM. The Task Scheduler then examines the key execution parameters of the TQI, to link the TQI to either the Timer or Priority Task Queues. If the request was a timed request, and had Wait Items

associated with it, the Wait Items are disabled so that they will not be satisfied during the time interval.

After the time interval expires and the TQI is moved to the proper Priority Queue, a check is made to determine if wait parameters were associated with the TQI. If not, the Dispatcher is called to attempt to place the task into execution. If Wait Items are associated with the task, the items are enabled so that monitoring for the event can begin. The Alert File is scanned to determine if any of the events had already occurred, as noted by previously defined alerts. If all the events are satisfied by the alerts, the Wait Items are deleted and the Dispatcher is called to attempt to place the task into execution. If not, the Task Scheduler exits.

If the call was a Delete call, the Task Scheduler deletes the identified TQI from its present scheduling queue, if its associated task has not yet entered execution via this TQI. Once a TQI is deleted, space is available in TQM and it is necessary to determine whether any task is currently waiting for such space to become available. A new resource evaluation is required if the occurrence of available space in the TQM resulted in a change in the status of the Priority Execution Queues.

4.4.2.2 Timer Scheduler

The Timer Scheduler is entered from the Task Scheduler. Its primary function is to link the Timer Queue and a new TQI which has time constraint execution parameters. Figure 4-8 presents a functional flow diagram of the Timer Scheduler.

If the requested TQI's time-to-execute is less than current time, the Timer Scheduler will pass control immediately back to the Task Scheduler which attaches the TQI to the Priority Execution Queues.

4.4.2.3 Priority Scheduler

Figure 4-9 presents a functional flow diagram of the Priority Scheduler. The Priority Scheduler is entered from the Task Scheduler. Its primary function is to link a new TQI to the proper Priority Execution Queue. If necessary, the Priority Scheduler opens the required Priority Execution Queue. Upon completion, the Priority Scheduler returns control to the Task Scheduler.

TIMER SCHEDULER

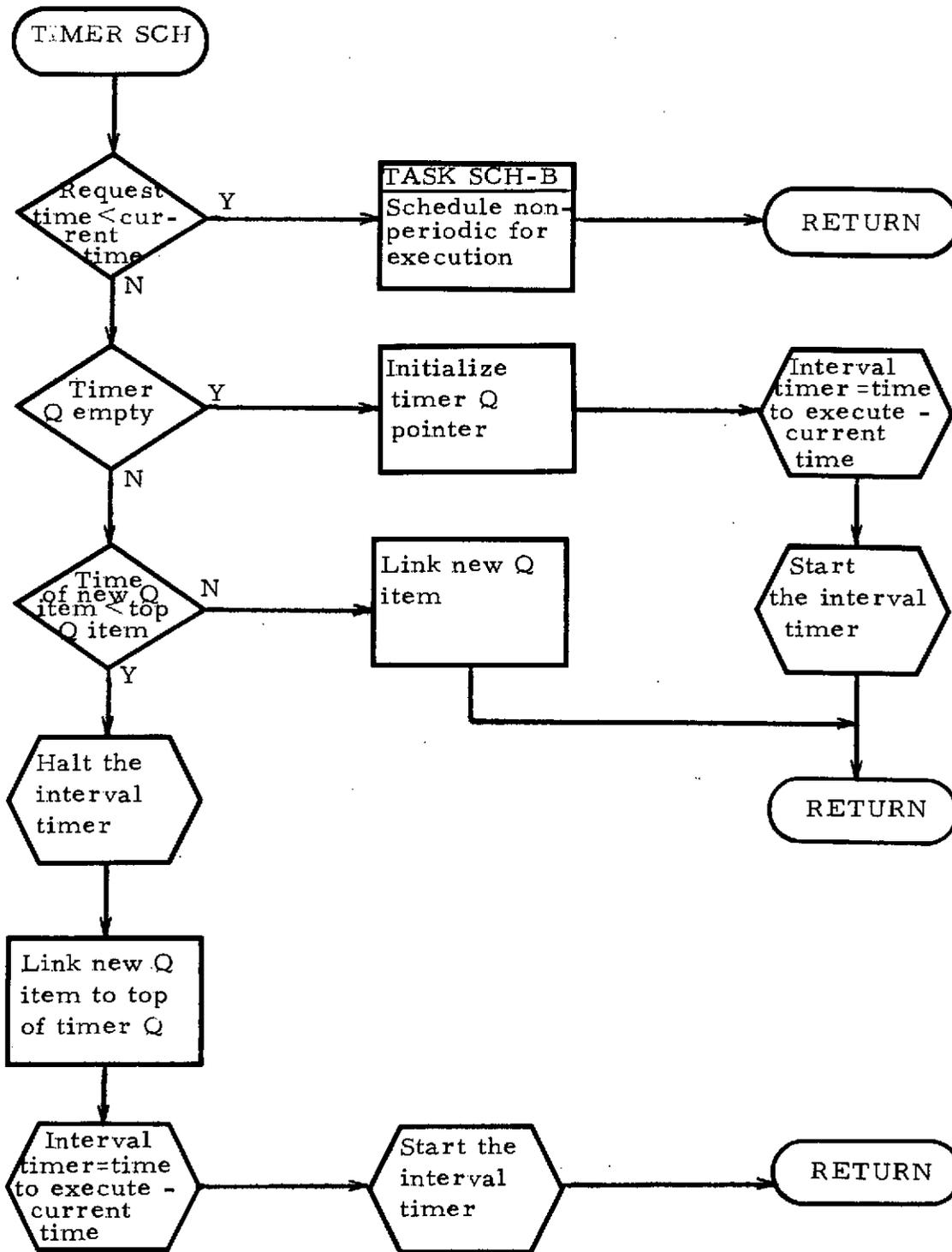


Figure 4-8

PRIORITY SCHEDULER

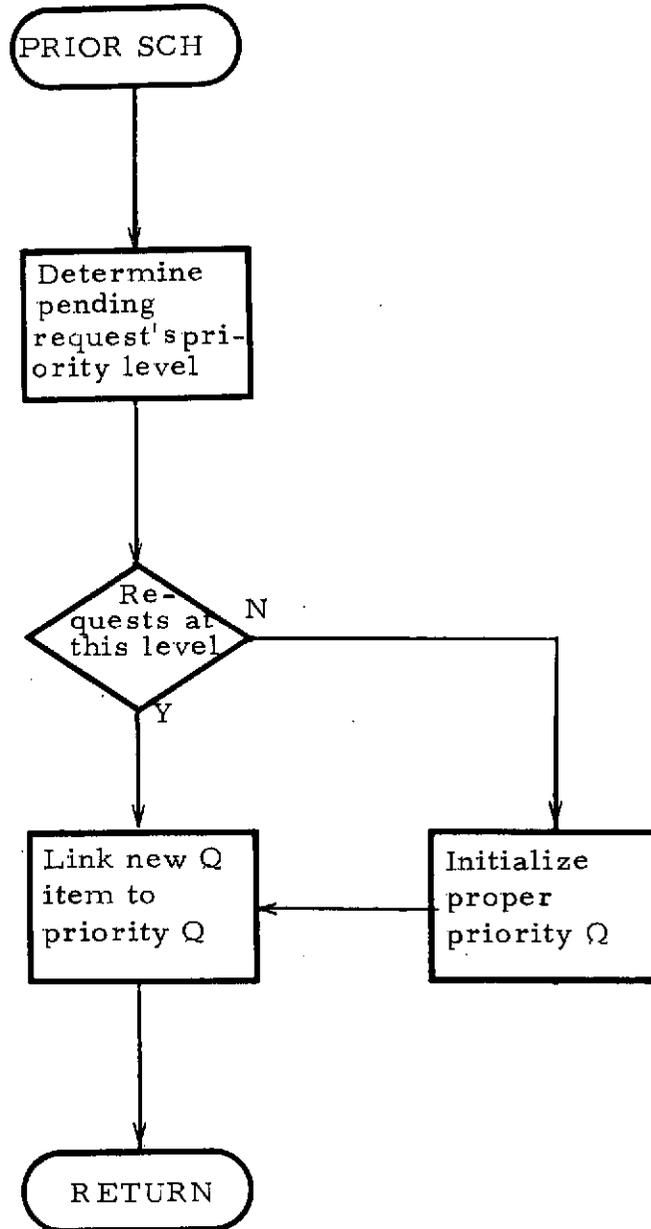


Figure 4-9

4.4.2.4 Dispatcher

The Dispatcher is responsible for determining which task to place into execution next. Figure 4-10 presents a function flow diagram of the Dispatcher logic.

The Dispatcher is entered when a system status modification occurs which alters the contents of the Priority Execution Queues, or when a task in execution terminates or enters the wait state. Whenever it is entered, the Dispatcher assumes that a need exists to re-evaluate all pending tasks in the light of currently available resources in order to select the next candidate tasks, eligible for execution. It also determines whether any device failures have been recorded since its previous entry.

Once entered, the Dispatcher scans each TQI, in turn, to determine if it can be placed into execution. When a TQI is marked as waiting for an event to occur, the Dispatcher performs no further evaluation for the TQI, but rather obtains the next TQI.

The Dispatcher determines if the highest priority pending task has a dispatch priority greater than at least one of the currently executing tasks. If it does, then it is a candidate for further consideration at this time. If not, then a test is made to determine whether the system is currently supporting execution of the maximum number of streams possible at the defined criticality levels. For ARMMS, this would be equivalent to supporting the simultaneous execution of streams with a cumulative stream weighting factor of four for both I/O and processing streams. If the current cumulative stream weighting factor is less than four of the desired stream type, then the current TQI is a candidate for immediate further consideration.

Once this decision is reached, the TQI's criticality is examined and, if the criticality is TMR, a check is made to insure that the TMR Dispatch Inhibit Flag (TDIF) is not set. If the TMR Dispatch Inhibit Flag is set, it indicates that a failure was identified in an executing TMR stream and that no further TMR streams are to be dispatched until the current TMR stream terminates and diagnostics are run to isolate the cause of the failure.

For other criticalities, or if the TDIF is not set, a check is made to insure that the task that is requested has not already begun execution in any other stream. If so, starting the task in this stream could yield task re-entrancy problems; therefore, the TQI is bypassed by the Dispatcher and the next TQI is obtained.

DISPATCHER

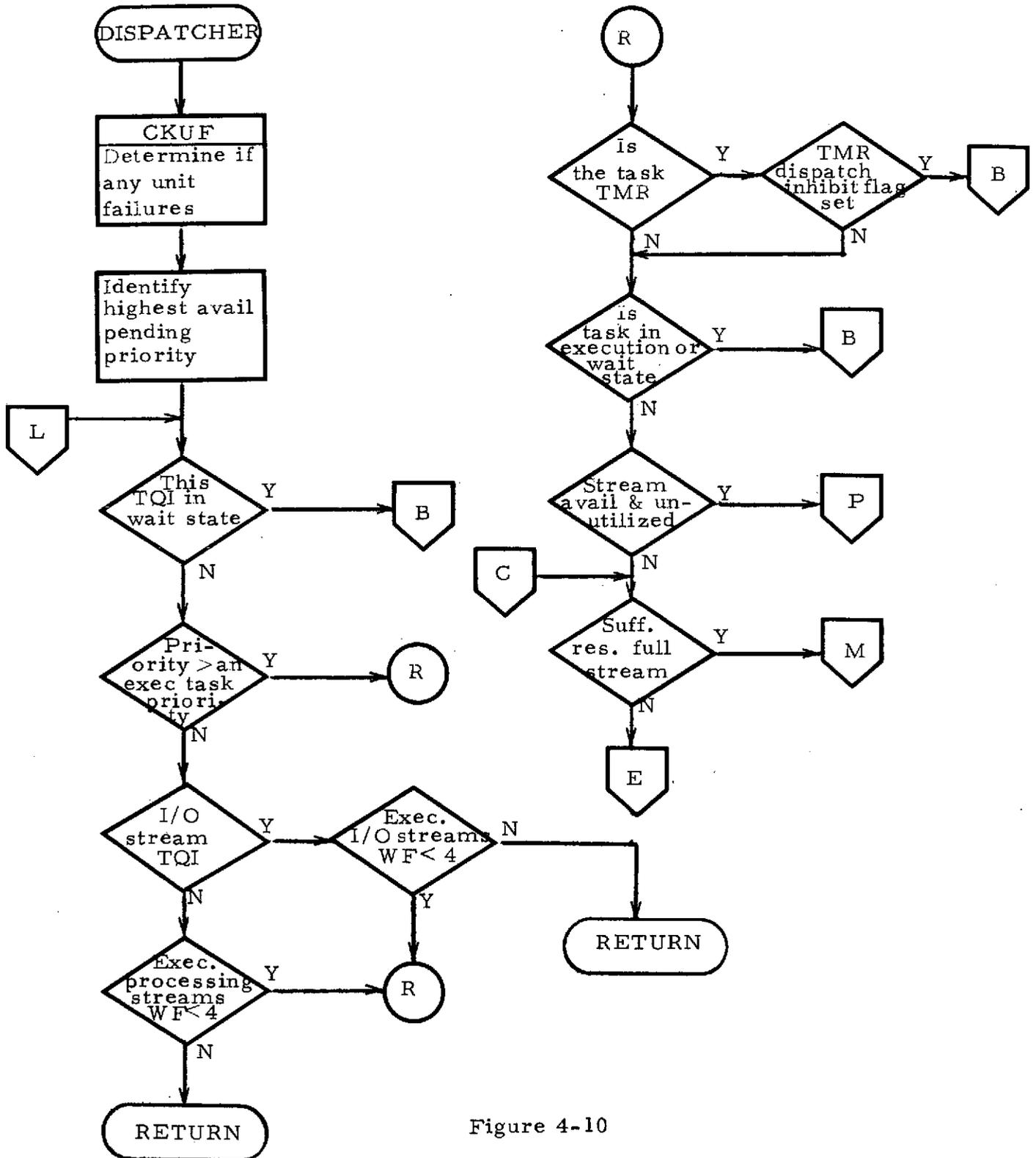


Figure 4-10

DISPATCHER
(continued)

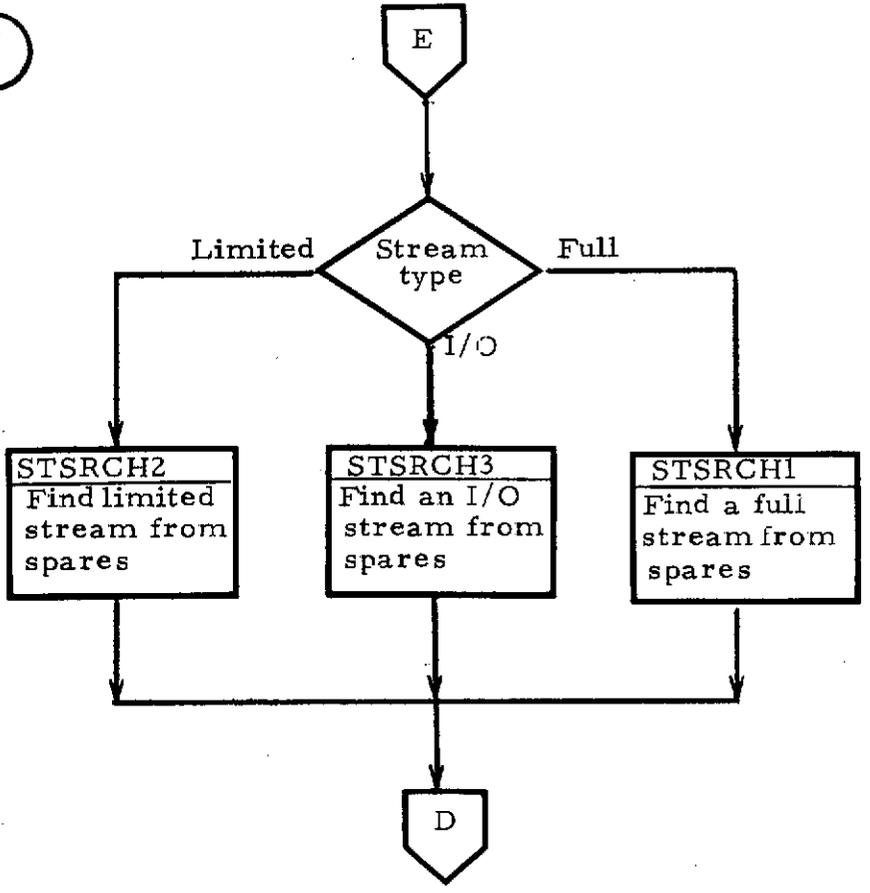
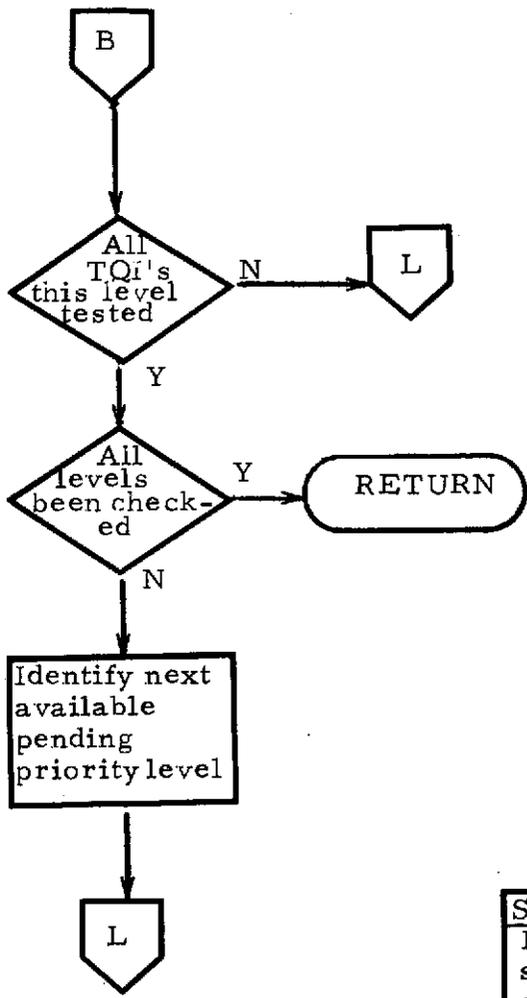


Figure 4-10
(continued)

DISPATCHER
(continued)

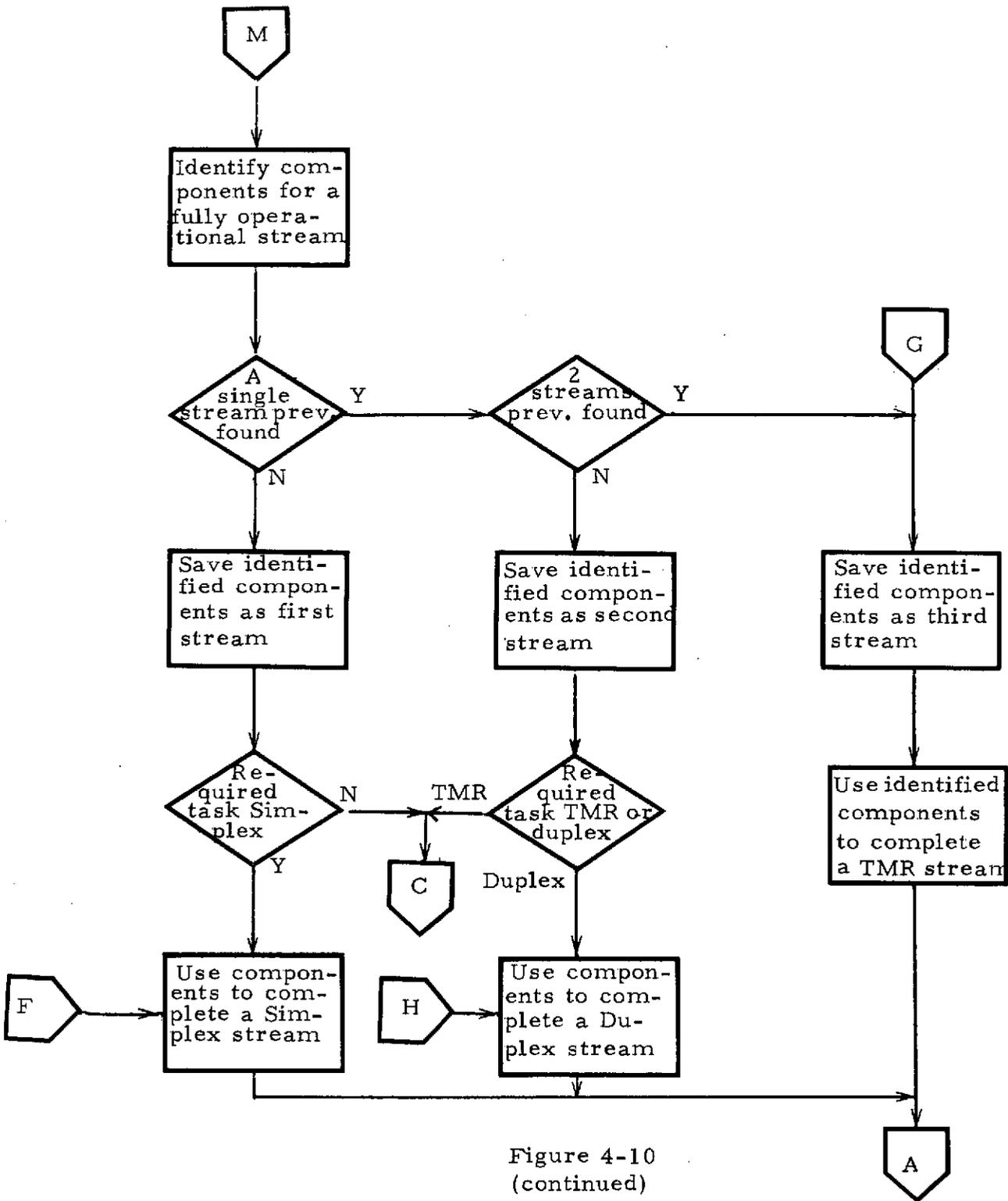


Figure 4-10
(continued)

DISPATCHER
(continued)

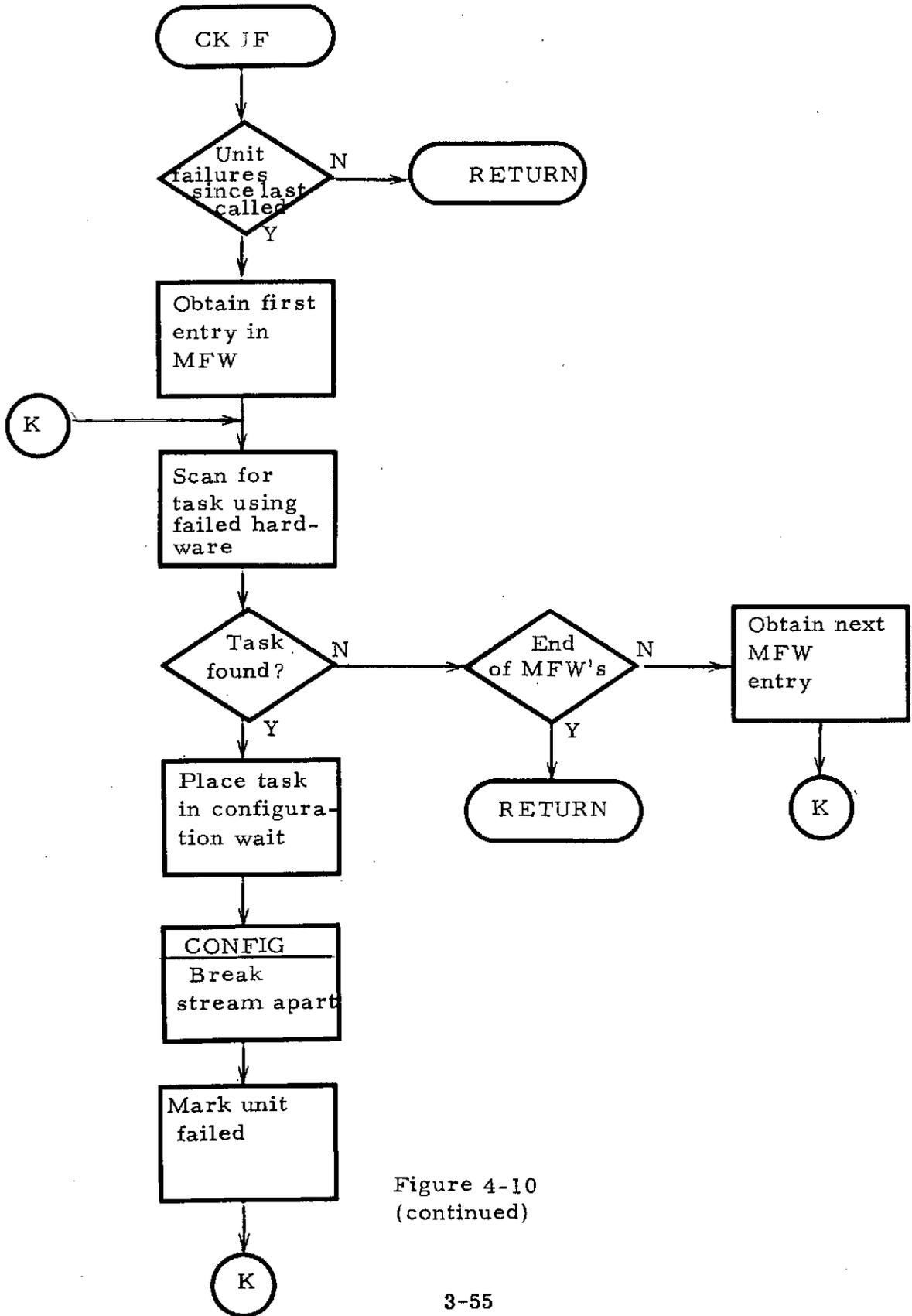


Figure 4-10
(continued)

DISPATCHER
(continued)

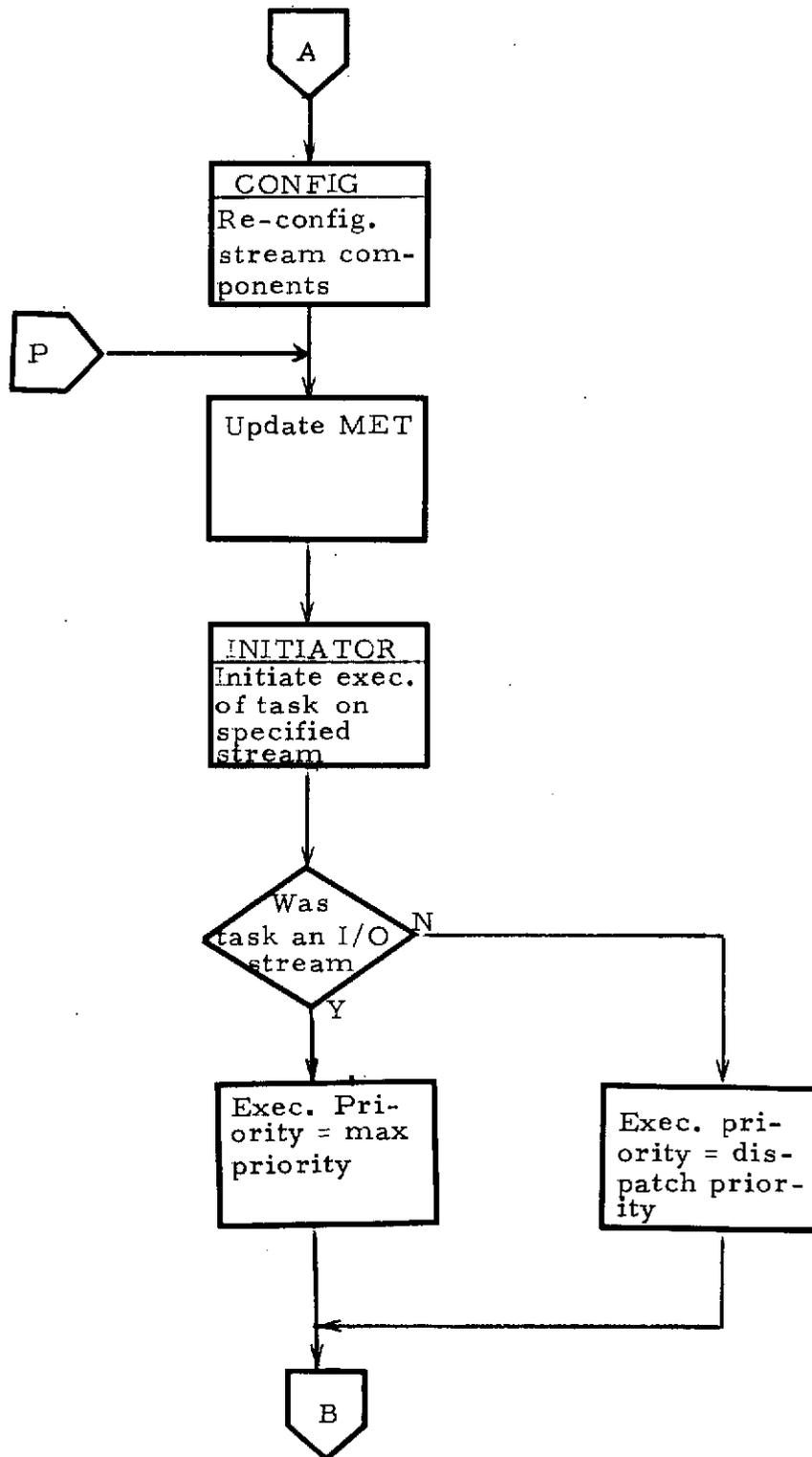


Figure 4-10
(continued)

If the requested task has not begun execution in another stream, the Dispatcher determines if a spare stream of this criticality and type is already built but not currently utilized. If so, the TQI is placed into execution utilizing this available stream.

If a spare stream is not available, a scan is made of the RPCs to determine if sufficient fully operational resources are available to construct a new simplex stream.

If the scan of the RPCs indicates that sufficient fully operational resources are available to construct a simplex stream, then the BSW and UST will be scanned to identify the explicit stream components to be utilized.

The Dispatcher looks for components for only one simplex stream at a time. If the TQI criticality is duplex or TMR, then multiple simplex stream components must be identified. Once sufficient resources have been identified and reserved to satisfy the criticality of the TQI, the Dispatcher passes control of the Configurator which is requested to interconnect the identified components into a stream with the desired criticality. As soon as the stream is ready, control is returned to the Dispatcher, which then passes control to the Initiator which is responsible for initiating task execution on the defined stream. The Dispatcher communicates with both the Configurator and the Initiator via the MET entry for this stream which defines the stream components and the using TQI.

Once the requested task has been dispatched, the Dispatcher loops back to its beginning to determine if it can make use of any of the remaining available resources to satisfy pending task requirements.

If at any time during its search for fully operational resources, the Dispatcher determines that insufficient resources are available to satisfy the TQI's criticality requirements, it will attempt to identify simplex stream components made up of any spare resources, fully or partially operational. The Stream Search subroutine is responsible for identifying such stream components.

Not until all possibilities of using available spare resources have been exhausted will the Dispatcher consider interrupting an already executing stream. When such a situation is identified, a number of tests are performed to determine the potential capabilities of the existing system as they relate to the criticality and priority of the task it is desired to dispatch.

First, a cumulative weighting factor is calculated for all equivalent spare simplex streams identified in the search, and for all streams currently executing with a priority less than that of the requested task. So long as this cumulative weighting factor is larger than or equal to the criticality of the requested task, sufficient lower priority streams and spare available resources exist to construct a stream for the requested task. In such a situation, sufficient lower priority executing tasks will be interrupted and placed in a configuration wait to make the resources required by the new task available.

If the cumulative weighting factor indicates that sufficient spare and lower priority resources are not available to dispatch the new task, the Dispatcher must determine if a temporary overload condition exists or whether there is a resource problem.

A second cumulative weighting factor is calculated for all spare simplex streams identified so far in the search, and for all tasks currently executing. This is effectively the weighting factor of the operational system resources. If this weighting factor is greater than or equal to the criticality of the requested task, then sufficient resources are currently operational in the system to eventually execute the requested task when it becomes the highest priority pending task. This situation is defined as an "overload" condition and the requested task's TQI is left pending by the Dispatcher.

If the second cumulative weighting factor is smaller than the criticality of the requested task, then a "resource problem" is said to exist since sufficient resources are not operational. When a resource problem occurs, the Dispatcher will equate the criticality of the requested task's TQI to the second cumulative weighting factor. This enables the requested task to be executed on the available operational system resources as soon as that task becomes the highest priority pending task.

4.4.2.5 Initiator

Figure 4-11 presents a functional flow diagram of the Initiator. The Initiator is entered from the Dispatcher and is responsible for initiating the execution of a specific task on a stream designated by an MET entry. The Initiator is capable of starting either an I/O stream, or a processing stream and, in the case of processing streams, is able to distinguish between a fresh start and a restart. The restart process requires a much more complex program restore and initiation sequence.

INITIATOR

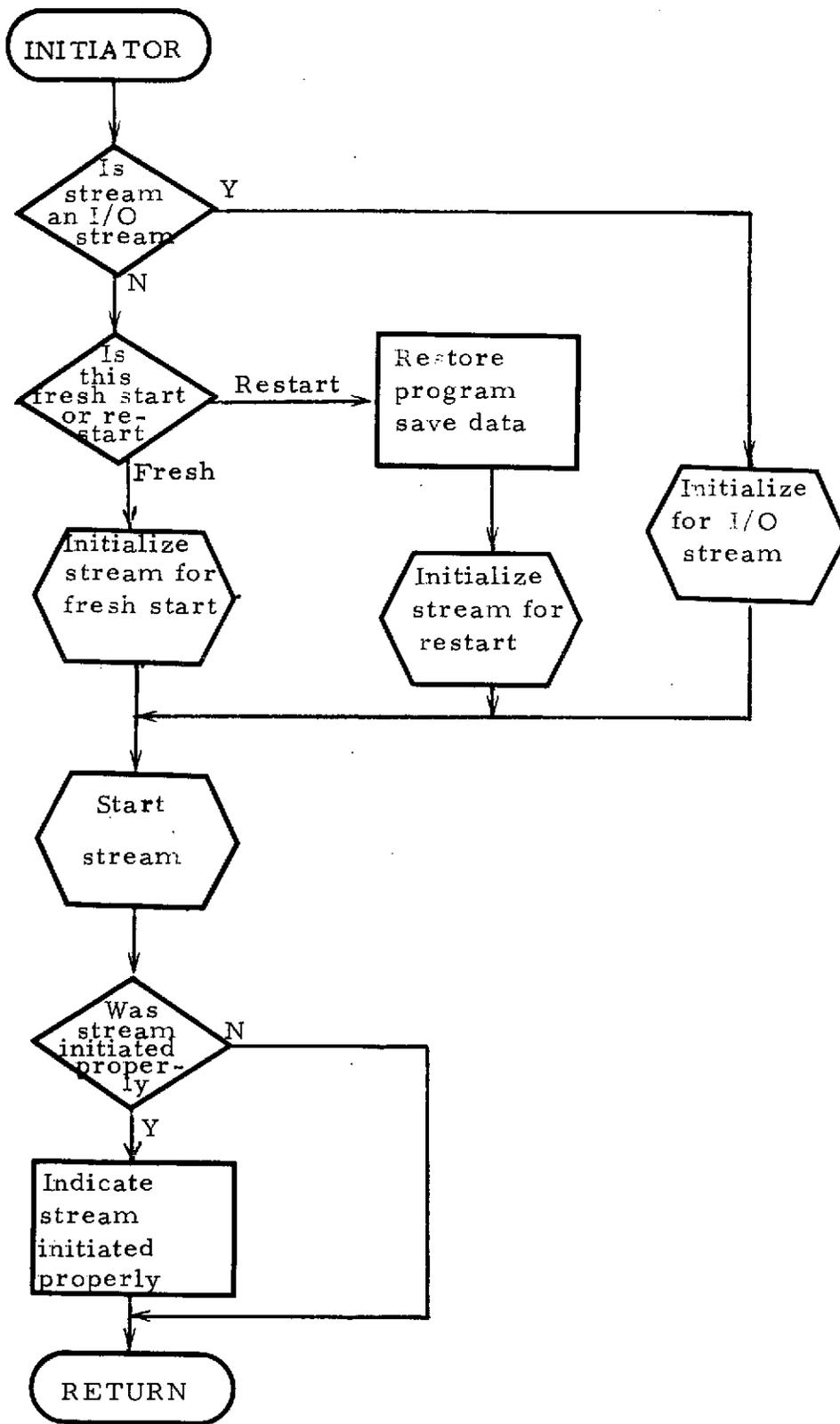


Figure 4-11

4.4.2.6 Timer Processor

The Timer Processor is entered in response to a timer interrupt from an ARMMS interval timer. Figure 4-12 presents a functional flow diagram of the Timer Processor logic.

The Timer Processor is responsible for determining when the time restriction on a requested task's TQI has been satisfied and for moving that TQI to the Priority Execution Queues.

When entered, the Timer Processor first confirms that current time (the interrupt time) agrees with the time-to-execute parameter of the top TQI in the timer queue. A disagreement indicates a timer error condition which must be diagnosed.

The top timer TQI is then moved to the Priority Queues. The TQI is then interrogated for wait parameters. If any are present, they are enabled and the Alert File is scanned in order to ascertain the status of wait events for which alerts have been established to monitor. After scanning the Alert File, if any of the events are not satisfied, the Timer Processor exits. If all events are marked satisfied after scanning the Alert File, the TQI waiting flag is reset and the Wait Items are deleted.

If there are no wait parameters, or if all of the items are satisfied by the Alert File scan, the Dispatcher is called to determine if this TQI can be placed into execution.

Next, the Timer Processor advances the Timer Queue and calculates a delta equal to the difference between the time-to-execute parameter of the new top queue TQI and the current real time. If this time is zero, the Timer Processor loops back to the beginning to process this TQI, whose time parameter has expired. If the delta is not zero, the internal timer is loaded and started before the Timer Processor exits.

4.4.2.7 Task Terminator

The Task Terminator is entered in response to a termination call from an application task. Figure 4-13 presents a functional flow diagram of the termination logic.

The Task Terminator is responsible for detaching the completed task's TQI from the system and restarting a periodic TQI. In general, the flow diagram is self-explanatory. However, a number of clarifying statements are pertinent.

TIMER PROCESSOR

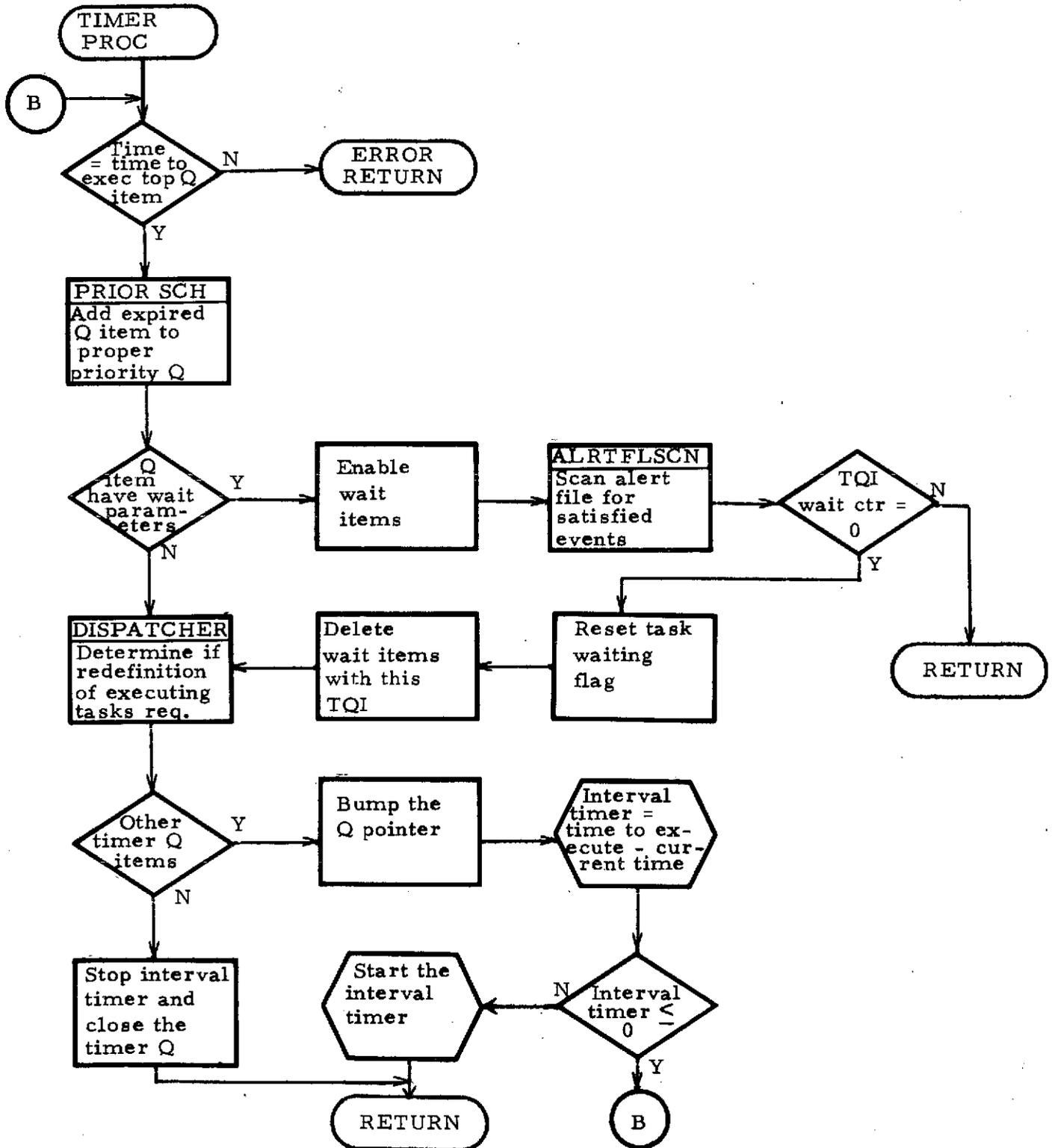


Figure 4-12

TASK TERMINATOR

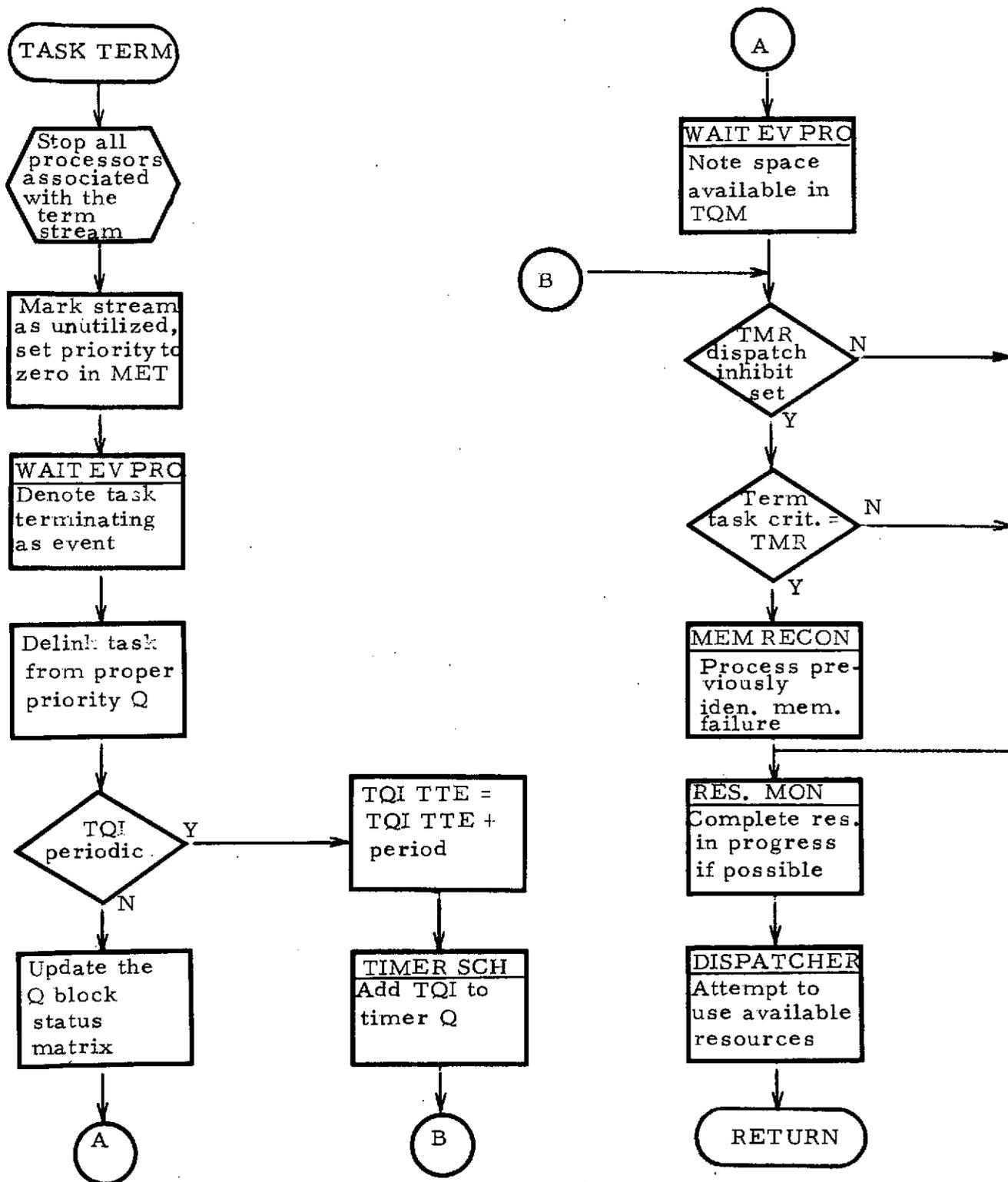


Figure 4-13

Note: TTE - Time to Execute

First, the Task Terminator does not break apart the stream that was executing. Instead, when a task terminates, the MET entry for the stream is marked as not utilized. The act of a task terminating is a defined event. The Wait Event Processor is called by the Task Terminator to communicate this event to the Wait and Alert Files.

If a task is not periodic, the TQI is flushed from the system and the Wait Event Processor is notified of such an event. If the task is periodic, the next period, or time to execute, must be established and the TQI placed into the Timer Queue.

Next, if the TMR Dispatch Inhibit Flag is set and the stream terminating is a TMR stream, it indicates that a failure was previously identified in a triad memory component and that the failure must now be isolated and the triad corrected, if possible. This operation will be performed by the Memory Reconfigurator.

Then, the Task Terminator will call the Reservation Monitor in order to determine if a diagnostic resource reservation is outstanding which might be satisfied by the resources being relinquished by this stream.

The last function of the Task Terminator is to call the Dispatcher to determine whether any pending task can be put into execution on a stream constructed from the currently available resources.

4.5 Event Recognition and Response

4.5.1 Event Processing Overview

Event Recognition and Response Processing consists of the algorithms and design concepts required to:

- o Allow application tasks to establish a system requirement to monitor and record specific event occurrences.
- o Allow ACES to initiate specific application and system tasks in response to dynamic event occurrences.
- o Allow application tasks to set and/or interrogate the condition of defined events during execution.

An event is defined to be any occurrence for which monitoring logic has been provided in the ACES. Currently, nine events have been identified as defined below:

- 1) Task Termination - a specific task (i. e., a named TQI) has terminated.
- 2) Task ABEND - a specific task has abnormally ended.
- 3) Task Waiting - a specific task has entered the wait state.
- 4) Reservation Complete - a diagnostic resource reservation has been completed.
- 5) Logical Memory Address Fully Functional - a logical memory address has been returned to full operation status.
- 6) Space Available in FM - space is now available in File Memory.
- 7) Space Available in TQM - space is now available in Task Queue Memory.
- 8) Program Flag - a program flag has been either set or reset.
- 9) Variable Locked - a specific lock-variable has been locked.

It is expected that this list will grow considerably in the future as new "official" events are identified. For example, the detailed definition of I/O processing requirements will probably result in a substantial number of new event definitions.

Some events are single shots, while others are flip-flops. For example, the Reservation Complete event is a single shot. That is, once a reservation request is completed by ACES, it is irreversible. Thus, the event status cannot, once satisfied, become unsatisfied. Conversely, the Program Flag event is a flip-flop event. One task may set the flag at one point and later another task may reset the flag.

Figure 4-14 presents an overview of the Event Processing components and defines their cross-communication linkages at the functional level.

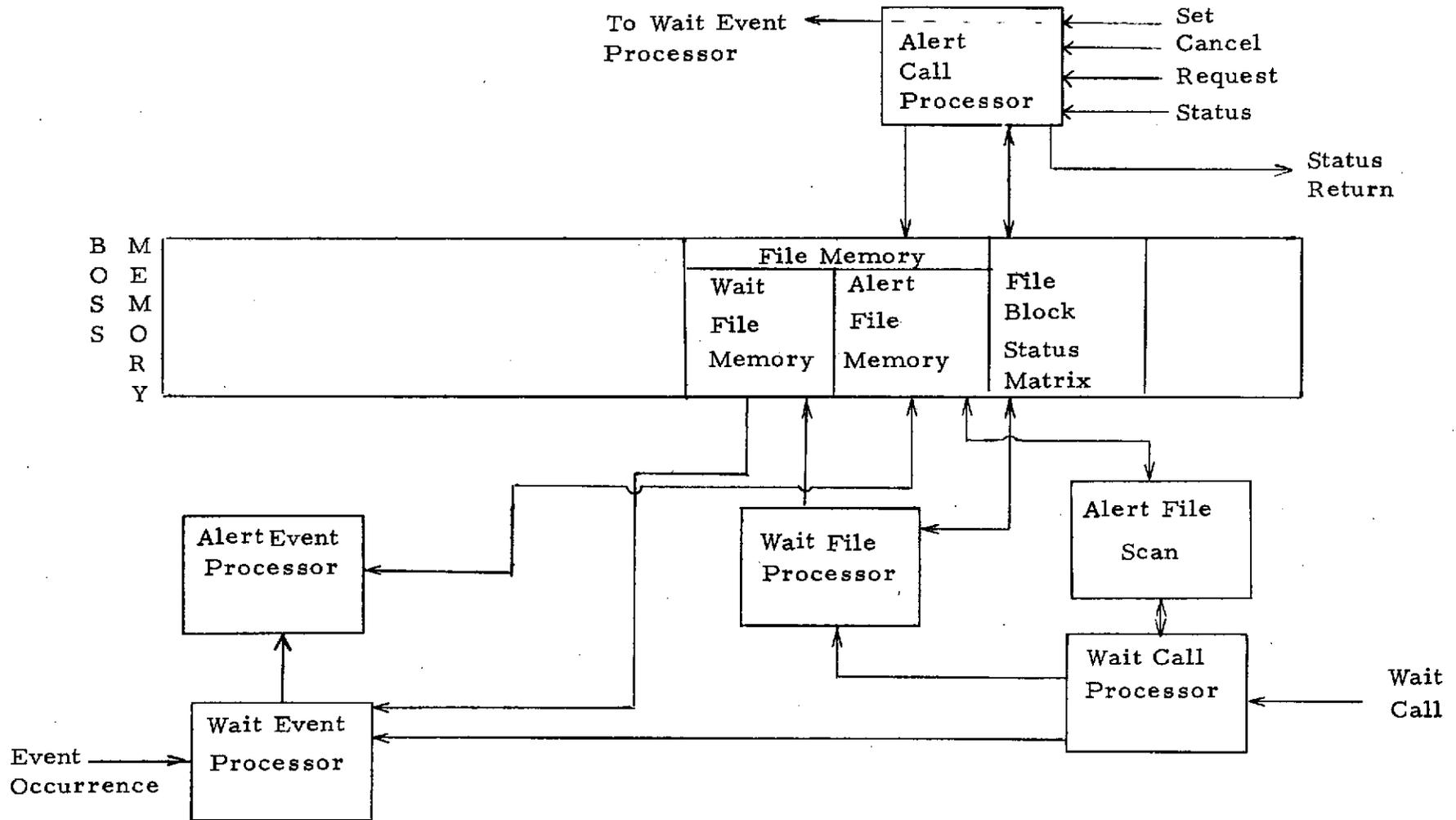
BOSS memory, as presented in Figure 4-14, is the central communications area for controlling all event recognition and response. It contains the File Memory (FM), and the File Block Status Matrix (FBSM). The FM is subdivided into Wait File Memory (WFM) and Alert File Memory (AFM).

Basically, ACES Event Processing logic provides application tasks with two separate mechanisms, Waits and Alerts, to initiate controlled response activity as the result of an event occurrence. In reality, the two mechanisms are closely interwoven to perform overall event monitoring. However, for ease of understanding, each is discussed separately below.

The Wait logic allows a task to request that ACES place it into a wait state until specific events, specified by the calling task, are satisfied. Figure 4-15 presents the functional parameters required by ACES in a Wait Call from an application task. A calling task can specify up to four separate events which must be satisfied before ACES may reactivate the calling task. Each of the four main event fields of the call is subdivided into four divisions. The first division contains the condition code which identifies the event, and divisions two through four are subfields used, as necessary, to define the event parameters which must be satisfied. These parameters are event dependent.

The Wait call parameters are utilized by the Wait Call Processor program to build a Wait Item (WI) in Wait File Memory. Figure 4-16 is a functional definition of WI. As events occur, the Wait Event Processor scans the Wait File. Whenever an event occurs, the status field of the WI's specifying this event, is set to the status of the occurred event. When all WIs, associated with a TQI, are satisfied,

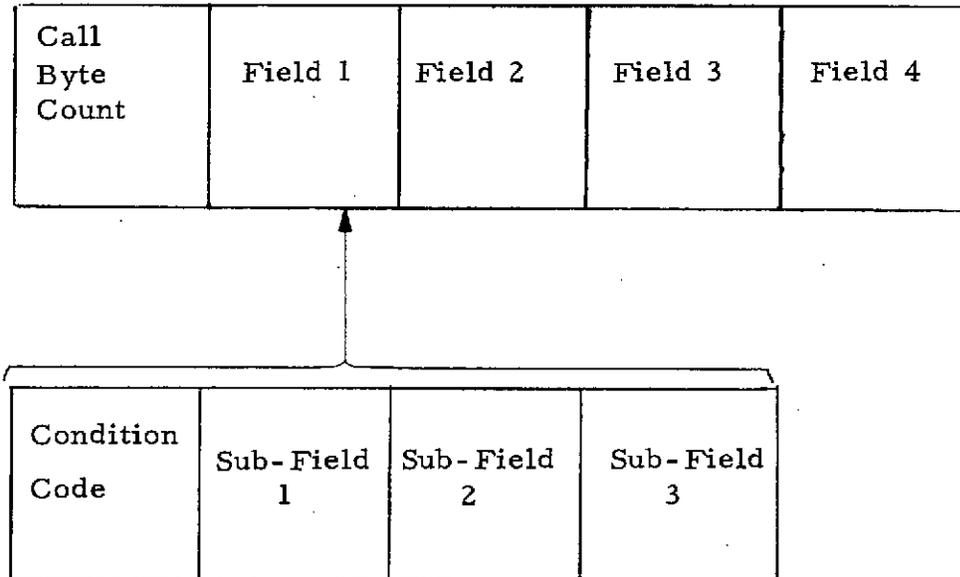
EVENT PROCESSING OVER VIEW



3-67

Figure 4-14

WAIT CALL (FUNCTIONAL DEFINITION)



89-8

Figure 4-15

WAIT ITEM (FUNCTIONAL DEFINITION)

Task Q Item Name	Pointer to TQI	Pointer to Previous File Item	Pointer to Next File Item	Event Status	Condition Code	Sub-Field 1	Sub-Field 2	Sub-Field 3
------------------------	-------------------	-------------------------------------	---------------------------------	-----------------	-------------------	----------------	----------------	----------------

Event Definition Fields

3-69

Event Status

0 = unsatisfied

1 = satisfied

Figure 4-16

the WIs are deleted and the TQI waiting flag is reset. This action eliminates all constraints upon a TQI and allows it to be selected for dispatching.

Wait Items can be associated with a TQI which has a time-to-execute requirement. During the period in which the time is expiring for the TQI, the Wait Items are disabled. This disabling prevents the WIs from changing from the initialized unsatisfied state. When the time has expired, the task's TQI is moved from the Timer Queue to the Priority Queue. It is at this point that ACES begins monitoring for the occurrence of the event.

The Wait call signals the point in a task where continuation of execution is dependent upon the occurrence of specified events. It is often desirable to start monitoring these events prior to the point that necessitates a Wait call. The Alert mechanism described below is provided to fulfill this requirement.

The Alert mechanism is invoked through an Alert Request call, which requests monitoring to be started for a specific event. Obviously, this establishes the requirement for an Alert Cancel call to request monitoring of an event to be terminated. The contents of these calls are functionally defined in Figure 4-17.

The Alert Call Processor uses the components of the Alert Request call to construct an Alert File Item (AFI) which is retained in BOSS memory in the Alert File Memory (AFM). Figure 4-18 defines the functional components of the AFI. It should be noted that each AFI defines only one specific event and is uniquely identifiable by its name supplied in the Alert Request call. It is, therefore, possible to define multiple Alerts, all monitoring the same event, merely by assigning unique names to each AFI.

The AFI and the WI are similar in structure and, in fact, are of the same physical length. Since WIs and AFIs are being continually added and deleted, considerable overhead would be experienced if the items were required to be in a physically sequential order. To prevent this "squishing" operation, each WI and AFI contains forward and reverse pointers with which each item is linked into either the Wait File or Alert File. Since the WI and AFI are not allowed to move within FM, holes or gaps accumulate through addition and deletion of items. Efficient utilization of File Memory is aided by subdividing it into fixed length segments, each equal to a file item. The File Block Status Matrix (FBSM) is then a map of File Memory where each bit in

ALERT CALLS (FUNCTIONAL DEFINITION)

I. Alert Request Call

Request Type*	Alert Name	Condition Code	Sub-Field 1	Sub-Field 2	Sub-Field 3

* Alert Request

II. Alert Cancel Call

Request Type*	Alert Name

* Cancel

III. Event Status Call

Request Type*	Alert Name

* Status

IV. Event Set Call

Request Type*	Alert Name	Event Status

* Set

Event Status: 0 = Unsatisfied
1 = Satisfied

Figure 4-17

ALERT FILE ITEM (FUNCTIONAL DEFINITION)

Alert Name	Pointer to Previous File Item	Pointer to Next File Item	Event Status*	Condition Code	Sub-Field 1	Sub-Field 2	Sub-Field 3
------------	-------------------------------	---------------------------	---------------	----------------	-------------	-------------	-------------

Event Definition Fields

*Event Status

0 = unsatisfied

1 = satisfied

Figure 4-18

a FBSM word represents a File Item (either a Wait Item or an Alert Item) and identifies whether it is utilized or empty. Thus, the utilization of FM is controlled by the Wait File Processor and the Alert Call Processor in much the same manner as the Task Scheduler controls TQM. It should be pointed out that Wait Items and Alert Items are intermingled in File Memory. Each item is only attached to a particular file via its forward and reverse pointers. The FBSM defines the availability and utilization of File Memory, without regard to whether a slot contains a Wait Item or an Alert Item.

In many cases, a task needs to use an event status to select different execution paths. To allow a task to test an event status, without being forced into a wait state, an Event Status call is provided. The Event Status call requests the Alert Call Processor to interrogate the status of a specified event, and provide this status information to the calling task. The Event Status call is functionally defined in Figure 4-17.

Finally, a mechanism is required to set/reset the status of individually alerted events and program flag WIs. This capability is provided to allow flexibility in the utilization of Alerts and in the case of intertask communication. The mechanism is invoked through an Event Set call. This call requests that a specified event is set or reset as specified in the call. The Event Set call is functionally defined in Figure 4-17.

The following summary comments can be made concerning the event processing logic.

- 1) In general, a task can place only itself into the wait state; not another task. The only exception to this rule is that at the time of a Task Schedule call (reference Section 4.4.2.1), a task can specify Wait Items which must be satisfied prior to dispatching the requested task's TQI.
- 2) A task can only be in one wait state at a time. However, that wait can be contingent on multiple events.
- 3) A wait condition can only be satisfied when the specified event occurs.
- 4) It is automatically assumed by ACES that on a Wait call, the "event status" (reference Figure 4-16), is initialized unsatisfied. However, if the Wait call specifies an alerted event, the event status is initialized to the current status of the alert.

- 5) Each application task can define its own set of program flags by the Alert Request call mechanism.
- 6) When a task makes an Alert Request, the alert name must be unique.
- 7) Any task can set, interrogate, or cancel any Alert File Item so long as it knows the alert name.
- 8) ACES will not monitor or record occurrences for any events not defined via an Alert Request or Wait Items.
- 9) No event history is kept prior to the Alert Request or Wait call.
- 10) Event monitoring (the alert mechanism) is a passive activity. That is, ACES doesn't notify any application task of event occurrences. The application task must interrogate, via the status request, the Alert File Item to determine its current status.

4.5.2 Event Processing Components

The ACES Event Processing logic has been subdivided into the six functional areas listed below:

- o Wait Call Processor
- o Wait File Processor
- o Wait Event Processor
- o Alert Call Processor
- o Alert File Scan Subroutine
- o Alert Event Processor

A functional description of each of these areas is presented in the remaining subsections of Section 4.

4.5.2.1 Wait Call Processor

The Wait Call Processor is entered in response to a Wait Call Request. Figure 4.19 presents a functional flow diagram of the Wait Call Processor logic.

WAIT CALL PROCESSOR

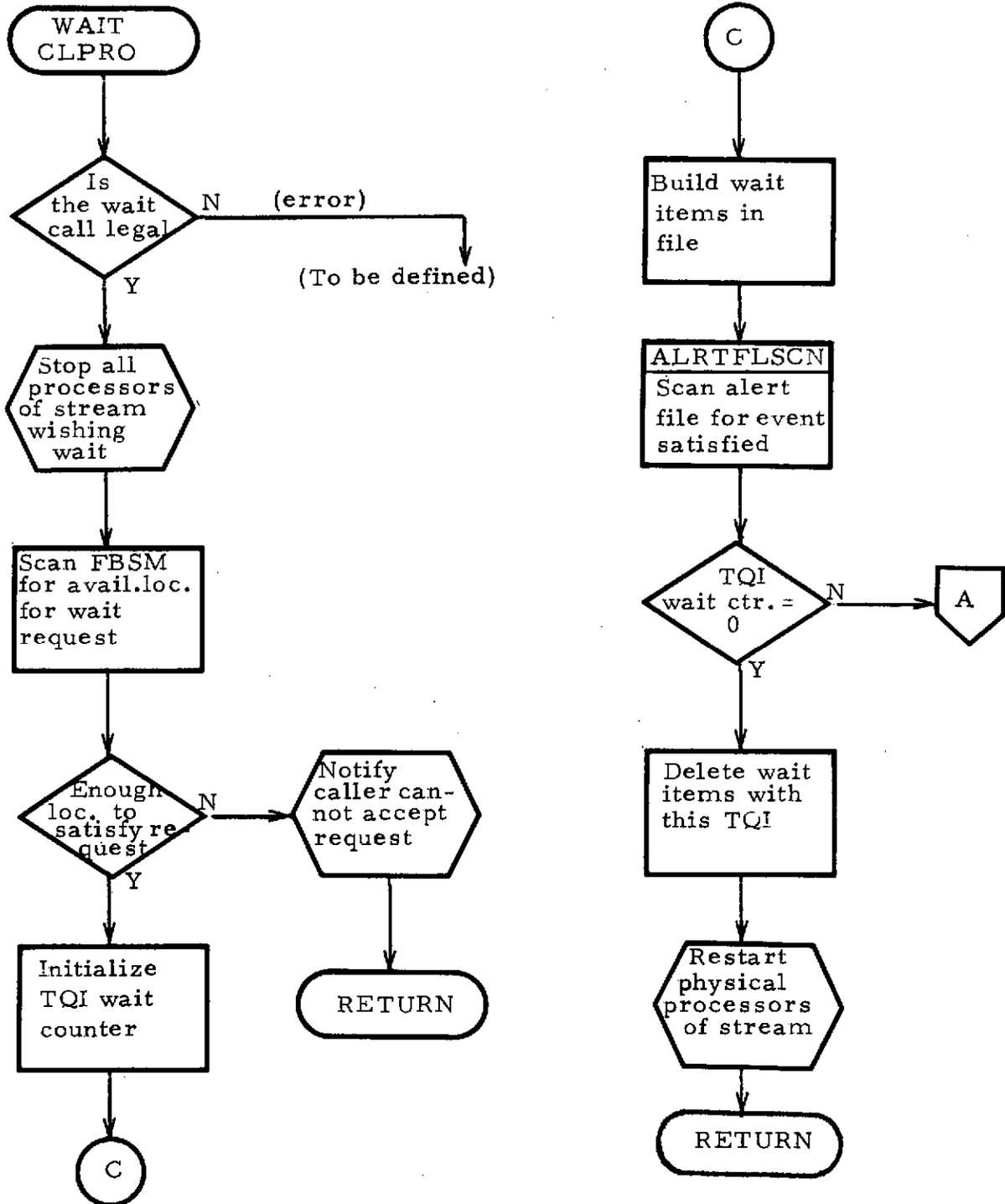


Figure 4-19

WAIT CALL PROCESSOR
(continued)

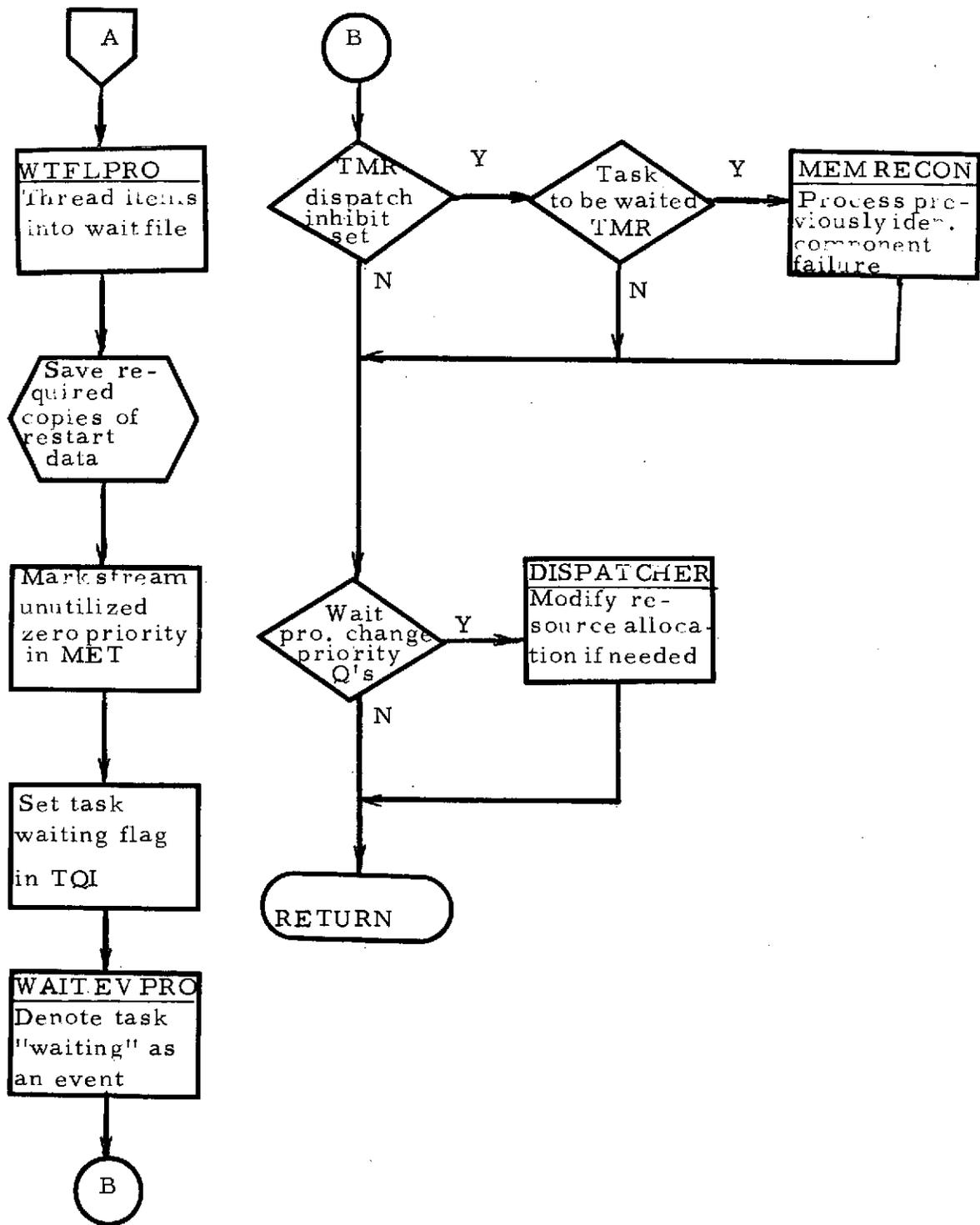


Figure 4-19
(continued)

The Wait Call Processor is the ACES program responsible for responding to an application task Wait Call Request. When entered, the Wait Call Processor confirms the validity of the Wait call data and determines if there is sufficient space in FM for the Wait Items. If not, the caller is notified that the Wait Request cannot be processed at this time.

If sufficient space is available, the Wait Items are built in FM and the TQI Wait Counter is initialized. If the Wait call specifies an alerted event the Alert File is scanned so that the event status of the WI can be initialized to the current status of the Alert. If the Wait call does not specify an alerted event, the event status is assumed to be unsatisfied.

If all of the Wait Items specified for the TQI were found satisfied by the Alert File scan, the Wait Items are deleted and the physical processor(s) restarted.

If any of the Wait Items are still not satisfied, the Wait File Processor is called to thread the built Wait Items into the Wait File. (Previous to this, the FM had been a temporary holding location for the Wait items.) The TQI is marked waiting in the Priority Queue to prevent it from being dispatched. Since a task entering the wait state is itself a definable event, the Wait Event Processor is called.

If the task which is placed in the wait state has a TMR criticality, and if the TMR Dispatcher Inhibit Flag is set, the Memory Reconfigurator is called to process a previously recognized triad memory module failure.

The final function of the Wait Call Processor is to call the Dispatcher to take advantage of any freed resources.

4.5.2.2 Wait File Processor

Figure 4-20 presents a functional flow diagram of the Wait File Processor. The Wait File Processor is entered from the Wait Call Processor or the Task Scheduler, and its primary function is to thread a TQI's WIs into the wait file, opening the file if necessary.

4.5.2.3 Wait Event Processor

The Wait Event Processor is entered in response to a definable ARMMS event. Figure 4-21 presents a functional flow diagram of the Wait Event Processor logic.

WAIT FILE PROCESSOR

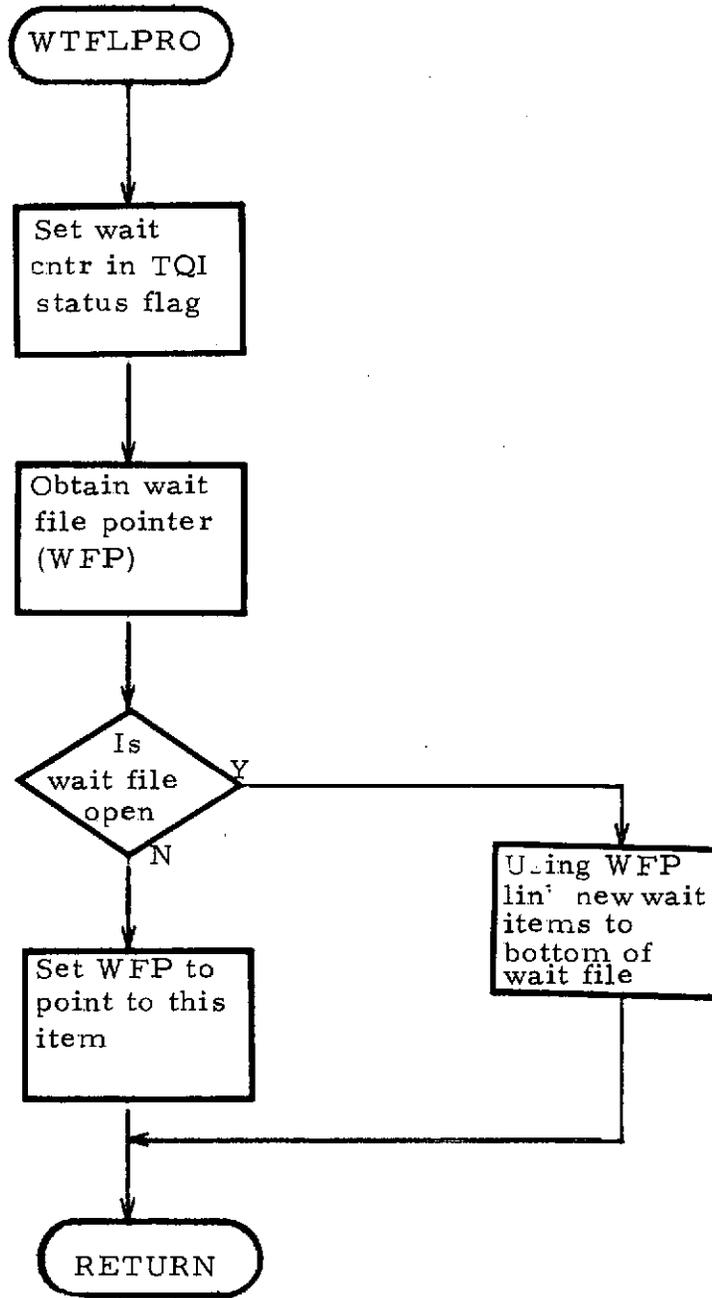


Figure 4-20

WAIT EVENT PROCESSOR

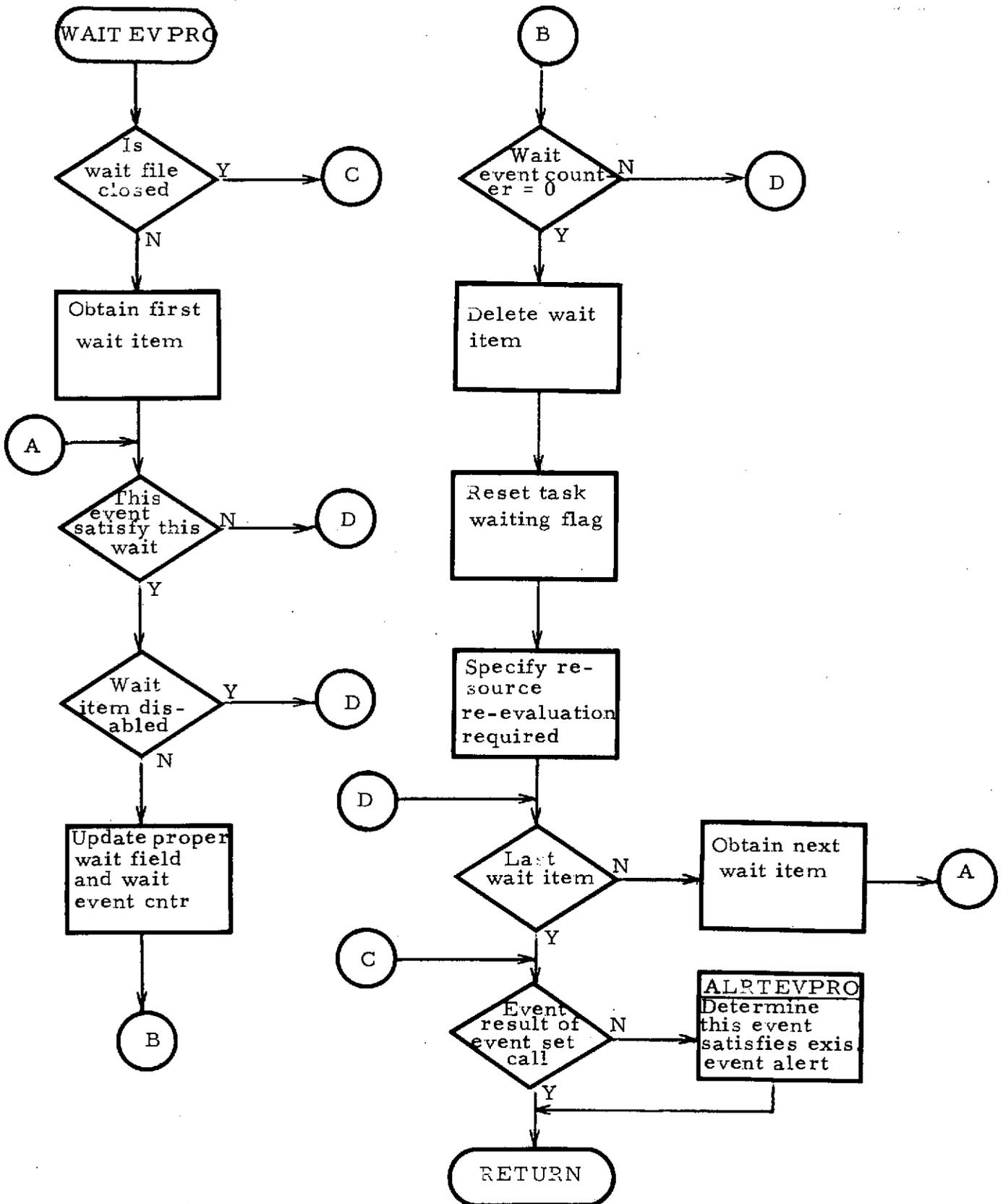


Figure 4-21

The Wait Event Processor is responsible for determining if an event satisfies a waiting TQI event specification or Alert File Item.

When entered, every WI, which is not disabled, and AFI is examined to determine if the occurring event requires that an update be made to the event status field of the WI or AFI. Whenever the wait events associated with TQI is zero, the TQI's Wait Items are deleted, the waiting flag in the TQI is reset, and a flag is set specifying that a resource re-evaluation is required.

If the event was the result of an Event Set Call, the Alert File is not scanned as part of the Wait Event Processor since it has already been updated by the Alert Call Processor.

4.5.2.4 Alert Call Processor

The Alert Call Processor is entered as a result of one of the four Alert calls; Request, Cancel, Set, or Status. Figure 4-22 presents a functional flow diagram of the Alert Call Processor.

Once entered, the Alert Call Processor decodes the call type and determines the legality of the call data. If the call was an Alert Request Call, the Alert Call Processor will construct an AFI and store it in Alert File Memory.

If the call was an Alert Cancel call, the Alert Call Processor will scan the AFM for the named AFI and, when found, will delete the AFI from AFM. The processor will then call the Wait Event Processor to denote the event of space availability in the AFM.

If the call was an Event Set call, the Alert Call Processor sets the event status of the named AFI to the condition specified in the call data and then calls the Wait Event Processor to determine if any WIs are waiting for this particular status change, if the event was a Program Flag.

Finally, if the call was an Event Status call, the Alert Call Processor scans the AFM to identify the designated AFI and transmits the event status of the identified AFI to the caller.

4.5.2.5 Alert File Scan Subroutine

Figure 4-23 presents a functional flow diagram of the Alert File Scan subroutine. This subroutine is entered from the Wait Call Processor and is responsible for processing alerted events for a TQI

ALERT CALL PROCESSOR

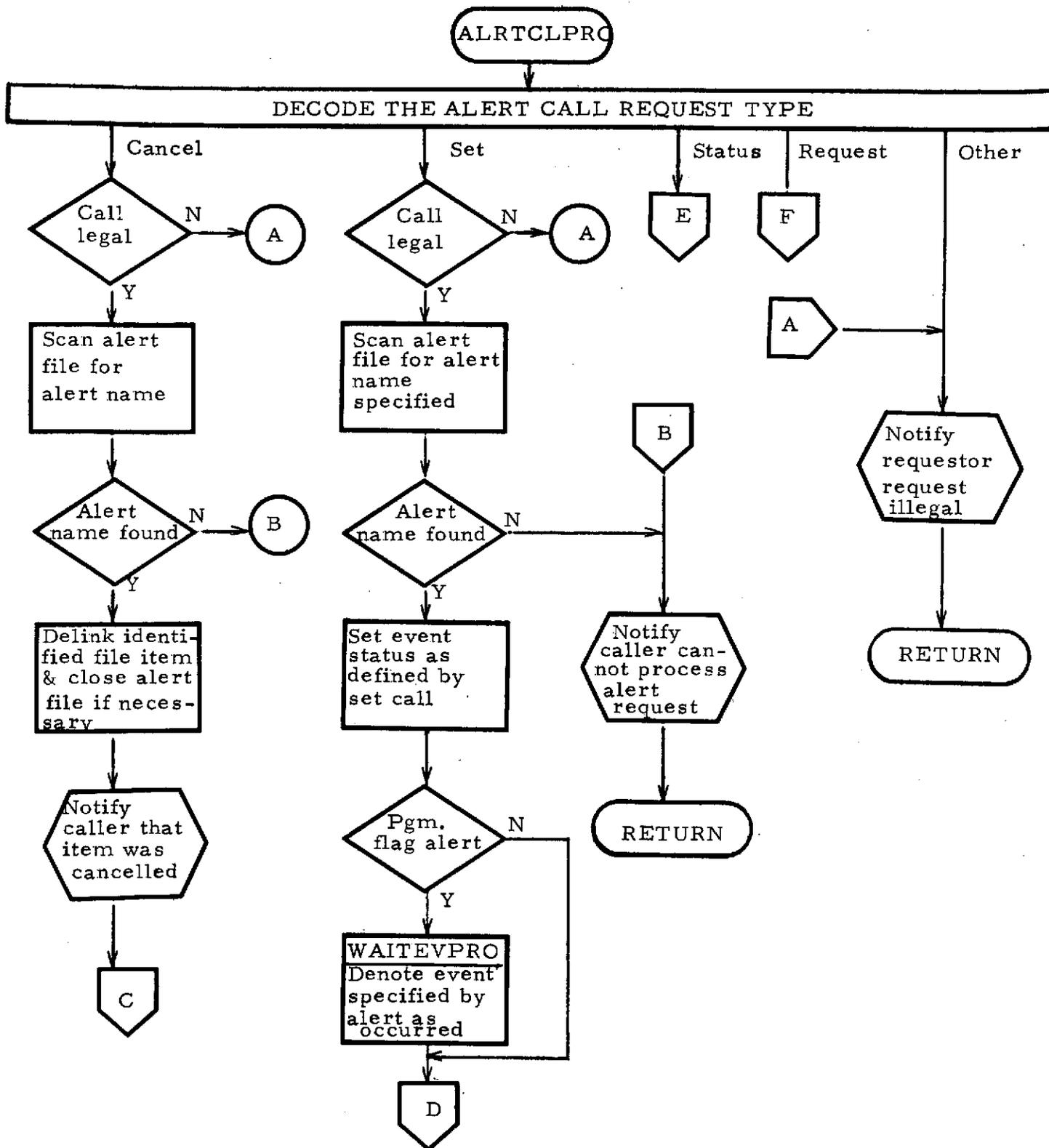


Figure 4-22

ALERT CALL PROCESSOR
(continued)

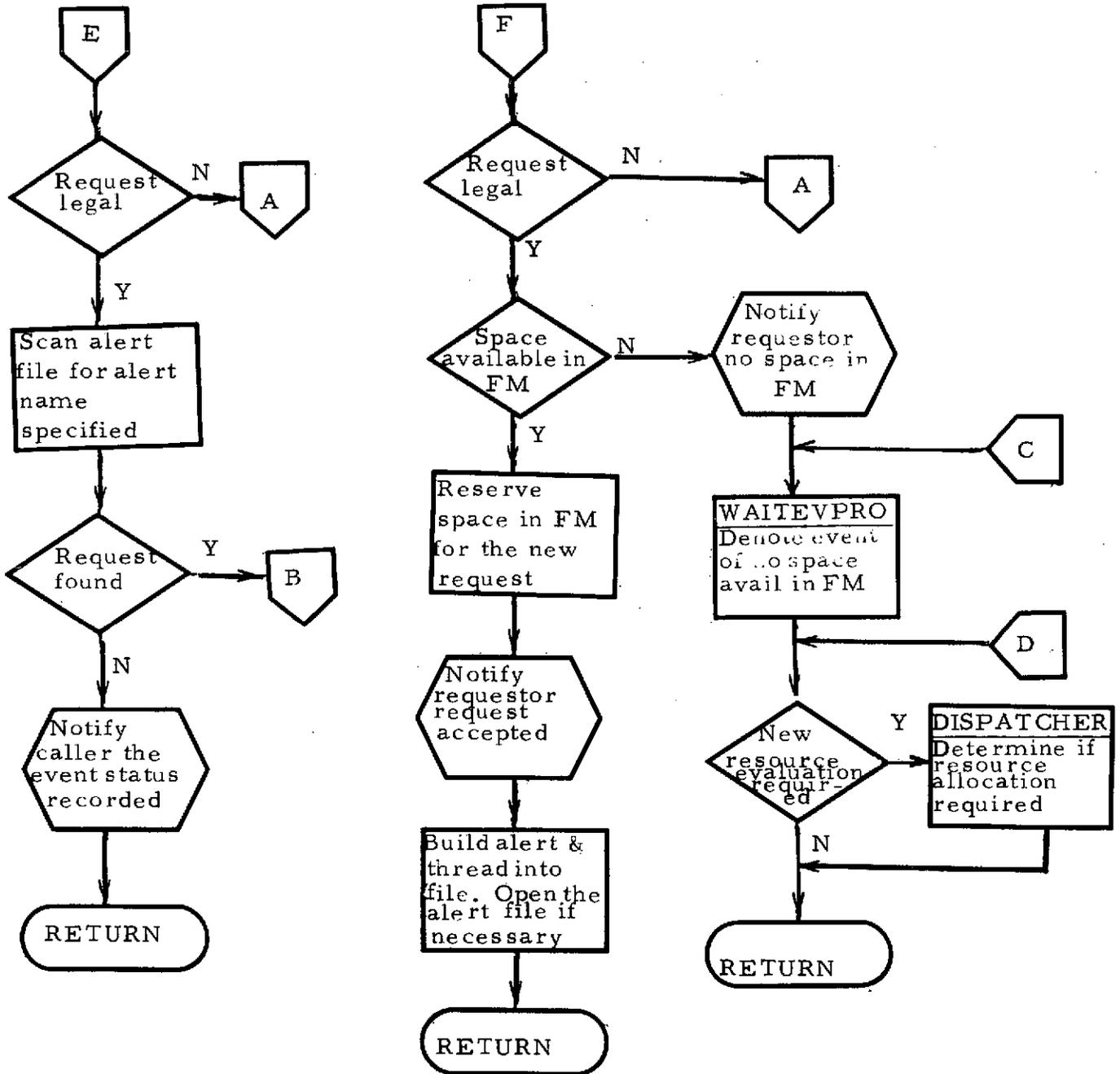


Figure 4-22
(continued)

ALERT FILE SCAN

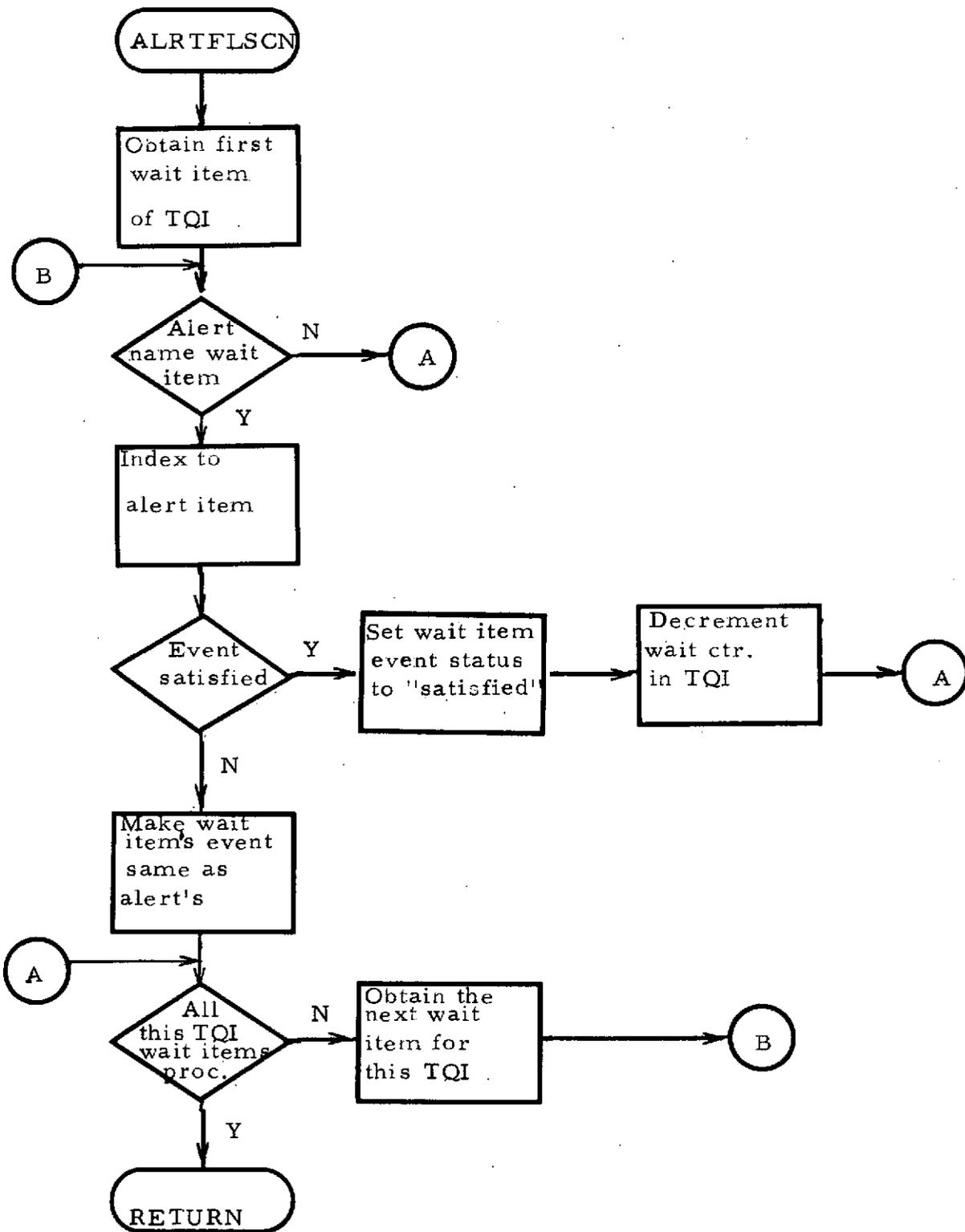


Figure 4-23

currently being processed by the Wait Call Processor. The Wait Call Processor calls the Alert File Scan program when a task makes a Wait Call Request or the time-to-execute has expired for a TQI and the TQI had wait parameters associated with it. In both cases, the Alert File is searched in order to initialize the event status fields of the TQI's WI which specified an alerted event.

The Alert File Scan subroutine determines if a Wait Item specifies an alerted event as the wait event. If so, the program checks the event status of the alert specified. If the event specified by the alert is satisfied, the WI is marked satisfied and the wait counter decremented. If the event is not satisfied, the AFI's condition code and subfields are copied to the WI's condition code and subfields. This allows the WI to monitor for the event of interest directly.

4.5.2.6 Alert Event Processor

Figure 4-24 presents a functional flow diagram of the Alert Event Processor. This routine is entered from the Wait Event Processor and scans the AFM to determine if a particular event is being monitored by an AFI. If a matching event is identified in AFM, the AFI's event status field is set to correspond to the status of the occurring event.

4.5.2.7 System Wait Call

Figure 4-25 presents a functional flow diagram of the System Wait Call subroutine. This subroutine is entered from the Memory Fail routine whenever a simplex or duplex stream encounters a fault when referencing a TMR triad memory module and a TMR task is active. In this case, the simplex or duplex task should be placed into the wait state waiting for the TMR task to terminate.

ALERT EVENT PROCESSOR

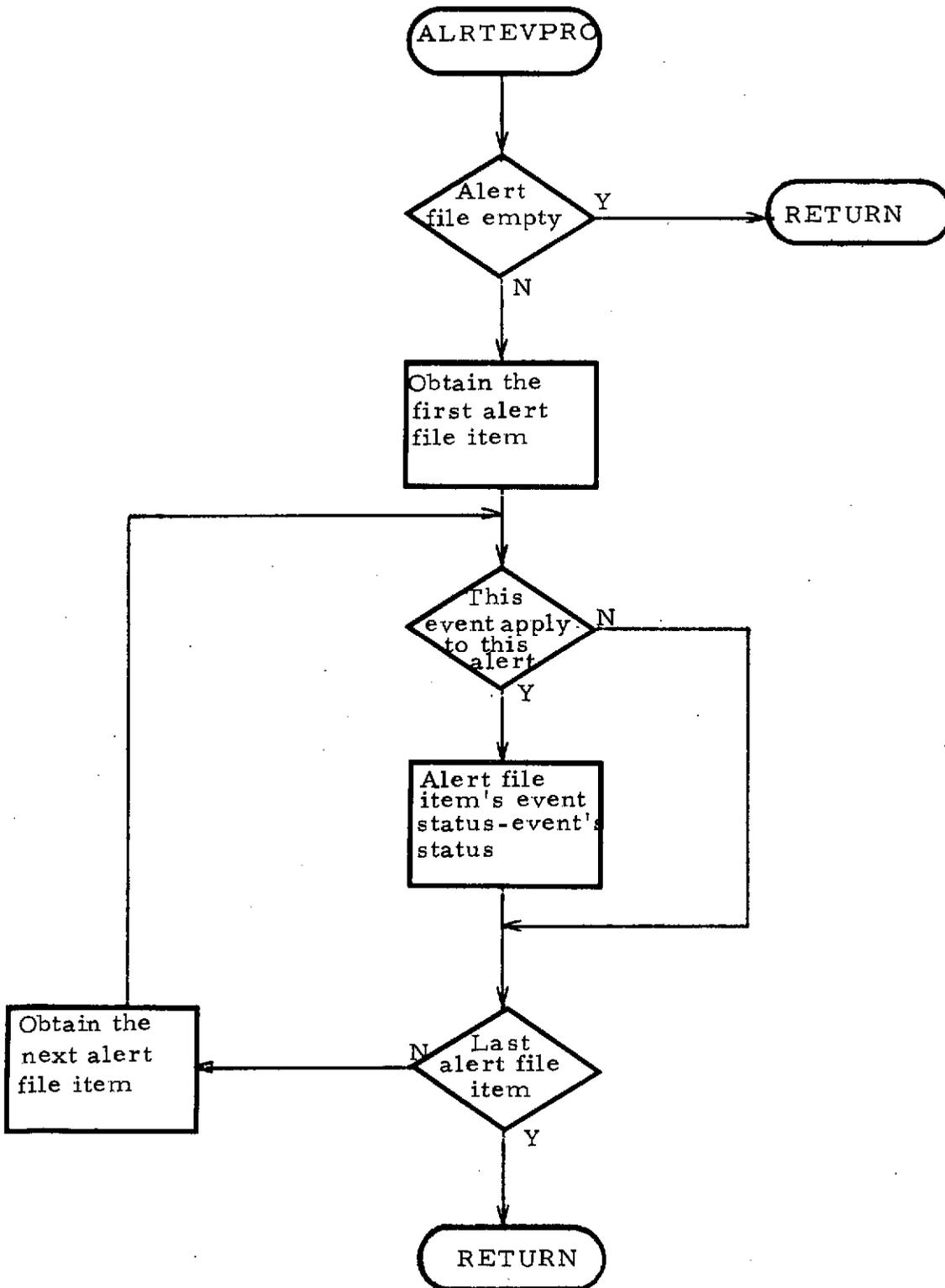


Figure 4-24

SYSTEM WAIT CALL

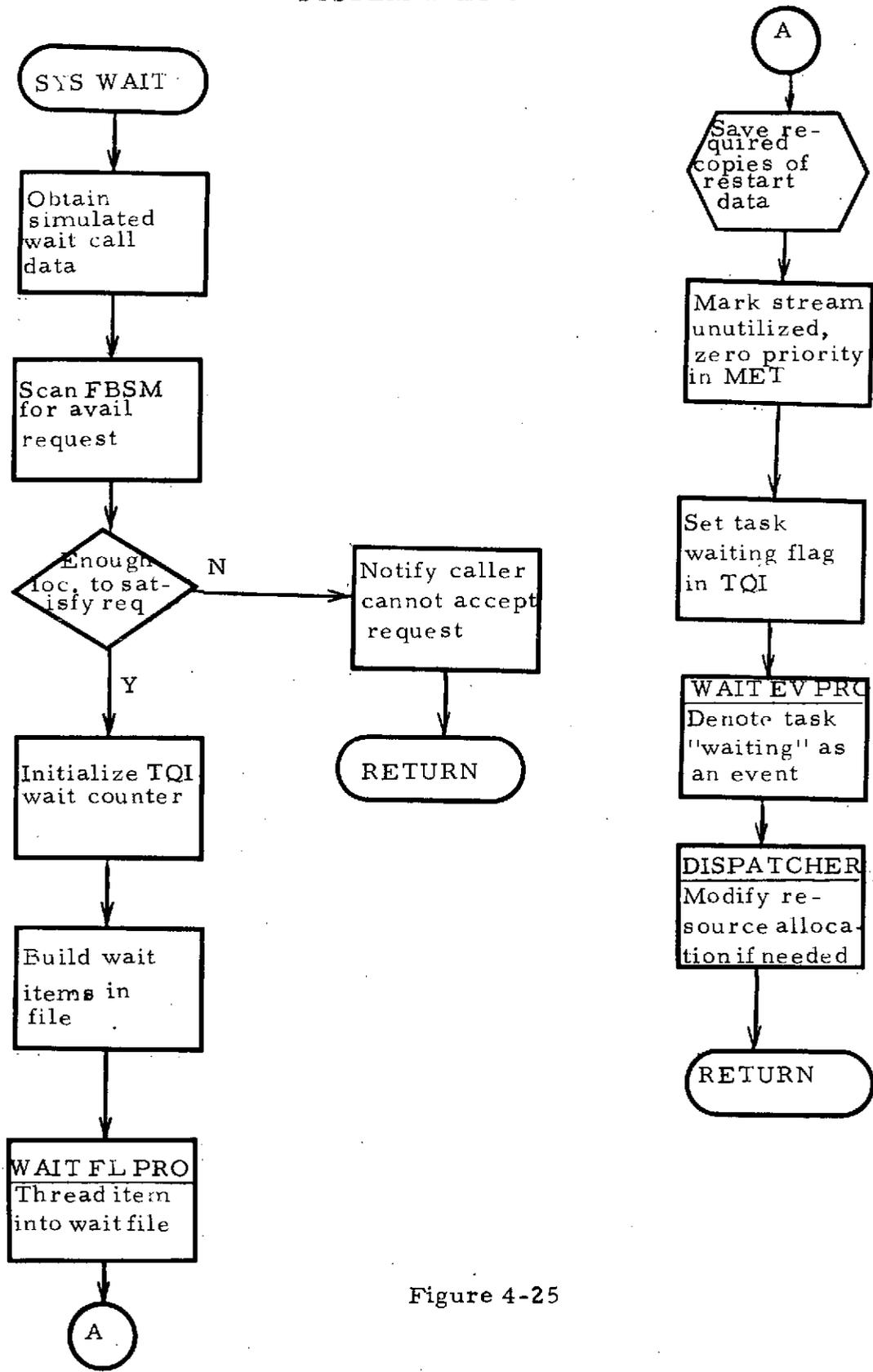


Figure 4-25

4.6 Resource Allocation and Control

4.6.1 Resource Control Overview

Resource allocation and control consists of those algorithms and design concepts required to control the allocation and utilization of available CPE's, IOP's, output voter/switches, system buses, and memory resources. As such, it is concerned with:

- o The construction of streams from available resources.
- o The maintenance of the resource pool.
- o The reservation of specific resources for diagnostic utilization.
- o The management of memory resources to insure that memory failures and any associated reduction in on-line memory resources remain transparent to application tasks.

The remainder of this subsection presents an overview of the following key areas:

- 1) Stream Resource Allocation and Control,
- 2) Diagnostic Resource Reservation, and
- 3) Memory Management.

4.6.2 Stream Resource Allocation and Control

Stream resource allocation and control consists of those algorithms and design concepts required to control the allocation and utilization of stream resources; i. e., IOPs, CPEs, output voter/switches, and system buses. In the current ACES design these concepts and algorithms are centralized in the Configurator and Stream Search subroutine and are controlled by the Dispatcher.

The Stream Search subroutine examines the resource pool and attempts to identify a set of components required by either a simplex I/O or processing stream as specified by the Dispatcher. To make the dominant search as efficient as possible, every attempt is made to utilize completely operational resources before resorting to partially operational resources. An example of a partially operational resource is a CPE with a

single port failure. So long as no attempt is made to use the failed port in the defined stream, the device can legitimately be utilized.

The Configurator is responsible for interconnecting the identified stream components into a stream with the criticality specified by the Dispatcher. It is the Configurator which issues the actual device interconnection commands.

If the Dispatcher identifies a need to interrupt lower priority streams, it is the responsibility of the Configurator to break the defined stream resources down into available spare components and to update the resource pool control tables; i. e., the UST, BSW, and RPC s.

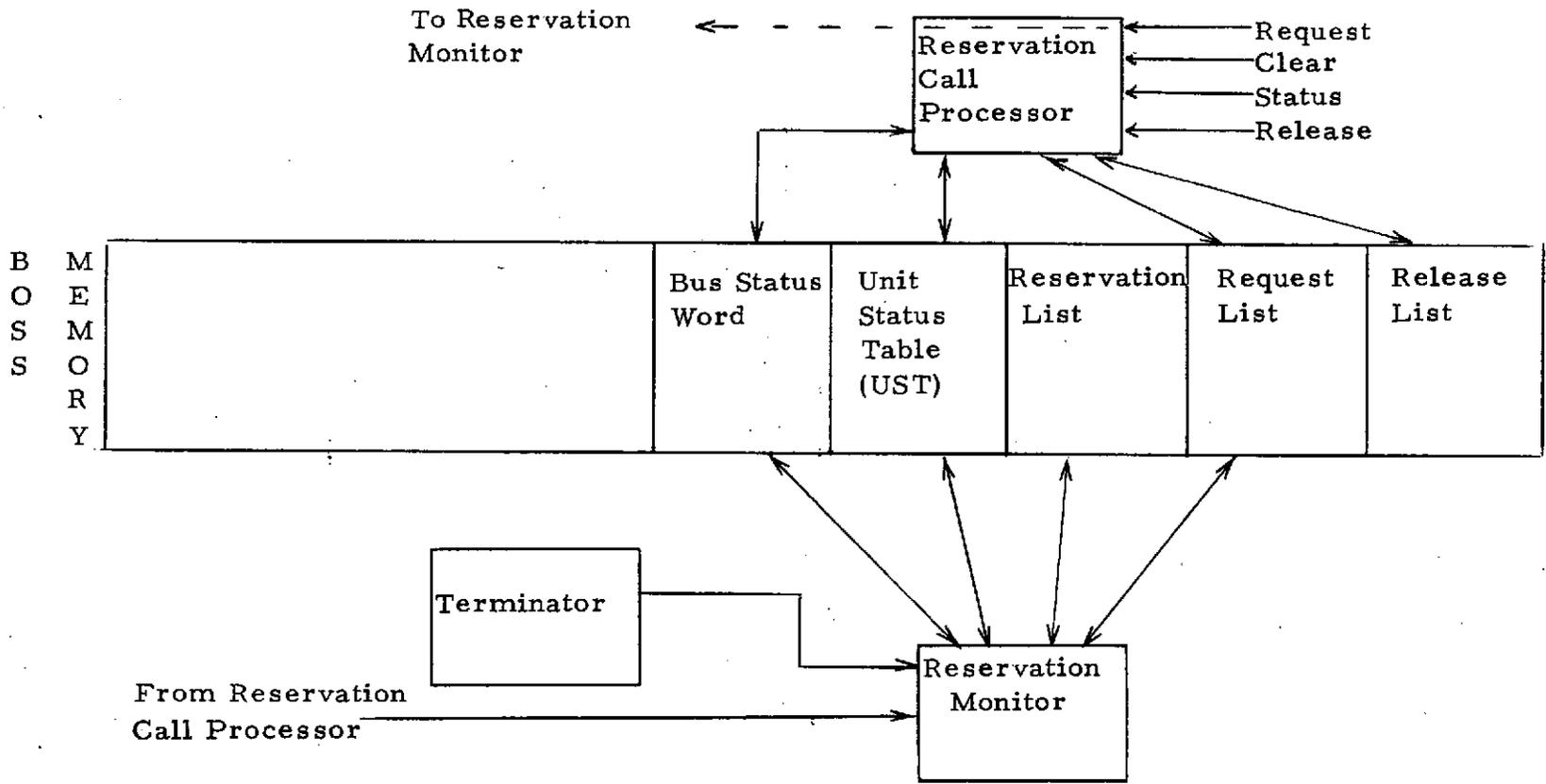
4.6.3 Diagnostic Resource Reservation

Diagnostic resource reservation consists of those algorithms and design concepts required to allow the diagnostic software to request that the ACES reserve specific system resources for diagnostic utilization. Figure 4-26 presents an overview of the reservation processing logic and defines its communication interfaces at the functional level. The diagnostic reservation concepts are centralized in the Reservation Call Processor and in the Reservation Monitor which is accessed by both the Terminator and the Dispatcher.

The Reservation Call Processor can accept and process four reservation calls:

- 1) Reservation Request Call - requests that the ACES reserve a specific set of resources for diagnostic purposes.
- 2) Reservation Clear Call - cancels a previously initiated reservation request.
- 3) Reservation Status Call - requests that the system interrogate the current status of a reservation request and return to the caller a definition of resources currently reserved.
- 4) Reservation Release Call - releases the specified resources, returning them to an available status.

RESERVATION PROCESSING (FUNCTIONAL DEFINITION)



3-89

Figure 4-26

Figure 4-27 presents a definition of the functional components of each of the four reservation calls.

The Reservation Call Processor uses the data associated with a reservation request call to build a Reservation Request List. This Request List defines to the Reservation Monitor the resources required. The Reservation Monitor then accumulates the desired resources from the resource pool, identifying and reserving them as they become available spares, and builds the Reservation List, which identifies the units so far reserved.

The Release List is constructed and utilized by the Reservation Call Processor as a result of a Reservation Release call. It defines the individual resources to be released for general application task utilization.

The following summary comments are pertinent to the diagnostic resource reservation concept:

- 1) Diagnostic resource reservation allows the diagnostic software to reserve specific resources for its exclusive use.
- 2) The Reservation Monitor will attempt to satisfy reservation requests for a specific resource prior to satisfying a request for an arbitrary resource of a particular type.
- 3) For the sake of simplicity, the Reservation Monitor does not look ahead in order to evaluate the possibility of completing a reservation. So long as one stream is executing, the system assumes that it may still be possible to satisfy an incomplete reservation.
- 4) Only one reservation request will be accepted at a time and a reservation request, once completed, will be held until it is specifically cleared via a reservation clear call.
- 5) If a requested resource is failed, it will be marked as such in the reservation list.
- 6) While only one reservation can be processed at a time, the total number of reserved resources is cumulative as sequential reservations are made until a reservation release is commanded.

RESERVATION CALLS (FUNCTIONAL DEFINITION)

I. Reservation Request

Call Byte Count	Call Type (Res. Request)	Memory Buses								I/O Buses								Processors				IOPs			
		MO1	MO2	MO3	MO4	MI1	MI2	MI3	MI4	IP1	IP2	IP3	IP4	PO1	PO2	PO3	PO4	P1	P2	P3	P4	IOP1	IOP2	IOP3	IOP4
		#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

= Module Number, or
"Not Used" indicator, or
"Any Spare" indicator

Output VS				Output Bus			
VS1	VS2	VS3	VS4	OB1	OB2	OB3	OB4
#	#	#	#	#	#	#	#

II. Reservation Clear

Call Byte Count	Call Type (Res. Clear)

III. Reservation Status

Call Byte Count	Call Type (Res. Status)

IV. Reservation Release

Call Byte Count	Call Type (Res. Release)	Memory Buses								I/O Buses								Processors				IOPs			
		MO1	MO2	MO3	MO4	MI1	MI2	MI3	MI4	IP1	IP2	IP3	IP4	PO1	PO2	PO3	PO4	P1	P2	P3	P4	IOP1	IOP2	IOP3	IOP4
		#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

= Module Number, or
"Not Used" Indicator

Output VS				Output Bus			
VS1	VS2	VS3	VS4	OB1	OB2	OB3	OB4

Figure 4-27

4.6.4 Memory Management

Memory management consists of those algorithms and design concepts required to control the utilization of the ARMMS memory resources. These concepts are centralized in the Memory Reconfigurator which is driven by the error detection software when a memory failure or addressing error is detected.

The memory paging techniques within the ACES framework are somewhat unique to ACES. Therefore, a discussion of the paging concepts and techniques is necessary. However, it is first necessary to summarize the paging groundrules and definitions adopted during the current design of the ACES philosophy:

- 1) Associated with any specific task dictionary will be a specific set of logical memory modules and spare physical modules.
- 2) A spare physical memory module can only replace a failed logical memory module. It will not be used to obtain additional logical memory modules for the current dictionary.
- 3) Paging will only be allowed upwards within any given dictionary period. That is, a simplex stream can only be paged into a simplex, duplex, or TMR pageable logical module and, once paged, utilizes the whole logical module. A duplex stream can only be paged into a duplex or TMR pageable logical module and a TMR stream can only be paged into a TMR pageable module.
- 4) A corollary to rule 3 is that pageable logical units are never broken into new logical units of lower criticality.
- 5) If it is desirable to have access to more memory pages of a specific criticality during a dictionary period than there are available logical modules of that criticality, then at least one logical module of that or a higher criticality must be pageable.
- 6) The system will always drop to a dictionary of lower level if an attempt is made to address a non-available page and there is no logical module pageable with a criticality equal to or higher than the criticality of the requesting stream.

- 7) A triad failure, when it is being utilized by a duplex or simplex stream, will be treated as a normal duplex or simplex failure. No attempt will be made to dynamically modify the way in which the duplex or simplex stream is connected to the triad in an attempt to circumvent the failure.
- 8) If a component of a memory triad or pair fails and there are no spares currently available, the failing component will be taken off-line and the remaining components will be made available as spares.

4.6.4.1 Logical Module Concept

The ARMMS design allows BOSS to assign logical addresses to physical memory units. Multiple physical memory units may have the same logical address, when in TMR or duplex operation. This provides the mechanism for programs and data to be resident in two or more memory modules simultaneously. All physical modules which share the same logical address are conceptually grouped together and called a "logical module". Stream criticality defines the number of physical memory modules required by a logical module. A simplex stream normally utilizes logical modules composed of one physical module, duplex streams normally reference logical modules composed of two physical modules, and TMR streams may reference only a triad logical module.

There exists, however, a general need for any stream to have access to a logical module of equal or higher criticality. Global data, for example, will probably exist in a TMR logical module; however, simplex jobs must be capable of referencing it. To make this possible, whenever the same page must be referenced by multiple streams of differing criticalities, the page must reside in a logical module associated with the highest criticality stream referencing it.

The logical module concept is an integral part of the ACES paging technique. Whenever paging occurs, logical memory pages are placed into a logical memory module of the same criticality as the executing stream. Since lower criticality streams can reference higher criticality logical modules, but not vice versa, whenever an addressing exception (due to a logical address not being available) occurs, the paging algorithm first determines if the same logical page resides in a lower criticality logical module. If not, the page is brought from bulk memory to a logical module of proper criticality. If the page is already in main memory but in a logical module of less criticality, the page is copied from the lower criticality to the higher

criticality logical module and the lower criticality logical module is made a spare. This provides a mechanism for pages to "float" upwards to higher criticality logical modules as needed. This also insures that only one logical module contains a specific logical page at any given time.

4.6.4.2 Locked Logical Pages

It is often desirable to prevent certain logical pages from being removed from main memory due to their high frequency of use or because of critical response time requirements. Any global data logical pages are a clear example of such data. The concept of locked logical pages increases overall efficiency by eliminating paging time for certain logical address pages which have a high utilization factor.

Locked logical pages, once resident in main memory, are not allowed to be replaced by another logical page. Thus, once such a logical page becomes resident in main memory, it is locked there and, while it may be moved to a higher criticality logical unit, it cannot be returned to bulk store. All logical pages are initially brought in to main memory at phase initialization time or on an addressing exception the first time the page is referenced.

Locked logical pages are defined for the duration of a task dictionary. Whenever a new task dictionary is set up, a new set of locked pages can be defined to exist during that dictionary period.

4.6.5 Resource Control Components

To perform the functions described in the previous paragraphs, the following components have been defined:

- o Configurator
- o Stream Search Subroutine
- o Memory Reconfigurator
- o Reservation Call Processor
- o Reservation Monitor

A functional description of each of these components is presented in the following subsections.

4.6.5.1 Configurator

The Configurator is responsible for accepting and processing Dispatcher requests to construct or disassemble a specified stream. Figure 4-28 presents a functional flow diagram of the Configurator logic.

The data passed to the Configurator by the Dispatcher consists of a Configuration Stream Request Word (CSRW) which defines the function to be performed and identifies the MET entry which contains the module identification numbers of the stream components of concern.

When entered, the Configurator establishes the legality of the configuration request and determines whether it is required to assemble or disassemble a stream.

If a stream is to be disassembled, the BSW, UST, and RPCs are updated accordingly. If the function code specifies a stream assembly, the proper interconnection control commands are constructed and transmitted.

4.6.5.2 Stream Search Subroutine

The Stream Search subroutine is entered from the Dispatcher and is responsible for identifying the stream components required by a simplex stream of a specific type. Figure 4-29 presents a functional flow diagram of the Stream Search subroutine logic.

The Stream Search subroutine has three entry points, one for each defined type of stream; i. e., full processing, limited processing, and I/O. Once entered, the Stream Search subroutine will attempt to identify from the UST and BSW a set of usable and interconnectible simplex stream components of the type specified. It will not only identify spare major components, CPEs, IOPs, etc., but will also insure that it is possible to interconnect them properly into an operational simplex stream.

4.6.5.3 Memory Reconfigurator

Figure 4-30 presents a functional flow diagram of the Memory Reconfigurator logic.

The Memory Reconfigurator is called on a memory failure in order to attempt replacing the failed physical unit with a spare module. If no spare modules are available, an attempt is made to page the required logical page into an available module.

CONFIGURATOR

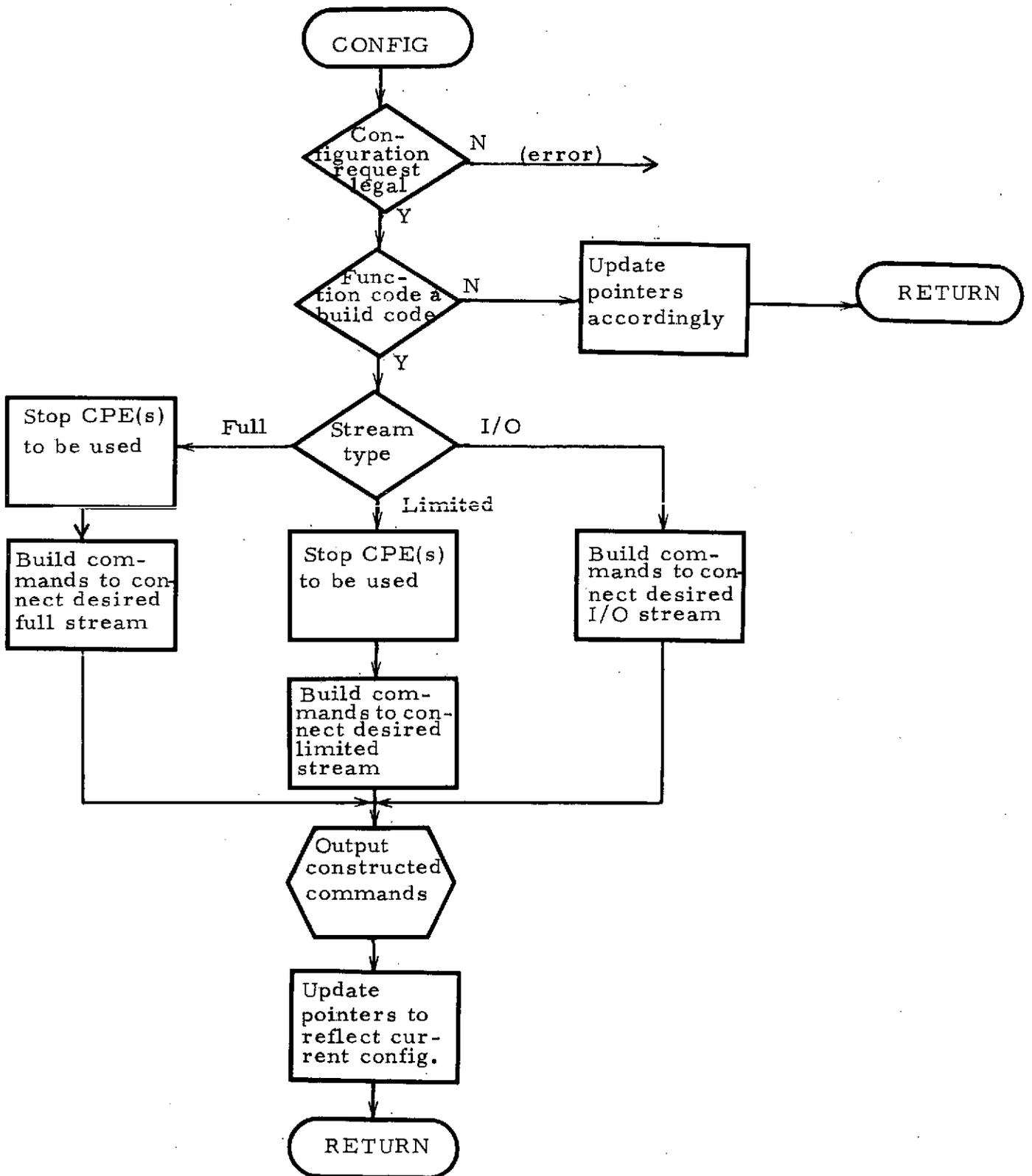


Figure 4-28

STREAM SEARCH SUBROUTINE

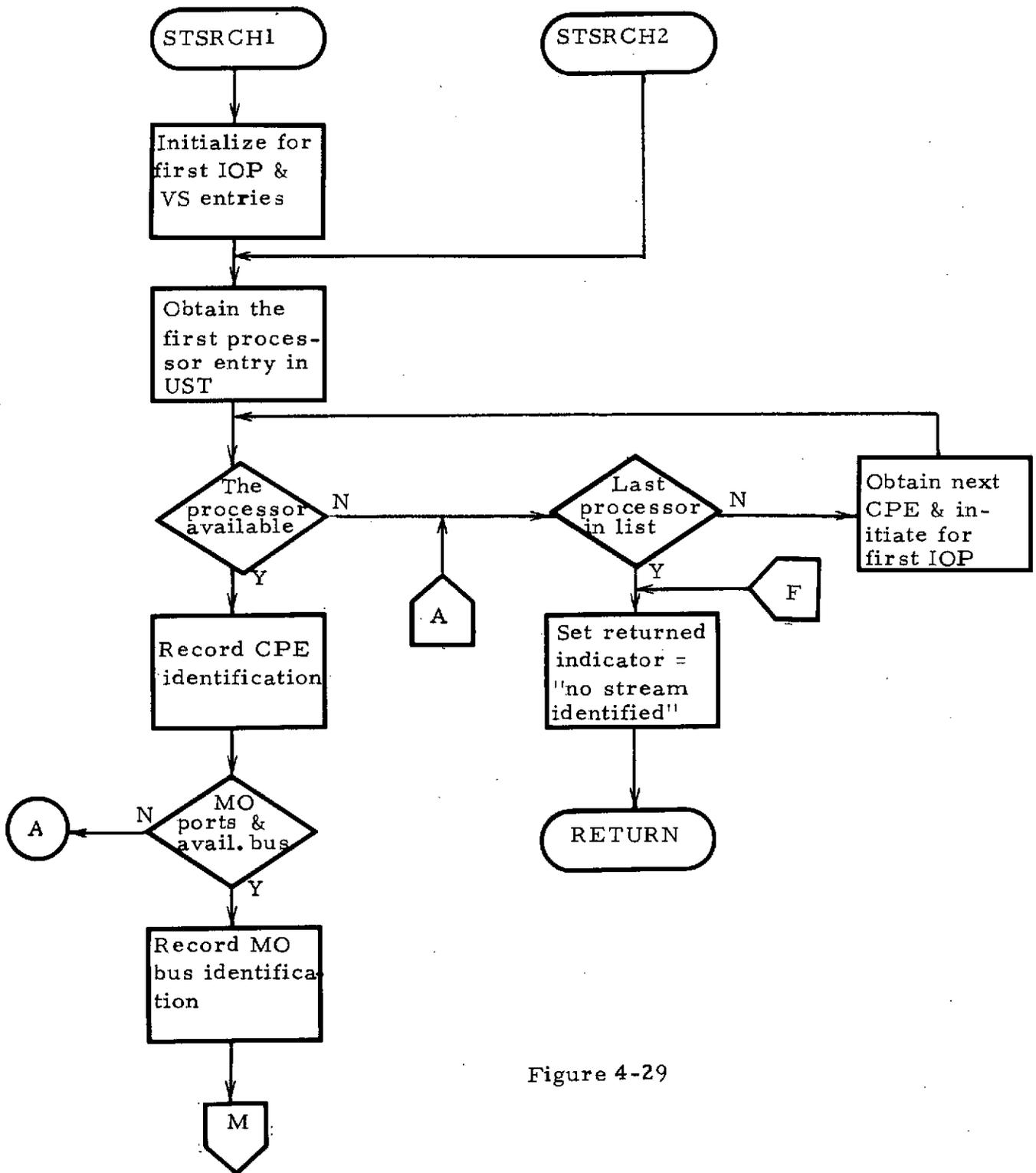


Figure 4-29

STREAM SEARCH SUBROUTINE
(continued)

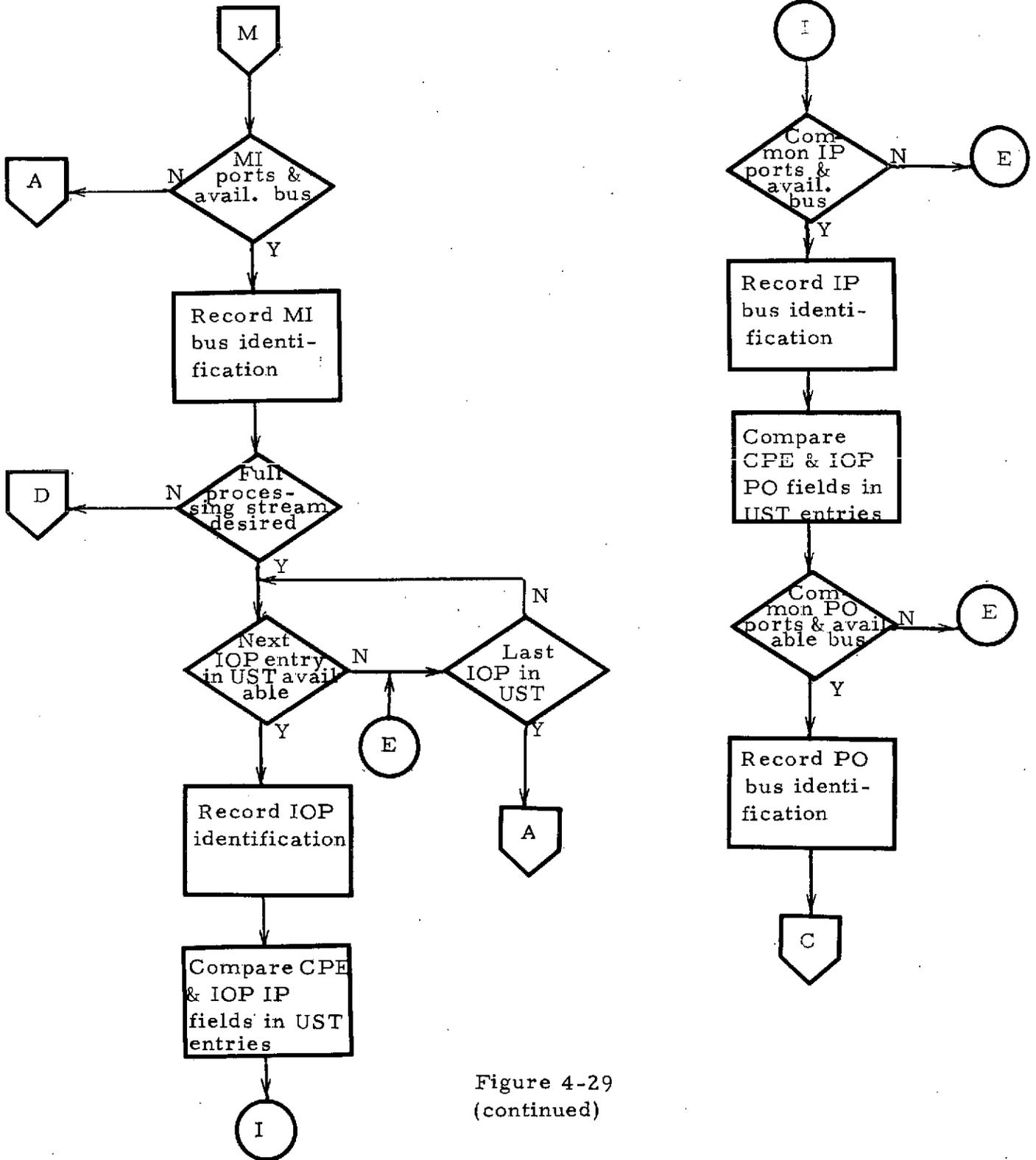


Figure 4-29
(continued)

STREAM SEARCH SUBROUTINE
(continued)

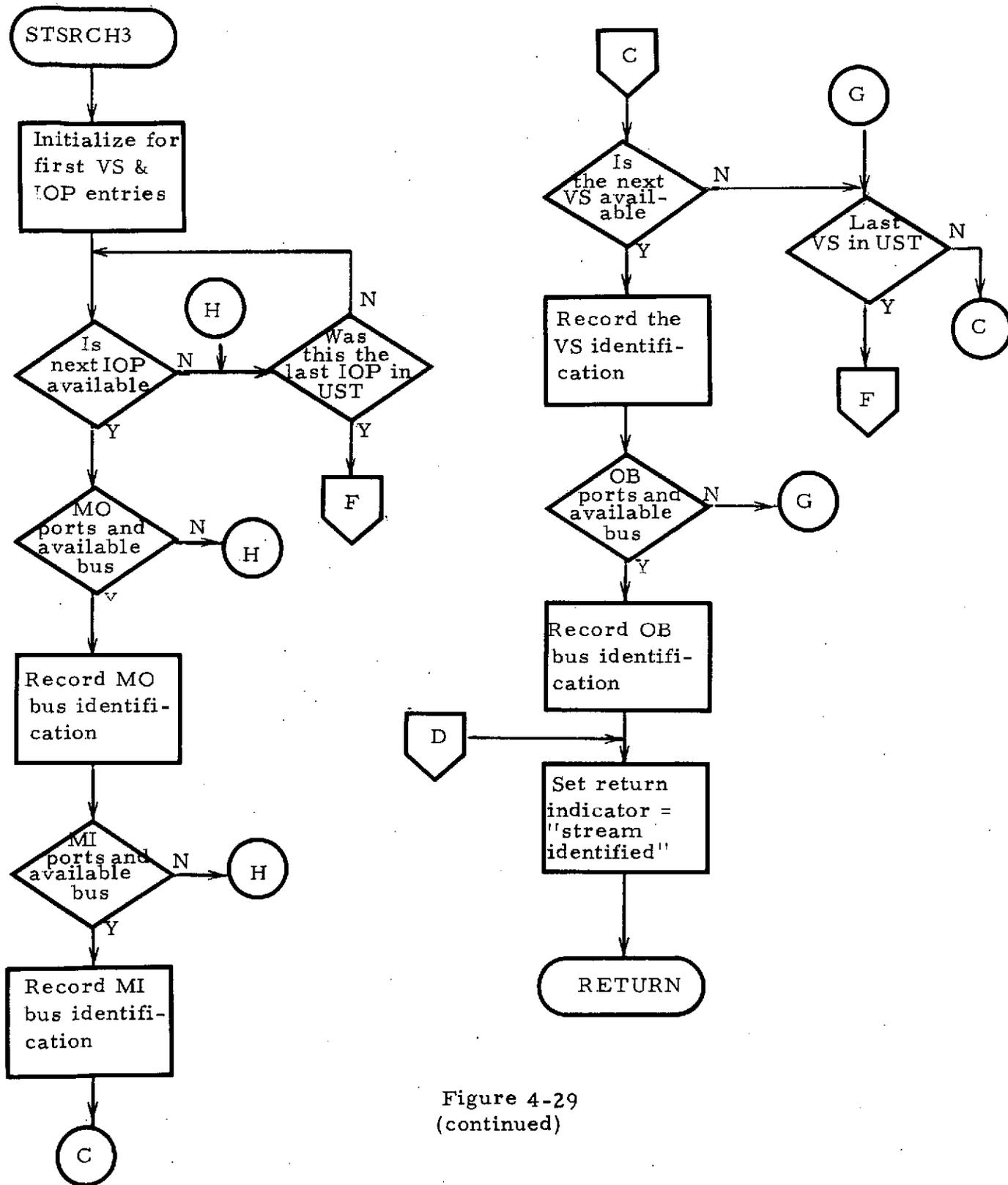


Figure 4-29
(continued)

MEMORY RECONFIGURATOR

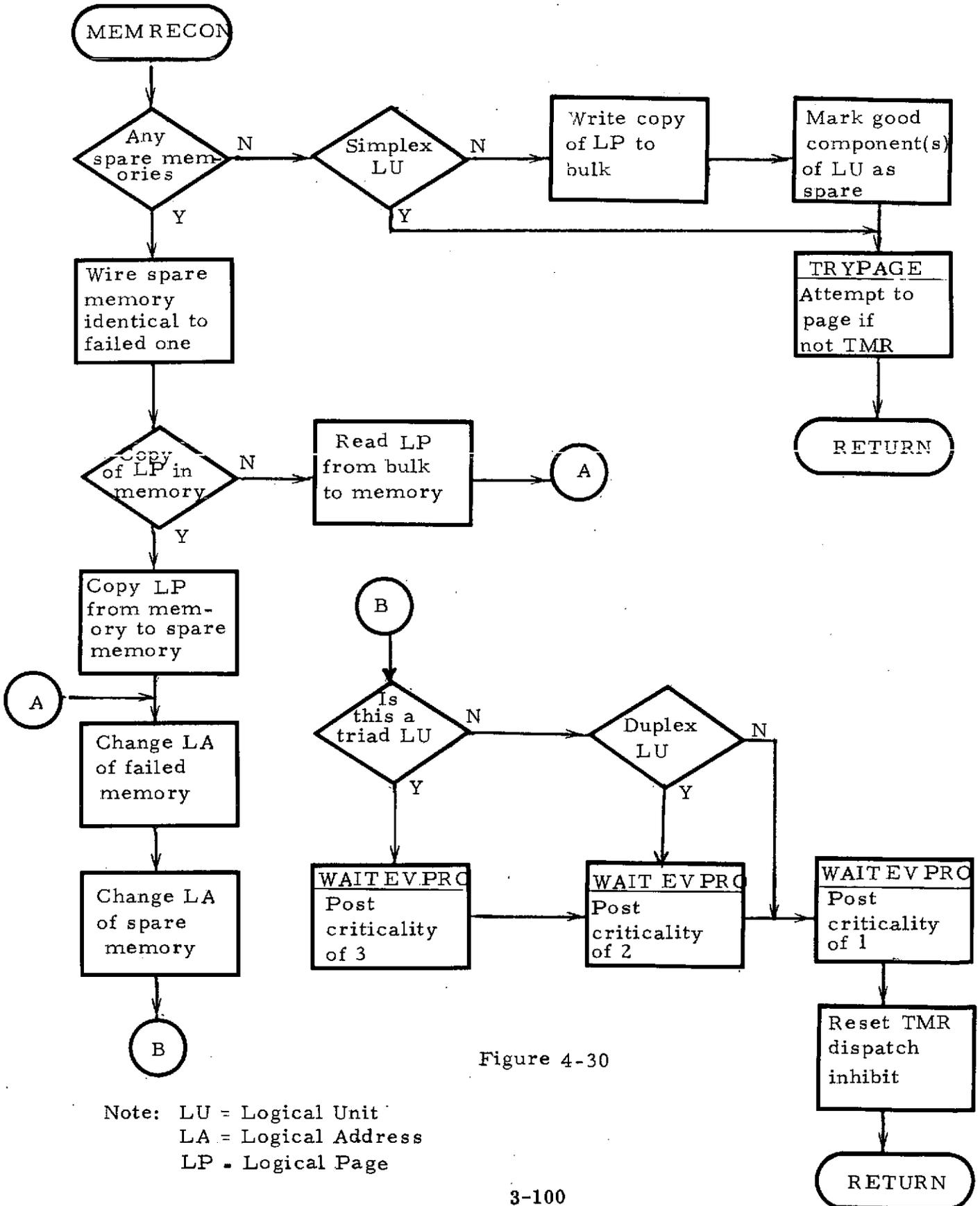


Figure 4-30

Note: LU = Logical Unit
 LA = Logical Address
 LP = Logical Page

The Memory Reconfigurator first determines if any spare memory modules are available. If no modules are available, a check is made to determine if the physical module which failed is the member of a simplex logical module. If so, TRYPAGE is called to attempt to page the desired logical module into main memory. If the module which failed is a member of a duplex pair or a TMR triad, a "valid" copy of the failing logical page is written to bulk as the latest available copy. Since no spare memory modules are available and since this logical module has a failed component, and as such is unusable as a module, the good modules are marked spare for possible later use. After the good modules are marked spare, TRYPAGE is called to attempt to page the needed logical page if the module was part of a duplex pair, to enable restart of the task.

If a spare memory is available, configuration commands are issued to the spare memory to connect it to the memory buses in the same way the failed physical module was connected. If a copy of the logical page is in main memory, the logical page is copied to the new physical module; otherwise, a copy is read from bulk storage into the module. Proper logical addresses are then set up for the memory modules involved in the replacement operation. The Wait Event Processor is then called to "post" possible tasks which are waiting for the logical page to become available, and the TMR Dispatch Inhibit Flag is reset to allow TMR dispatches to continue if the corrected failure was in a TMR triad.

4.6.5.4 Reservation Call Processor

The Reservation Call Processor is entered in response to a reservation call. Figure 4-31 presents a functional flow diagram of the Reservation Call Processor logic.

Four reservation calls are currently defined:

- 1) Reservation Request Call
- 2) Reservation Clear Call
- 3) Reservation Release Call
- 4) Reservation Status Call

The Reservation Request call establishes the reservation requirements and calls the Reservation Monitor in an attempt to immediately fulfill the specified requirements. A reservation request will only be accepted if a Reservation Clear call has been previously accepted.

RESERVATION CALL PROCESSOR

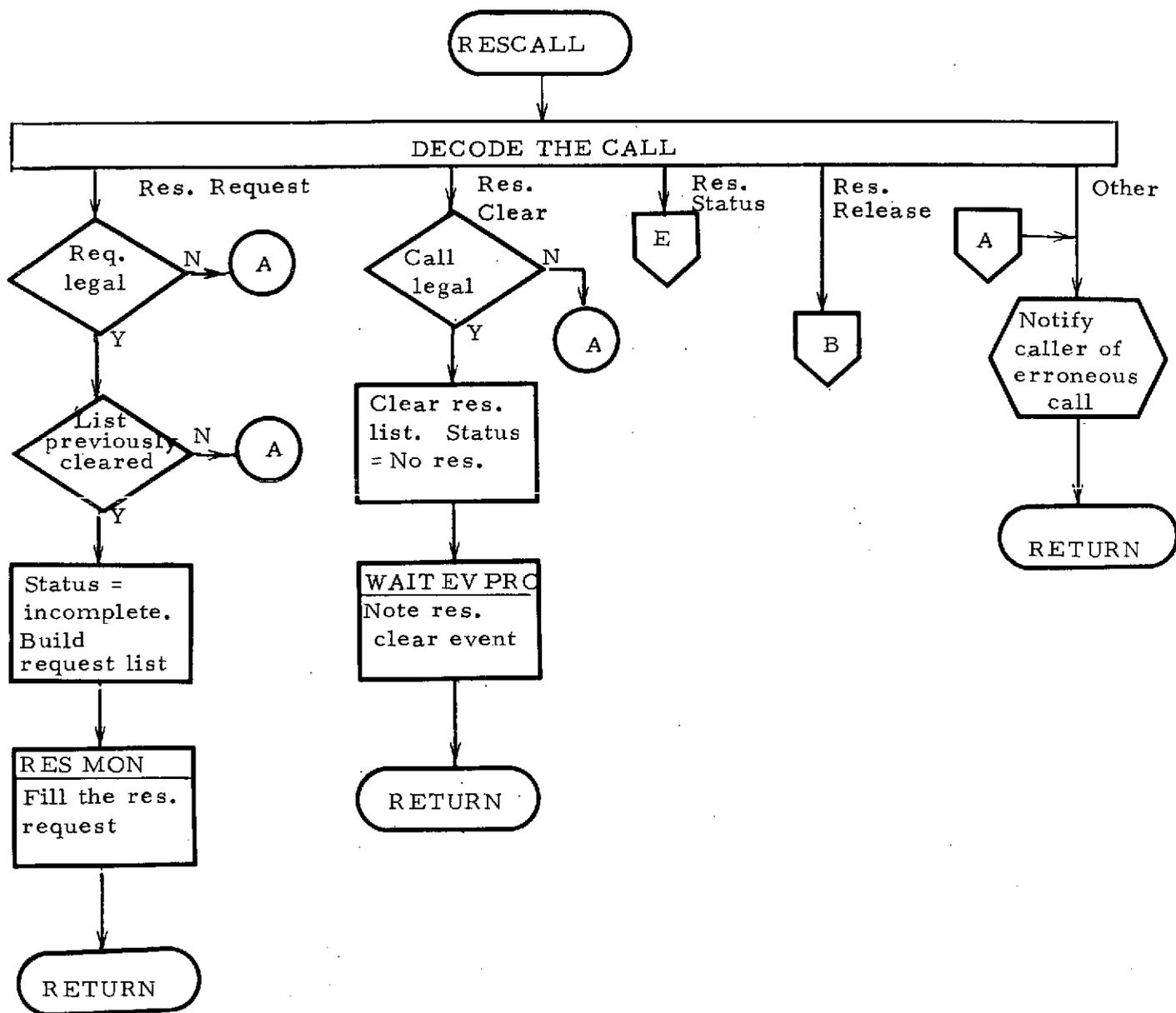


Figure 4-31

RESERVATION CALL PROCESSOR
(continued)

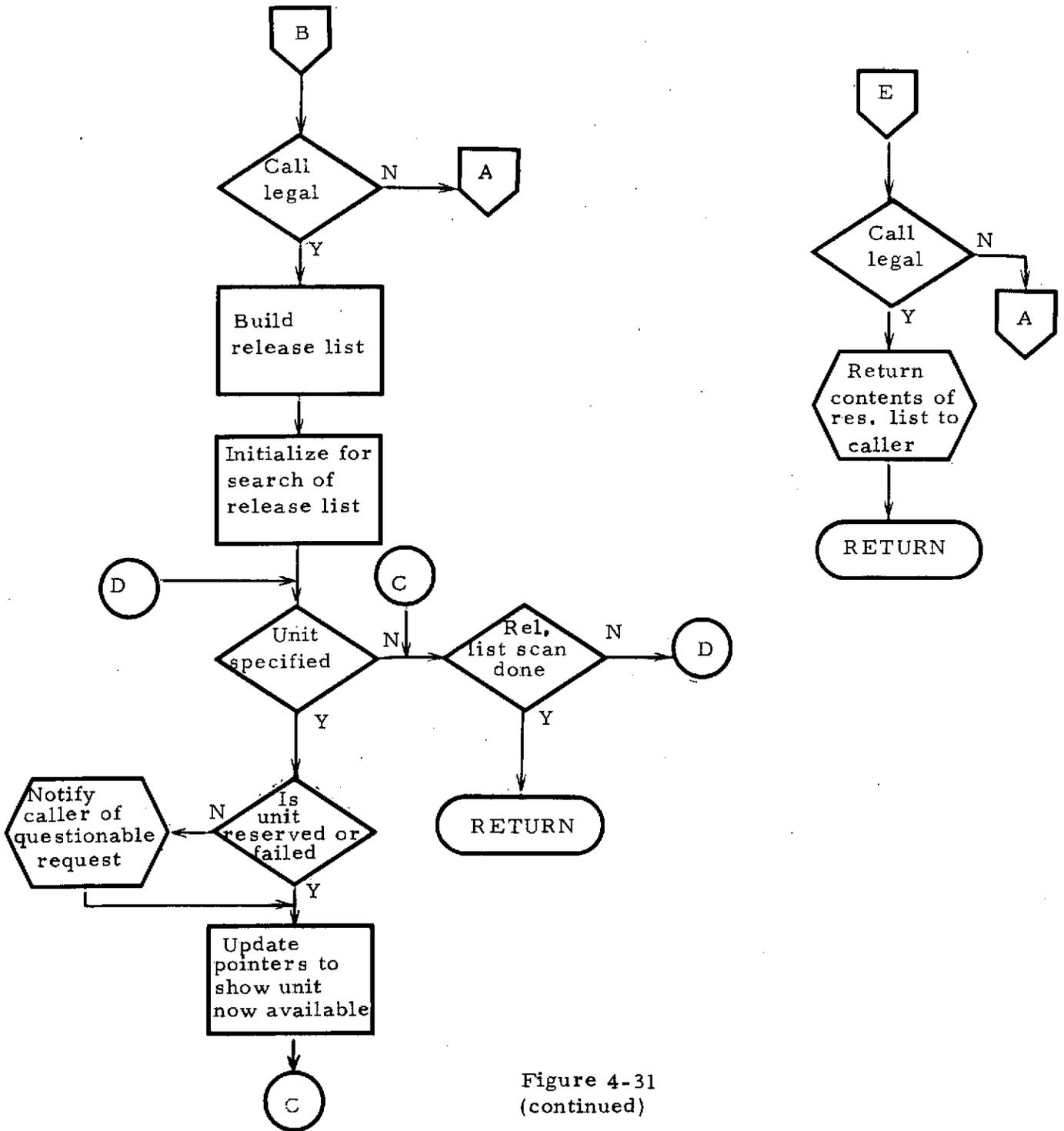


Figure 4-31
(continued)

The Reservation Clear call clears the Reservation List and calls the Wait Event Processor in order to determine if any task is currently waiting on the reservation clear event.

The Reservation Release call is used by the calling task to return specific resources to an available (spare) status.

The Reservation Status call is used by the calling task to request the current status of the Reservation List.

4.6.5.5 Reservation Monitor

The Reservation Monitor is currently entered from either the Terminator or the Reservation Call Processor. Figure 4-32 presents a functional flow diagram of the Reservation Monitor logic.

The Reservation Monitor is responsible for examining the resource pool control tables; i. e., UST and BSW, to determine if any available resources can satisfy a reservation requirement. If any of the examined units can satisfy a reservation requirement as specified by the Reservation Request List, the resource is "reserved", the unit identifier is added to the Reservation List, and the request is removed from the Reservation Request List.

The Reservation Monitor always attempts to satisfy reservation requests for specific resources prior to determining whether resources exist to satisfy a general request for any module of a particular type.

RESERVATION MONITOR

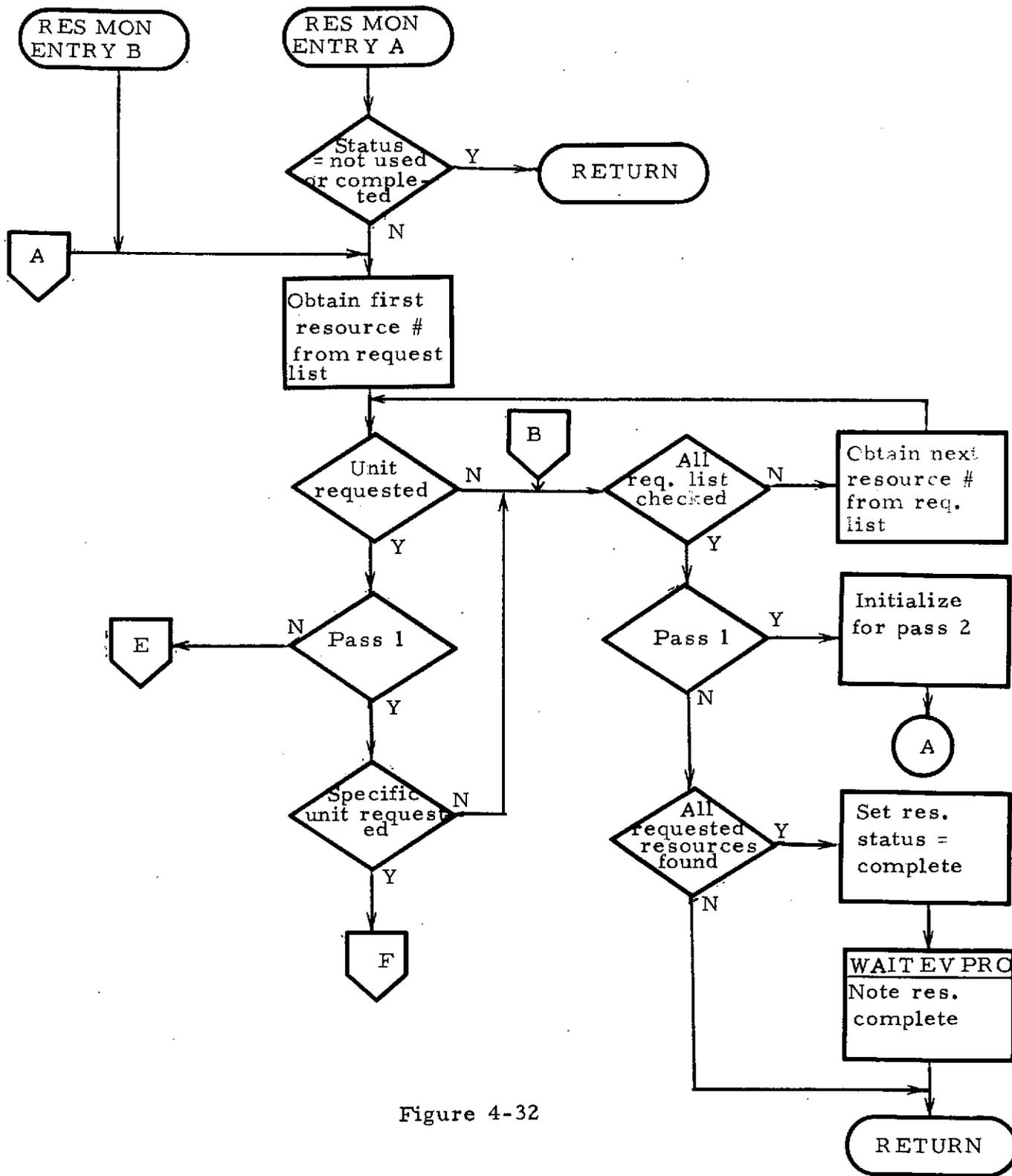


Figure 4-32

RESERVATION MONITOR
(continued)

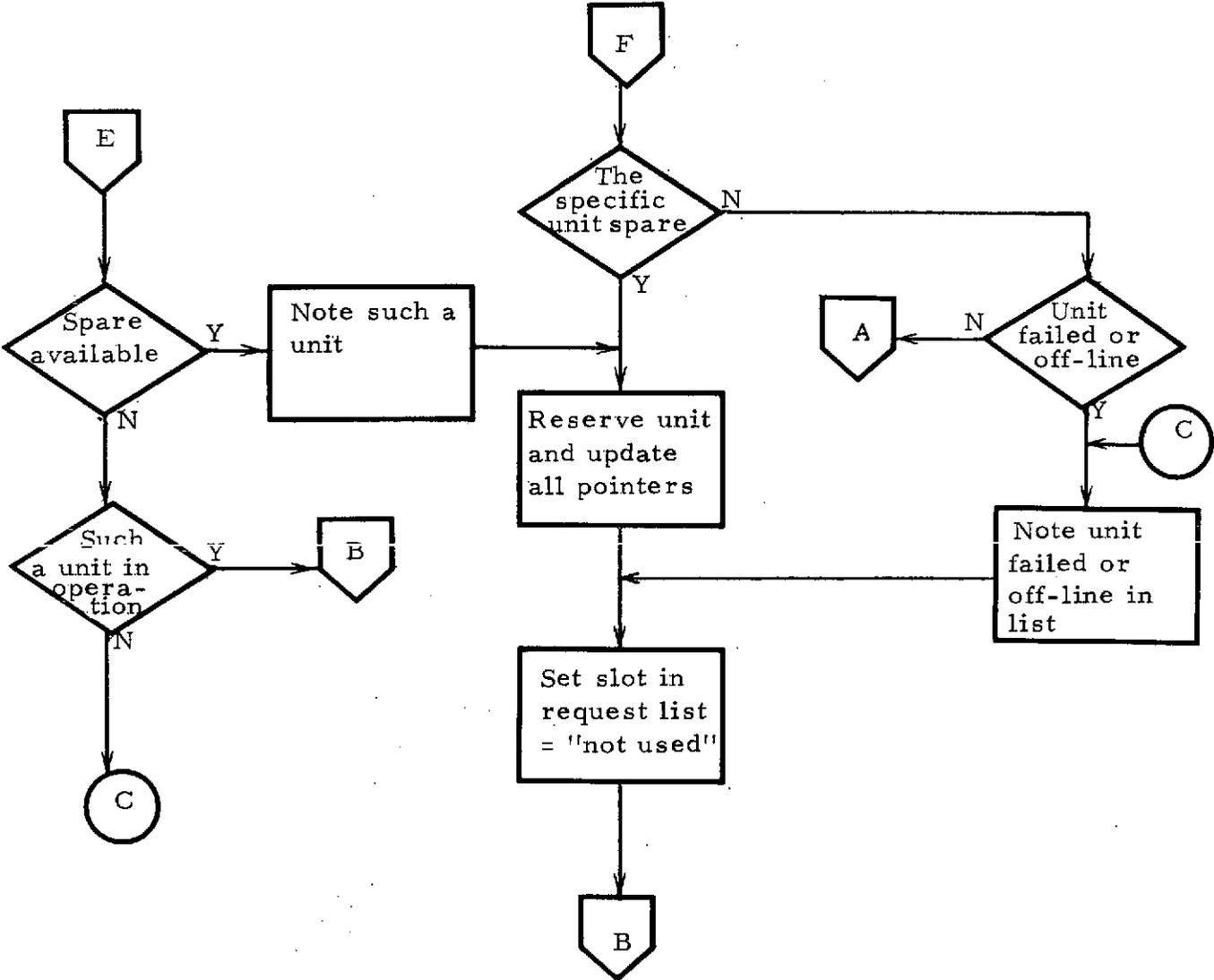


Figure 4-32
(continued)

4.7 Fault Detection and Diagnostic Processing

4.7.1 Diagnostic Overview

Fault detection and processing in BOSS is divided into two independent tasks which execute at different priority levels. These tasks are Fault Detector (FD) and Diagnostic Processor (DP). Both of these involve extensive communication with the ACES Dispatcher.

Fault detection is a high (if not highest) priority ACES task. It is placed into execution via a hardware interrupt to BOSS which signifies one or more module status words that have changed due to a failure. It is assumed that the hardware stream in which the fault occurred is automatically stopped when the MSW is updated, unless the stream is associated with a TMR task and thus is a first failure. The Fault Detector identifies the MSW(s) which have been updated and by interrogating these MSW(s) determines which units have failed. It then updates a Module Failed Word (MFW) accordingly and calls the Dispatcher and Diagnostic Processor.

When control is passed to the Dispatcher, it determines if an updated MFW exists. If so, the Dispatcher determines which units may be at fault by scanning the MFW and marks these units in the necessary tables as being failed. It then scans its streams to determine if any streams are utilizing possibly failed hardware. Any streams utilizing suspected hardware are placed into a configuration wait and are disassembled into individual components. The Dispatcher then continues its normal operation.

The Diagnostic Processor is a low (if not the lowest) priority task within the ACES. The Diagnostic Processor begins execution by scanning the MFWs to determine which components are suspected of failing. The Diagnostic Processor utilizes the MFWs and their corresponding time tags to determine the sequence of failures and attempts to run diagnostics upon the suspected failed units. In order to determine if a failure is continuously occurring, the DP must have the capability to reset the failure indications in the MSW. Communication is established between the FD and the DP so that when diagnostics are being performed upon individual components, the FD does not need to process those errors that are the result of the DP diagnostics. Since all possible failed components are initially marked by the Dispatcher as failed, the DP can perform diagnostics on those components without fear of their use by other executing streams.

As failure indications are diagnosed and pinpointed or resolved, non-failed modules are returned to the resource pool as available resources, and the Dispatcher is called so that they may be utilized in application streams as required.

As failures are found, a brief history of the failure is logged and the DP reschedules the diagnostic to be performed again on the module at some later time to determine if the failure no longer occurs after a period of time. If so, the modules are restored to an available status and returned to the resource pool.

4.7.2 Diagnostic Processing Components

The ACES fault processing is divided into two main programs, the Fault Detector (FD) and the Diagnostic Processor (DP). The following subsections describe the basic function of each program.

4.7.2.1 Fault Detector

The Fault Detector is entered upon notification of a failure condition. Typical notification conditions are an MSW changing states or a processor attempting to reference an illegal address. An addressing exception interrupt is generated whenever a logical page is referenced and that page is not currently assigned to any physical memory module. Figure 4-33 presents a functional flow diagram for the Fault Detector.

The FD first determines if the entry was due to an attempt to address an absent logical page. If so, an attempt is made to "page" the logical page from bulk to an available logical memory module (reference Section 4.7.2.2 for a complete description of the TRYPAGE paging routine). After the logical page is in main memory, the stream is restarted by a "start" command and FD exits.

If the entry was not due to a paging exception, the MSWs are scanned to determine if a failure condition bit has been set. If a failure condition indication is found in an MSW, ACES needs to diagnose the error and a Memory Fail Word (MFW) is built by the FD based upon the information in the MSW. The MFW contains information concerning the type of error which occurred (number of voters which detected an error, etc.) and identifies suspected failed modules.

Once the FD has completed the MFW, the count of the number of MFWs is incremented. If no memory module was suspect, the Dispatcher is called to disassemble the stream(s) utilizing the failed module(s).

If a memory module is suspected, the subroutine MEMFAIL is called to switch a spare physical memory module into the failing logical memory module. After MEMFAIL processes the memory module failure,

FAULT DETECTOR

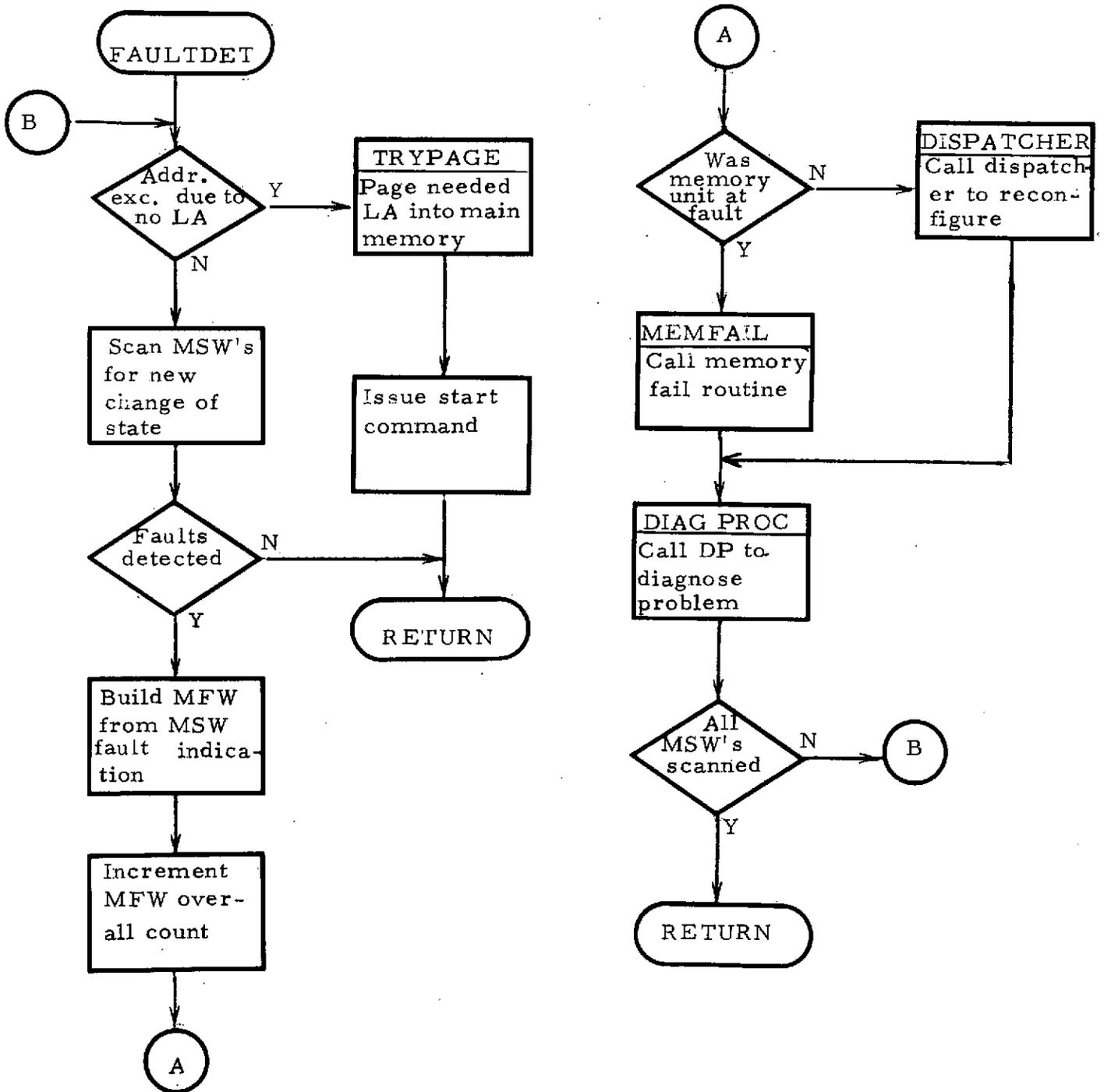


Figure 4-33

the DP is called to perform diagnostics on the modules specified in the MFW(s). If the DP can reproduce the error indication, the unit is marked "failed" in the resource pool control table, making the unit unavailable for normal utilization. Otherwise, the unit(s) are returned to the resource pool as available resources.

TRYPAGE

TRYPAGE is a subroutine used by both the Fault Detector and the Memory Reconfigurator. TRYPAGE determines whether the desired logical page can be brought into main memory. If not, a memory overload condition has occurred which requires that a task dictionary of lower level be initiated. A functional flow diagram of TRYPAGE is presented in Figure 4-34.

Upon entry, TRYPAGE determines if the logical page in question is "pageable". If not, and it does not exist in another logical module, the number of pageable logical modules will be reduced since, once a non-pageable page is resident, it cannot be returned to bulk. Thus the count of the current number of pageable logical modules for the current task dictionary will be decreased and compared to the number required during this task dictionary period. If the number is not sufficient for the current task dictionary, paging is not performed, but rather a new task dictionary of lower level is brought in from bulk store.

PAGER

PAGER is a subroutine called by TRYPAGE when an available logical unit has been found into which a logical page can be paged. PAGER performs the actual paging functions. Figure 4-35 presents a functional flow diagram for the PAGER subroutine logic.

To insure that all pageable logical units are paged through sequentially, PAGER keeps account of which logical module at each criticality was the last module paged. Therefore, by beginning the scan for the next available logical module at this point, the logical modules can be cyclically selected. The current contents of the chosen logical modules are first written to bulk to insure that an up-to-date copy of the page is available for later use. If the logical page required is already resident in main memory, it is copied to the new logical module and the old logical module is made a spare. If a copy is not in main memory, it is read from a bulk storage device.

Once the desired page is in main memory, a call is made to the Wait Event Processor to signal that a memory module paging function has

SEARCH PAGEABLE UNITS

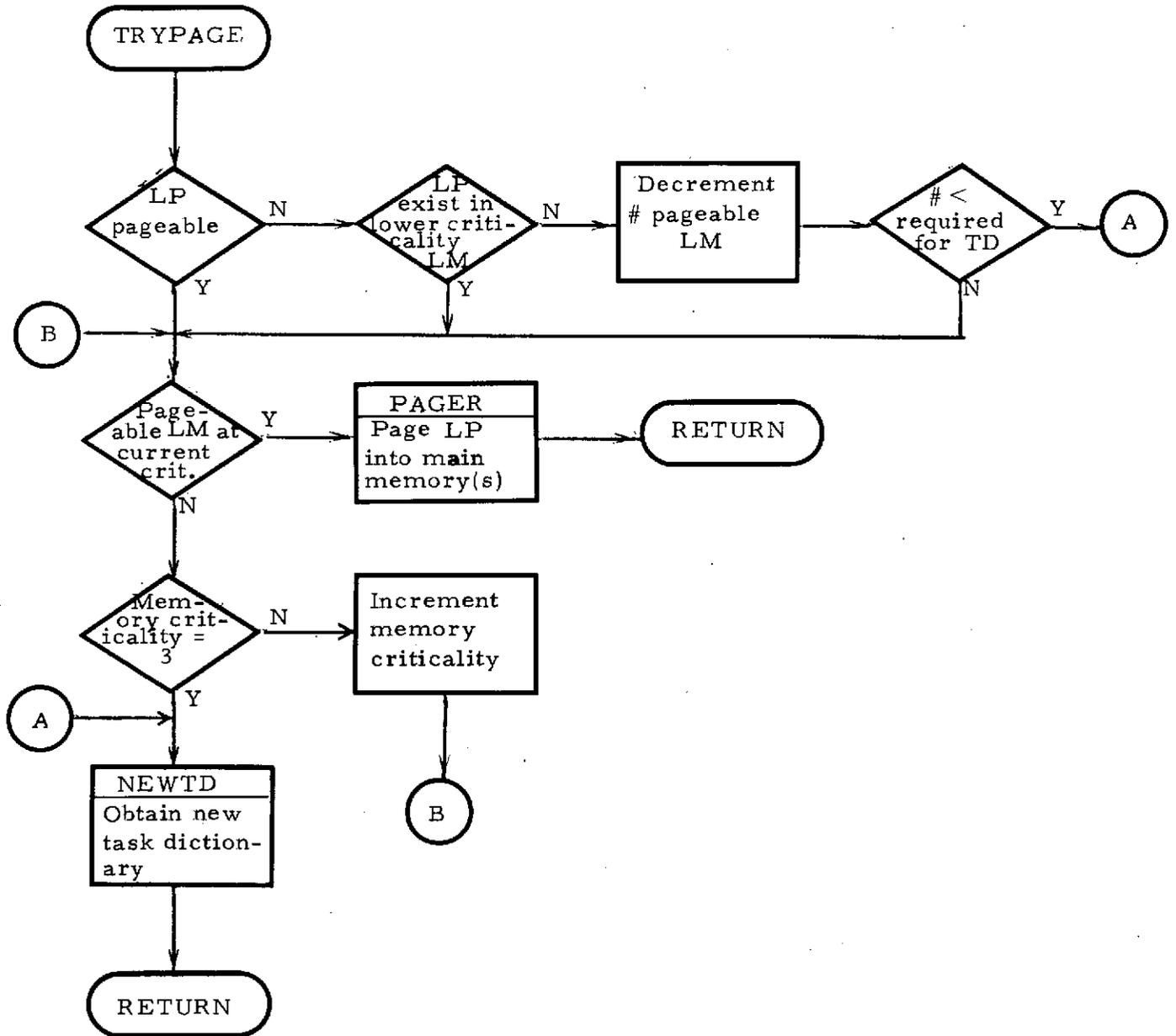


Figure 4-34

PAGE ROUTINE

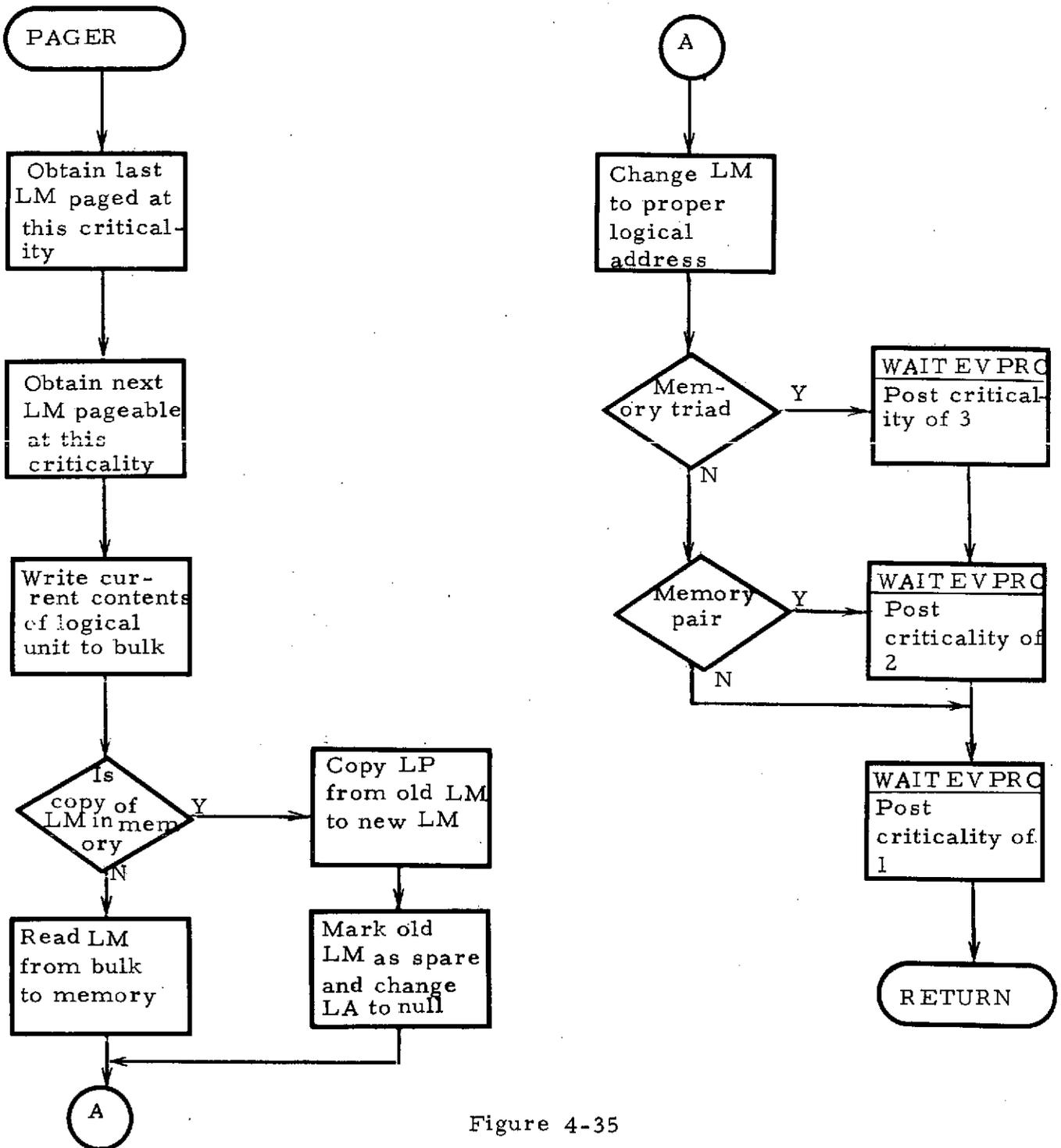


Figure 4-35

been performed, in case a task is waiting upon the newly loaded logical page. Since TMR triad logical memory modules can be accessed by duplex and simplex streams, three separate calls must be made to the Wait Event Processor to identify the availability of the new page to the TMR, duplex, and simplex streams that might wish to utilize it. Similarly, if the logical memory module was a duplex pair, both waiting duplex and simplex streams are posted.

NEW TASK DICTIONARY

The New Task Dictionary (NEWTD) routine is called from TRYPAGE whenever a memory overload condition occurs requiring that a task dictionary of lower level be initiated. The functional flow diagram of the New Task Dictionary program is presented in Figure 4-36.

NEWTD stops all processing streams upon entry. It then determines which task dictionary is to be initiated and loads it into BOSS memory. The routine then updates the pointers associated with task dictionaries and initializes the count of the number of pageable units. The New Task Dictionary is then initialized and initiated.

MEMORY FAIL ROUTINE

The Memory Fail (MEMFAIL) routine attempts to recover from a memory module failure by replacing that module with a spare. If the physical memory module is a member of a TMR triad logical unit, certain conditions must be satisfied before the unit is replaced. Figure 4-37 presents a functional flow diagram for the Memory Fail logic.

Upon entry, MEMFAIL marks the suspected failed memory module failed in the Logical Address Assignment Table (LAAT). If the memory module is not a member of a TMR triad, the Memory Reconfiguration routine is called in an attempt to replace the suspected failed module.

If the module is a member of a TMR triad, a check is made to determine if it was a TMR stream which detected the failure. If so, the stream should be allowed to continue executing the task until it completes or until it enters the wait state since all single point failures from the module will be masked. However, a bit is set to inhibit further TMR dispatches and the task's execution priority is set to the highest possible level to insure the problem is corrected at the earliest time which will not interfere with this task's execution.

NEW TASK DICTIONARY

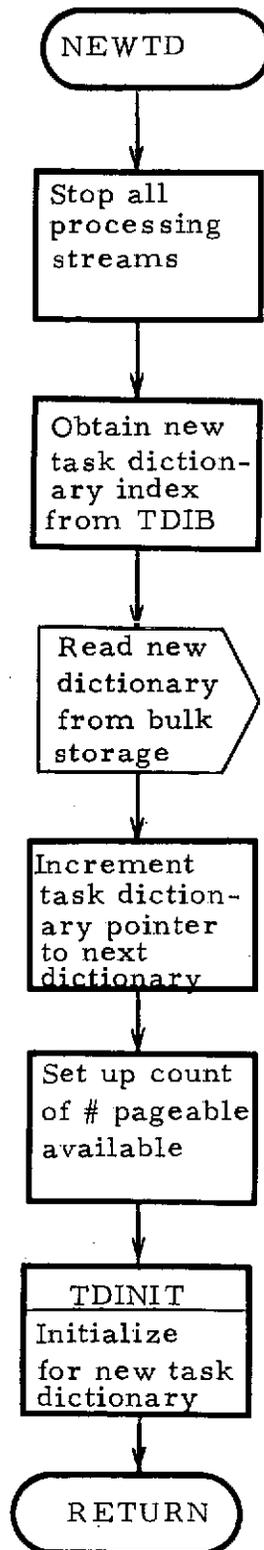


Figure 4-36

MEMORY FAIL

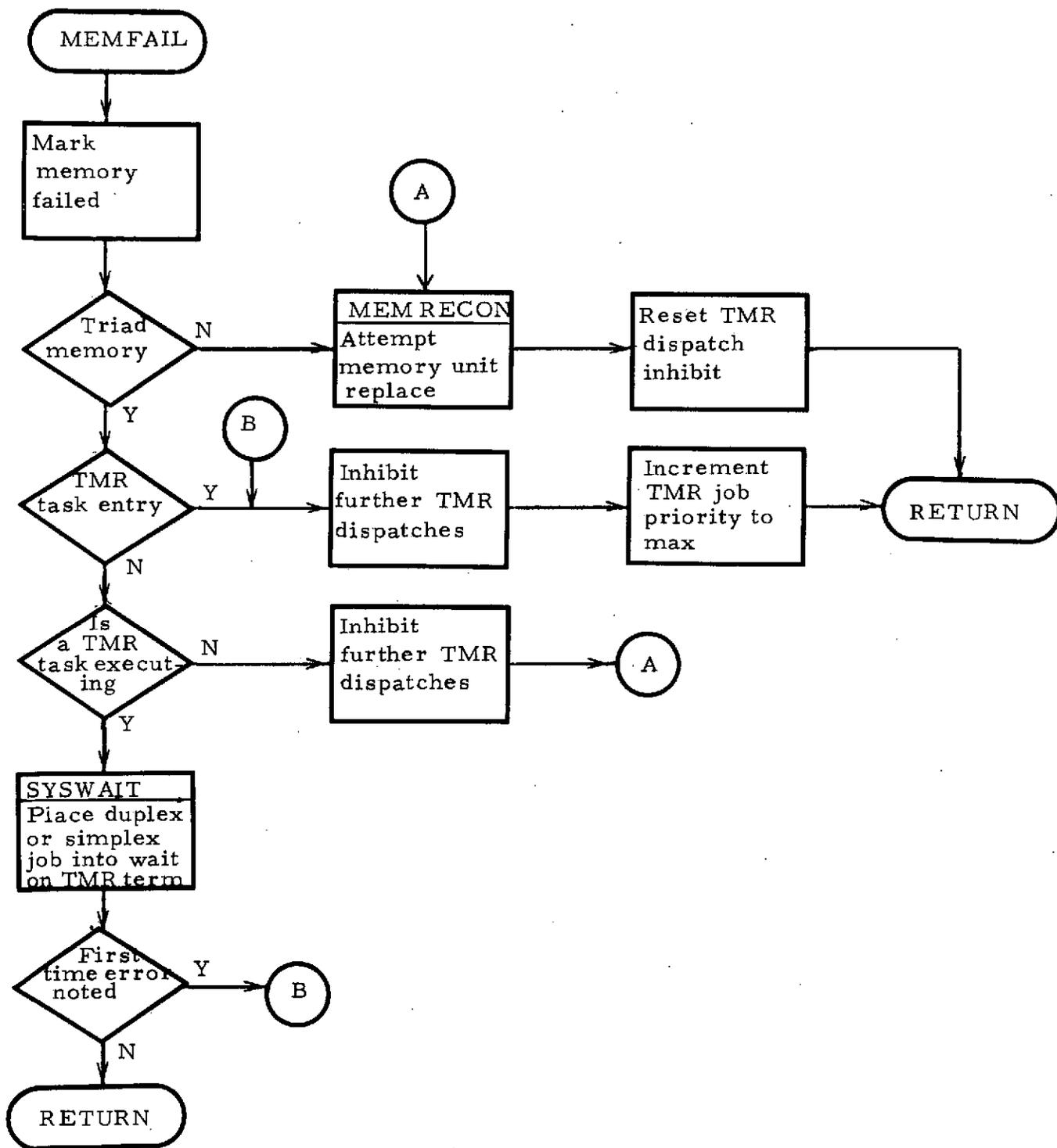


Figure 4-37

If a TMR stream did not note the failure, a check is made to determine if a TMR task is currently executing. Any such TMR stream should not be interfered with when replacing the failing module. However, the simplex or duplex stream which noted the failure must be placed into the wait state until the TMR task completes or enters the wait state and the failing logical memory module can be repaired.

If the failure had been noted previously, no further processing is needed and MEMFAIL exits. Otherwise, further TMR dispatches are inhibited and the current TMR task's execution priority is incremented so that it will come to a timely completion.

If a TMR task is not currently executing, further TMR dispatches are inhibited while the failing memory unit is being repaired, to insure that no TMR tasks began execution during the replacement operation.

4.7.2.2 Diagnostic Processor

The Diagnostic Processor is called by the Fault Detector to perform diagnostics on suspected failed modules. The Diagnostic Processor performs diagnostics and communicates to the Dispatcher the status of the module. The Diagnostic Processor is also executed periodically to retest all previously failed modules so that any module which at some future time resumes proper operation can be returned to an operational status.

To confirm the overall diagnostic philosophy developed, a detailed design has been completed for one segment of the diagnostic package. The programs FAIL1, FAIL2, and FAIL3, present the concept as it applies to the faults which are discovered by voter detection logic within ARMMS. These programs diagnose failures which occur during a TMR task execution.

Figure 4-38 presents a functional flow diagram for the Diagnostic Processor logic.

When entered, the DP tests to determine if a voter has detected a failure since the DP's last execution. This is accomplished by testing the MFW(s) as they are built by the FD whenever an error occurs. If an error has occurred, the DP calls the appropriate routine to diagnose the error. A unique routine is available for each type of error that may occur. After the appropriate diagnostic routine has completed, the overall MFW count is decremented to indicate that the error has been processed.

DIAGNOSTIC PROCESSOR

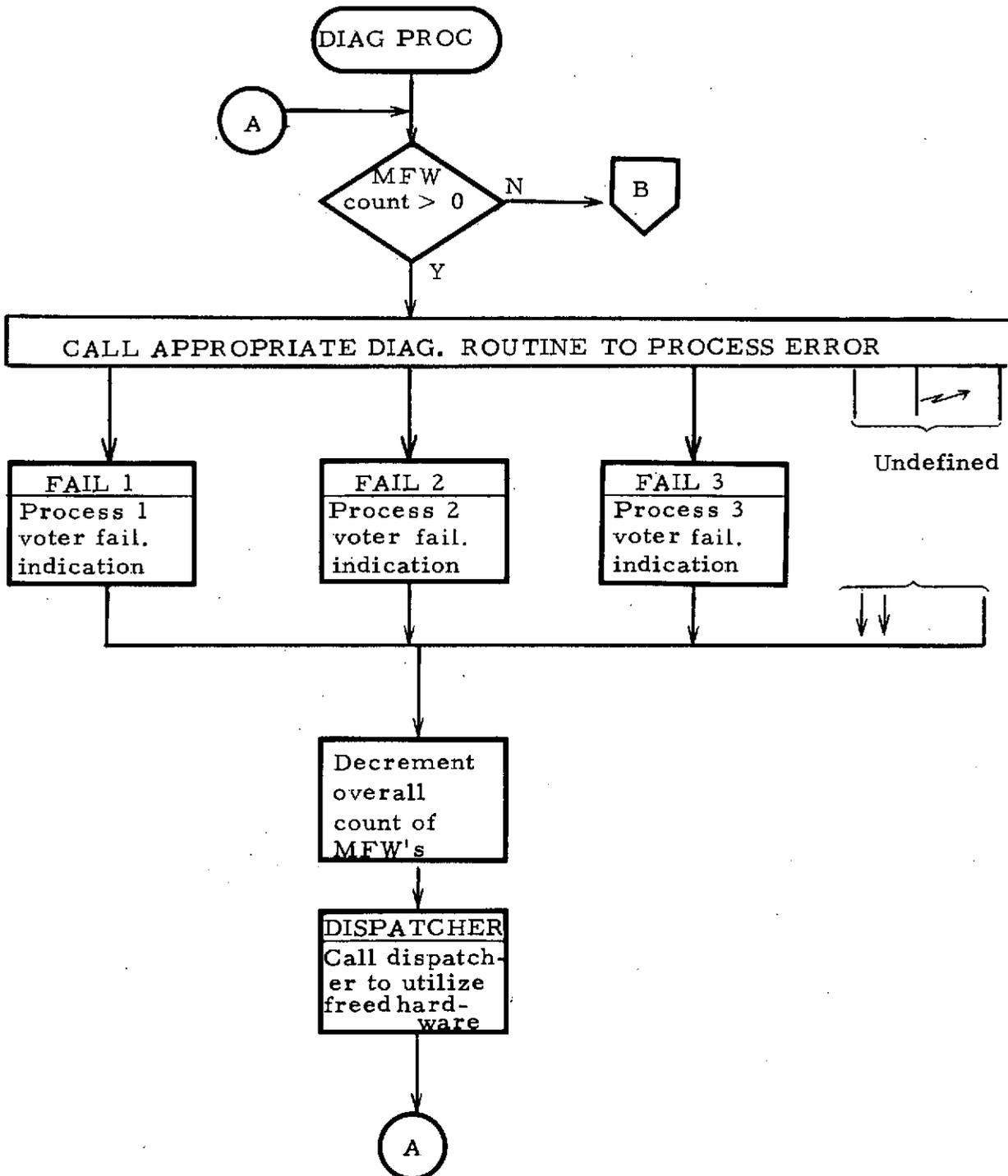


Figure 4-38

DIAGNOSTIC PROCESSOR
(continued)

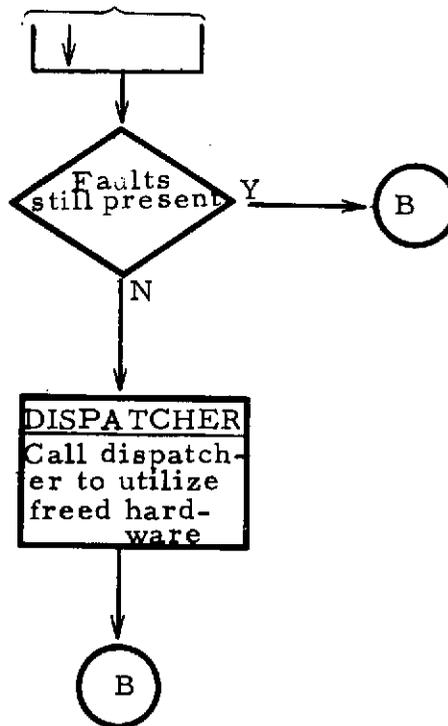
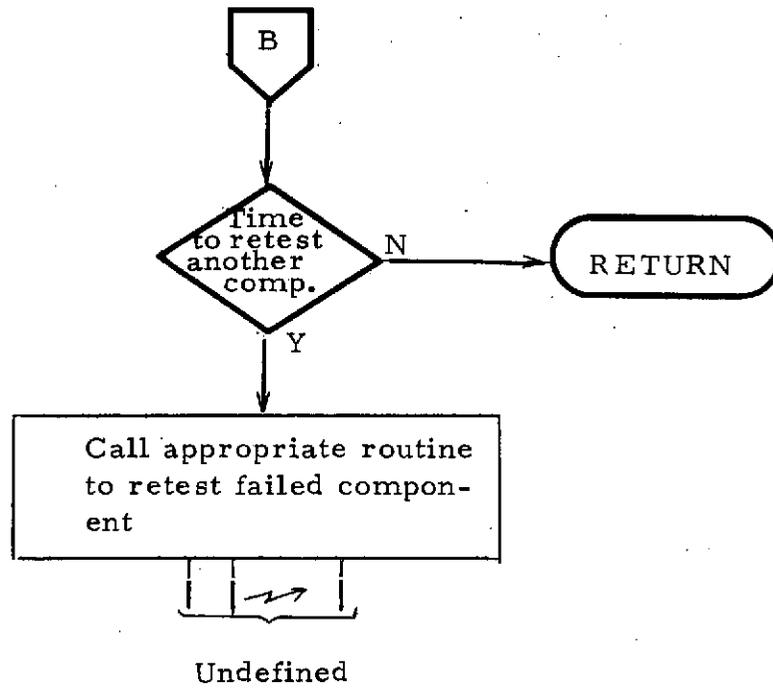


Figure 4-38
(continued)

Since units marked failed previously may have been returned to an available status, the Dispatcher is called to determine if any of the resources freed may be used to form a required stream.

The DP then loops to determine if more MFWs are awaiting processing.

FAIL1

FAIL1 is a subroutine of the Diagnostic Processor that diagnoses failures due to a single voter of a TMR set (i. e., memory triad, CPE set, or IOP set) detecting a failure condition. When one voter in a TMR set detects a failure, one of two possible single point failures can exist:

- 1) Localized Bus Port Failure - A localized bus port failure has occurred where only one bus input was received in error. If the complete bus had been in error, the other two voters would have detected the same failure condition.
- 2) Voter Failure - A voter failure within one component has occurred such that a non-compare is indicated.

Since these are the only two single point failures that can occur with FAIL1, the MFW associated with the diagnostic routine FAIL1 contains the bus port address marked as failed by the module and the module which noted the failure. FAIL1 determines which failure actually occurred. Figure 4-39 presents a functional flow diagram for the FAIL1 logic.

Upon entry, FAIL1, via a Reservation Request call, requests the other necessary modules (buses, memories, etc.) necessary to form a usable stream to diagnose this particular problem. A Wait call is then issued to wait for the reservation to be completed, since the requested units must be available before meaningful diagnostics can be performed. A module test is then performed for the module which is believed to have failed. If no failures are detected by the diagnostic software, an intermittent failure indication is then logged for both the bus port and the module since both were suspected.

A check is then made to determine if either the bus port or module has had an excessive number of failures. This check is performed to identify modules which have had a high number of apparent intermittent failures which the diagnostic routines have not been able to recreate. If it is decided that either the bus port or the module has had an excessive number of failures, they are marked "failed" and retesting is scheduled for later in the mission.

FAIL 1

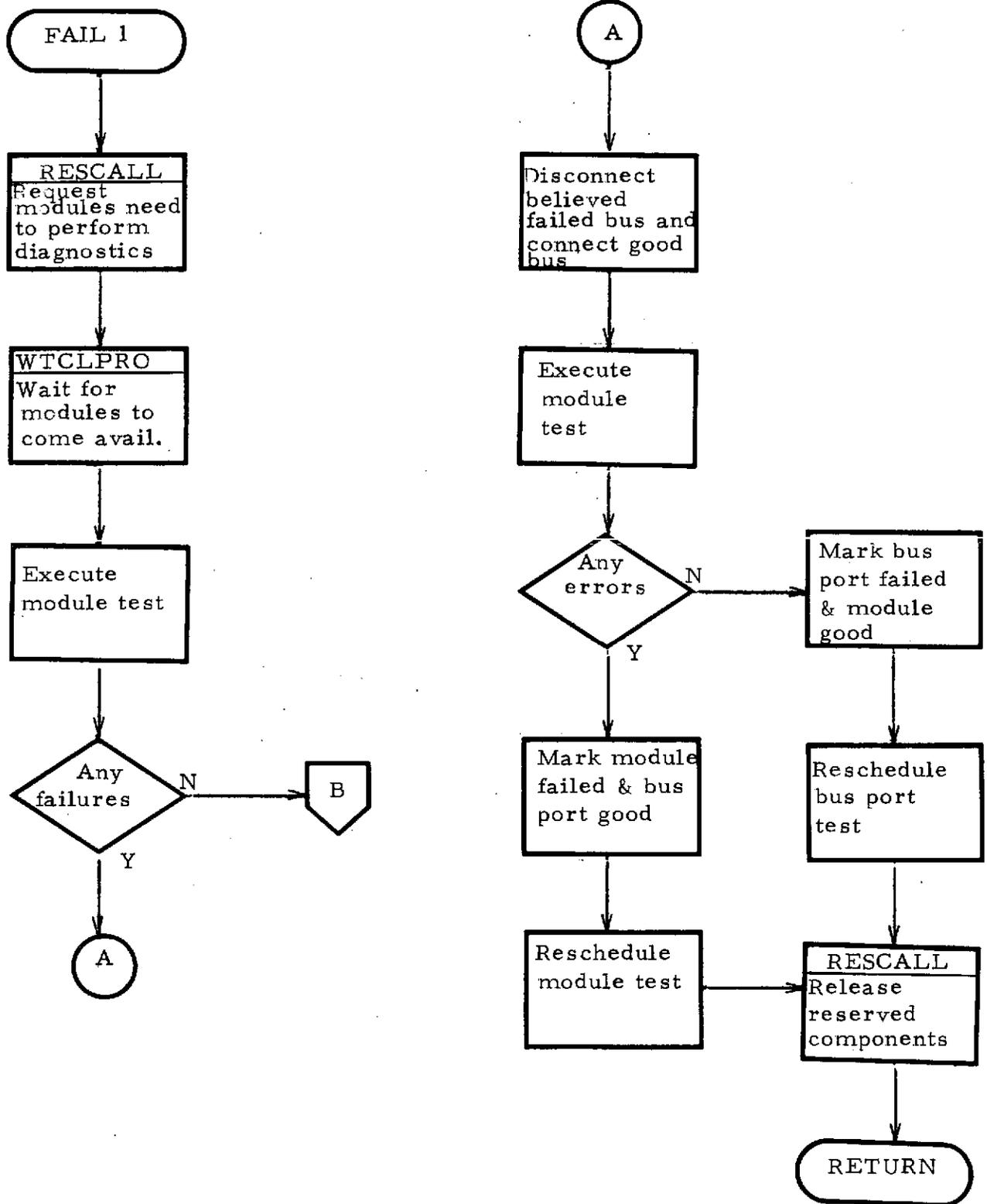


Figure 4-39

FAIL 1
(continued)

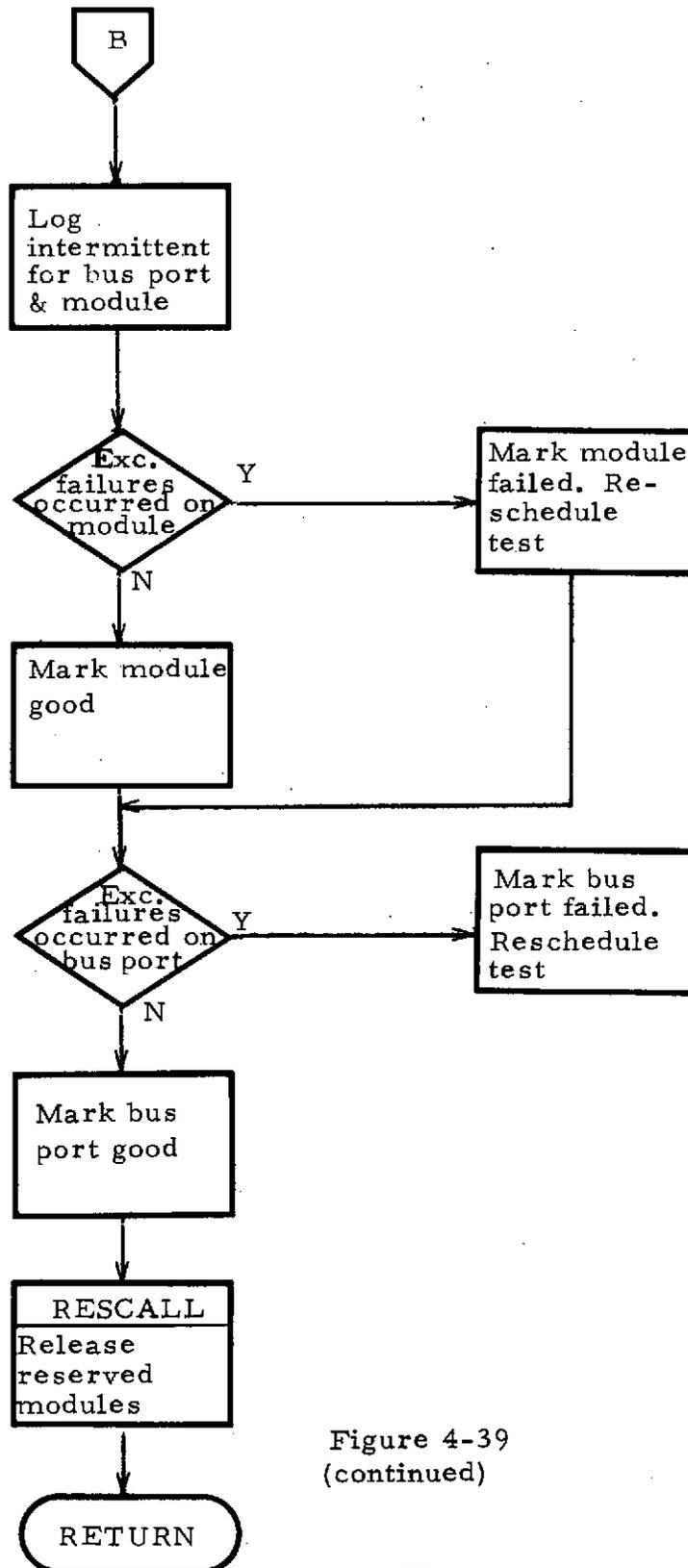


Figure 4-39
(continued)

If the intermittent counts for the modules are not excessive, each is marked good and returned to an available status. The reserved modules are then released for general use by the Dispatcher.

If a failure is detected by the module test, the failure has been re-created and the routine begins to determine which module really failed. The bus associated with the failed bus port is disconnected and a new bus is connected to the suspected module. The unit test is re-executed. If failures are still present, the unit is declared failed. If failures are not present, it is assumed the bus port marked in the MFW is at fault and it is marked accordingly. The module which was marked failed is rescheduled for later test and the operational reserved modules are released.

FAIL2

FAIL2 is a subroutine of the Diagnostic Processor which diagnoses failures due to two (2) voters of a TMR set (i. e., memory triad, CPE set, or IOP set) detecting a failure condition. When two voters in a TMR set detect a failure, only one single point failure can occur. This failure must be a result of partial bus failure which causes some (this case at least two) input ports to receive incorrect data and other input ports to receive valid data. Since, functionally, there is no difference between a port failure and a partial bus failure from a receiving device standpoint, the ACES philosophy will identify partial bus failures as input port failures. Therefore, the two input bus ports suspected of failure are identified in the associated MFW. Figure 4-40 presents a functional flow diagram for the FAIL2 logic.

Upon receiving control, FAIL2 requests, via a Reservation Request call, the necessary modules needed to form a usable stream to diagnose the suspected problem. A wait is then issued for the desired units to be reserved since they must be available before meaningful diagnostics can be performed. A diagnostic test is then performed in an attempt to recreate the failure. If failures are found, both the input ports are marked failed since the failure could be recreated.

If no failures were detected, an intermittent failure indication is logged for both input ports. A check is then made to determine if either or both input ports have had an excessive number of apparent intermittent failures. This check is performed to fail components which have a large number of apparent failures which the diagnostic routines cannot recreate.

FAIL 2

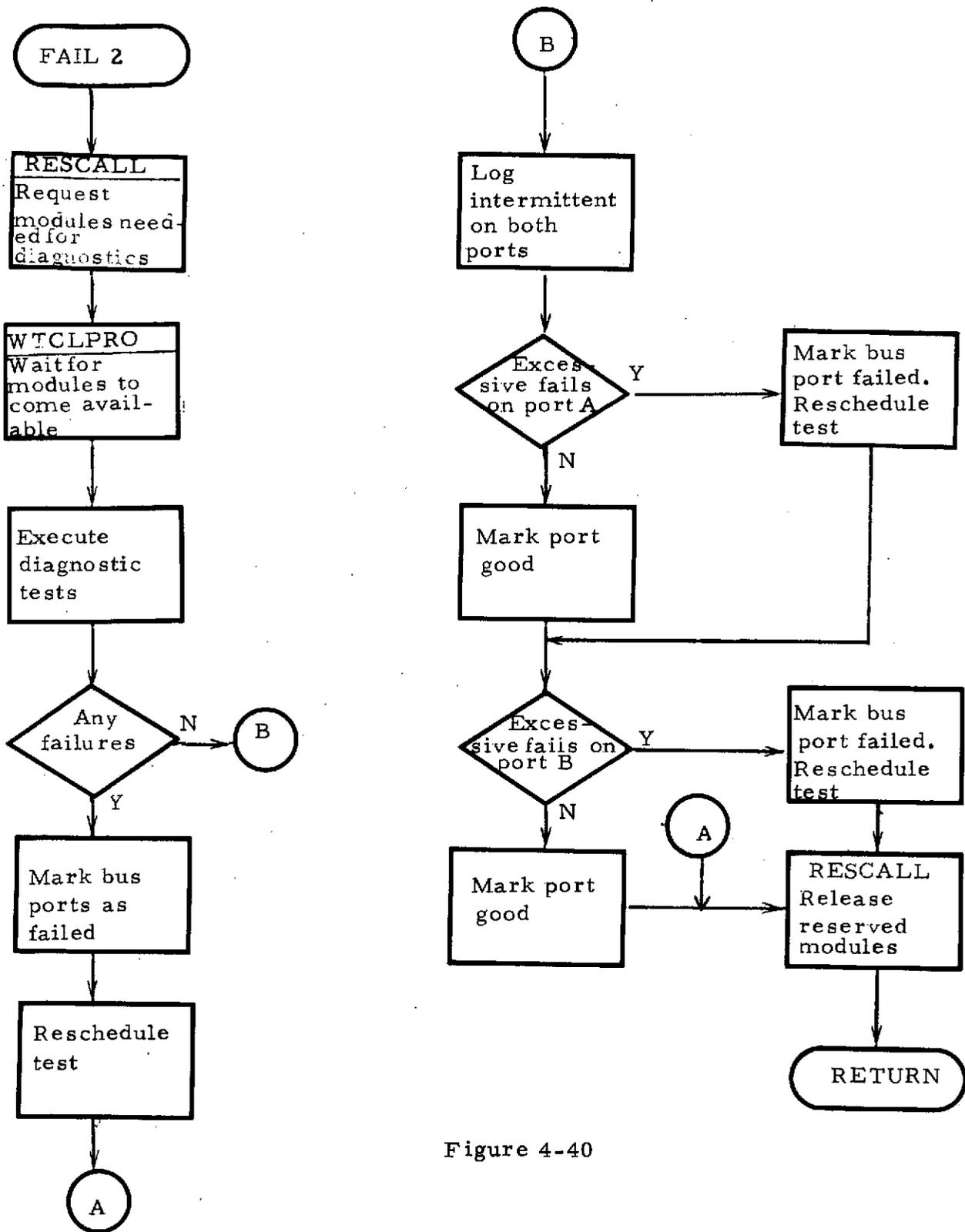


Figure 4-40

If either input port has had an excessive number of failures, it is marked "failed" and retesting is scheduled for a later time. If the failures are not excessive, each is cleared of the suspected failure and returned to an available status.

FAIL3

FAIL3 is a subroutine of the Diagnostic Processor which is called to diagnose failures due to all three (3) voters of a TMR set of voters detecting a single failure condition. Whenever three voters detect the same failure, one of three possible single point failures can exist:

- 1) Unit Failure - One transmitting unit could have failed and sent invalid data to all three receiving units.
- 2) Output Bus Port Failure - An output bus port from a transmitting module could have failed, thereby sending invalid data to three modules.
- 3) Total Bus Failure - One bus could have failed which would cause all modules to receive invalid data.

The MFW associated with this failure denotes the suspected failed module, bus port, and bus. A functional flow diagram of the FAIL3 logic is presented in Figure 4-41.

FAIL3, upon entry, requests the modules necessary to form a usable diagnostic stream to diagnose the problem. A wait is issued until the desired modules are made available by the reservation system. Diagnostic tests are then performed in an attempt to recreate the problem. If the failure condition cannot be recreated, it is assumed to have been intermittent. LOGINT is called to log the intermittents and the reserved modules are released.

If the failure is recreated, the diagnostics begin to determine which module caused the failure. The output bus, which is suspected, is disconnected and another bus is connected to the diagnostic stream. Diagnostic tests are again executed. If failures still occur, the suspected module is marked failed since the failure is still present and neither the bus nor output port is functional within the diagnostic stream. The module test is rescheduled for later, in the event the unit becomes functional at a later time, and the reserved modules are released for use by the Dispatcher.

FAIL 3

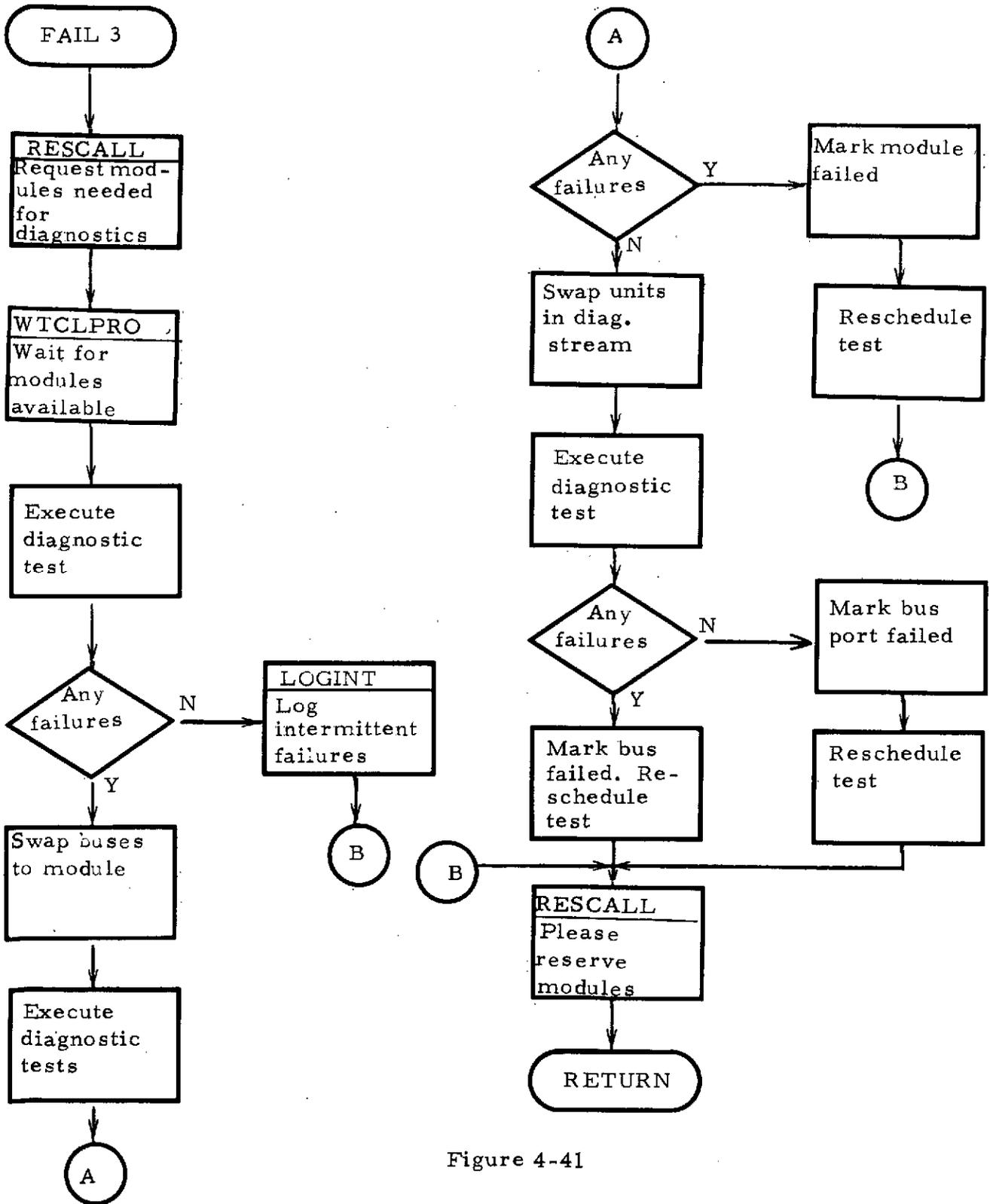


Figure 4-41

If failures were not detected, the module is assumed to be good and the diagnostics must determine if the bus port or bus caused the failure. Another unit is connected to the stream in the place of the module just discovered to be good. The original suspected bus is connected to the unit and the diagnostic test is again executed.

If no failures are found, the bus is determined to be good since valid data flows through it when connected to another module. Therefore, by default the original suspected bus port is marked failed.

If errors are present with the new module and the original bus, the bus is marked failed, and the reserved modules are released.

LOGINT

Log Intermittent (LOGINT) is a subroutine of FAIL3 which is called to log failures that cannot be reproduced. Figure 4-42 depicts the Log Intermittent logic in a functional flow diagram form.

An intermittent failure is logged for the suspected bus, bus port, and module. A check is then made to determine if any module has associated with it an unusually high number of recorded intermittent failures. If so, the module is marked failed. If excessive failures are not found, the module is marked good and returned to spare status.

LOG INTERMITTENT

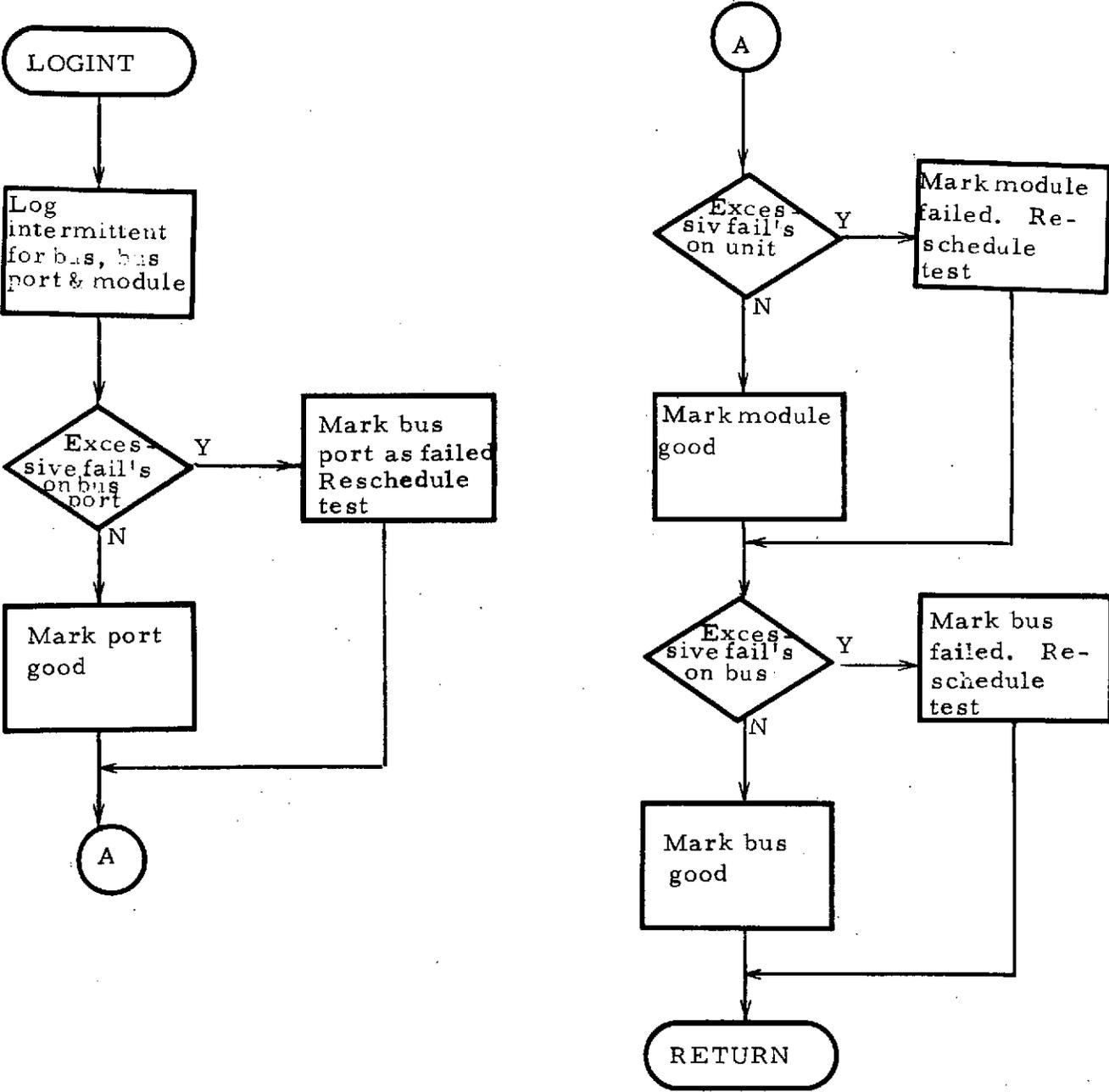


Figure 4-42

4.8 Information Protection

4.8.1 General Description

Execution time protection of main memory contents in this preliminary version of the ACES is concerned with three areas:

- o Protection against unanticipated accesses (hardware/software errors).
- o Coordinated use of data shared by multiple, concurrently executing, tasks.
- o Coordinated use of subtasks (subroutines common to tasks) by multiple, concurrently executing, tasks.

The method proposed uses base/bound registers, rather than protect keys. That is, access to an area in memory is authorized by the ACES by providing the task with the lower and upper limits of an area in memory. Any access to memory is checked against these limits prior to the execution of a memory access.

The main advantages of this particular approach are the following:

- o Accessible areas in memory may start and end anywhere in memory. That is, the size and location of a protected memory area are completely flexible.
- o Once accesses have been authorized, further participation by the ARMMS Control Executive is not necessary, thereby preventing excessive overhead.
- o A minimum of special hardware is required.

The majority of the logic required to accomplish this function can be resident within the individual processors, rather than within BOSS. The exception to this is the logic that should be added to ACES to interface with the function discussed here.

4.8.2 Base/Bound Registers

Access to a memory area is enabled for a task by providing it with the access limits in a set of registers, called the base/bound registers. Accesses within this memory area are performed relative to the contents of the base register.

An individual task may require access to four separate (non-contiguous) memory areas and therefore requires, at least, four sets of base/bound registers. These registers and their associated memory areas are discussed below:

1) Program Base/Bound

These registers delineate the area used for program storage. It is, therefore, only used during an instruction fetch to insure that the program did not go outside its anticipated bounds.

These registers are set when a task is assigned to a processor for execution by the Task Initiator.

2) Local-Data Base/Bound

These registers delineate the memory area containing data local; (i. e., non-shared) to a task. Whenever the base register is used to form an address, the effective address is compared against the base/bound limits prior to access.

These registers are set when a task is assigned to a processor for execution by the Task Initiator.

3) Temporary-Storage Base/Bound

Temporary storage (i. e., work storage) may be dynamically allocated when a task is scheduled for execution, and automatically deallocated when a task terminates execution. Temporary storage may also be requested and released during task execution.

The main purpose for dynamic allocation/deallocation is to minimize required memory by sharing work storage between multiple tasks.

These registers, therefore, delineate the memory area assigned to a task for temporary storage. Whenever the base register is used to form an address, the effective address is compared against the base/bound limits prior to access.

These registers are set if and when ACES assigns temporary storage to a task.

4) Shared-Data Base/Bound

These registers are used to delineate a memory area containing data accessed by more than one task. Such common data requires certain types of synchronization locks prior to access. Locking logic is described in detail in Section 4.8.4. Whenever this base register is used in an address, the effective address formed is compared against the base/bound limits prior to access.

These registers are set whenever a lock request (reference Section 6.4) has successfully been executed.

4.8.3 Processor Architecture Implications

The implementation of this method has several implications which impact the basic processor architecture. The major effects are described below:

1) Base Register Addressing

Although other methods are possible, a base register instruction addressing format is the most logical format.

The main requirement is that the instruction address indicates which base/bound registers are to be used for access verification. Note that the program base/bound registers are never explicitly indicated in any instruction, but are completely internal to the instruction fetch logic.

2) Separation of Base Registers from Other Registers

To insure full protection, the base registers should not be directly accessible by an application task. Therefore, any registers used for indexing or as accumulators should not be used as base registers.

3) Base Register Set/Reset Flag

To insure that a base register is not used by a task, unless it has been set by the ACES for that task, a set/reset flag has to be associated with each base/bound register pair.

The flags have to be reset whenever a processor terminates or suspends execution of a task for any reason.

4.8.4 Shared Data Locks

Any contiguous set of shared data locations may be "Read-Locked" or "Write-Locked".

A read-lock, applied to a set of data, prevents any other task from modifying that data set until the read-lock has been removed. A write-lock, applied to a set of data, prevents any other task from reading that data set until the write-lock has been removed.

To accomplish the locking, ACES uses "Lock-Variables". A lock-variable is a memory location that contains lock information pertaining to a contiguous set of shared data locations. To facilitate their use, a hierarchy of lock-variables may be defined as depicted in Figure 4-43.

The lock-variable contains user provided information and various system flags to indicate lock activity. The contents are described in Section 4.8.5.

Two Control Executive commands are provided to provide the interface between a task and the locking logic.

1) LOCK $\left\{ \begin{array}{c} \text{READ} \\ \text{WRITE} \end{array} \right\}$ XXXX, $\left\{ \begin{array}{c} \text{READ} \\ \text{WRITE} \end{array} \right\}$ YYYY

This command requests that lock-variables XXXX and YYYY are to be read or write-locked. If the lock cannot be applied, the task will be suspended until the lock can be accomplished.

A task may only have one active (maximum of two lock-variables) lock request. A new lock request may not be issued until all previous locks are removed, to prevent deadlocks.

The functional logic of the ARMMS Control Executive to handle the lock request is depicted in Figure 4-44.

Since lock-variables may be within a hierarchy structure, the Lock Request routine insures that all lower level variables can be locked before any lock is actually applied. If no lower level locks are found or, if found, are of the same type, the lock request is fulfilled by setting the proper indications and incrementing a lock count in each lock level below the requesting variable involved in the lock hierarchy.

LOCK VARIABLE USAGE

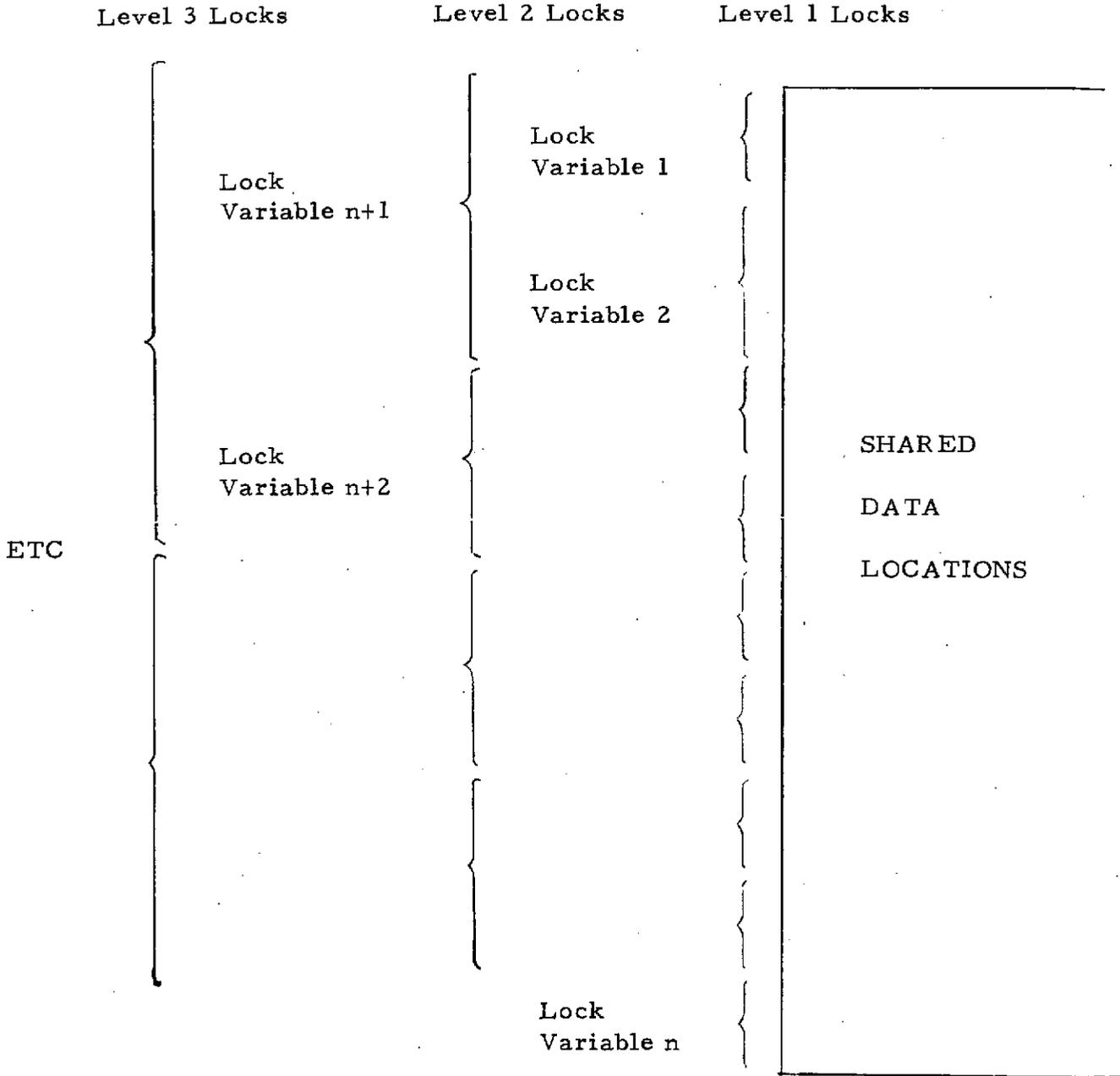


Figure 4-43

LOCK REQUEST LOGIC

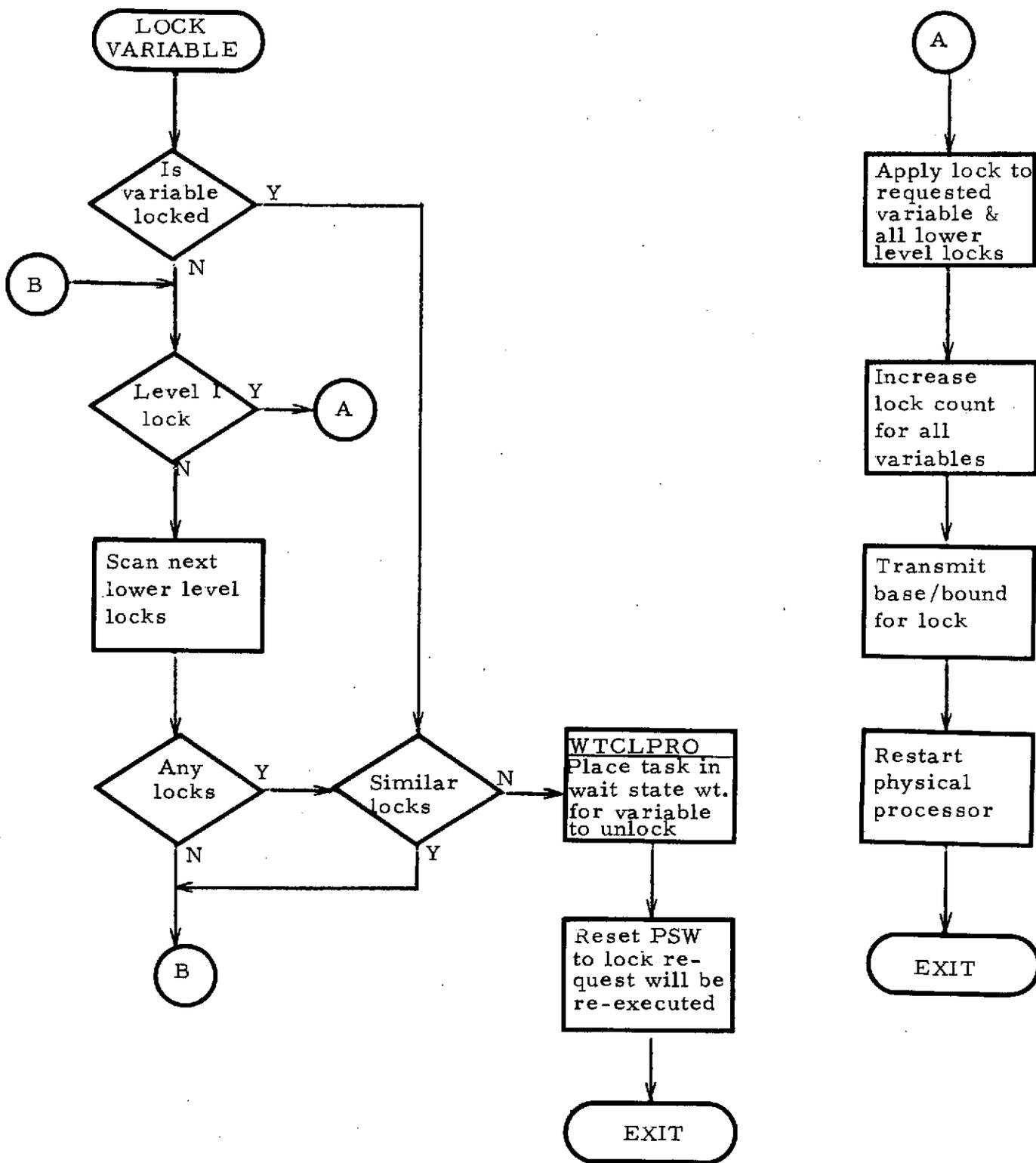


Figure 4-44

If a dissimilar lock is found in the lock search, the lock request cannot be fulfilled at this time and the requesting task is placed into the wait state, waiting for the dissimilar lock-variable to be unlocked. The requesting task Program Status Word (PSW) is reset so that when the task is reactivated, the lock request will be re-executed.

2) UNLOCK XXXX, YYYY

This command removes the lock from lock-variables XXXX and YYYY.

Note that it is assumed that two sets of base/bound registers can be used to access shared data. However, both have to be set within a single command to prevent deadlocks.

Figure 4-45 depicts the functional logic flow of the ARMMS Control Executive to handle the unlock request for a locked variable.

Upon entry, the routine insures that the variable which is to be unlocked is locked. If not, an error return code is sent to the requesting program.

The count of the number of similar locks is decremented by the routine. If the count reaches zero, indicating the variable has no further lock requests, it is unlocked. Since a task may be waiting for a variable to become unlocked, the Wait Event Processor is called to indicate the event of the variable becoming unlocked. This procedure is followed until all lower locks in the hierarchy have been processed. The Base/Bound registers are reset by the Unlock Request routine before exiting.

4.8.5 Lock-Variable Contents

The lock-variables are generated by the user and are placed into main memory. At phase initialization time, the location of the lock-variables is made known to ACES. Figure 4-46 defines the functional components of a lock-variable.

The contents of the lock-variable are functionally described below:

UNLOCK REQUEST LOGIC

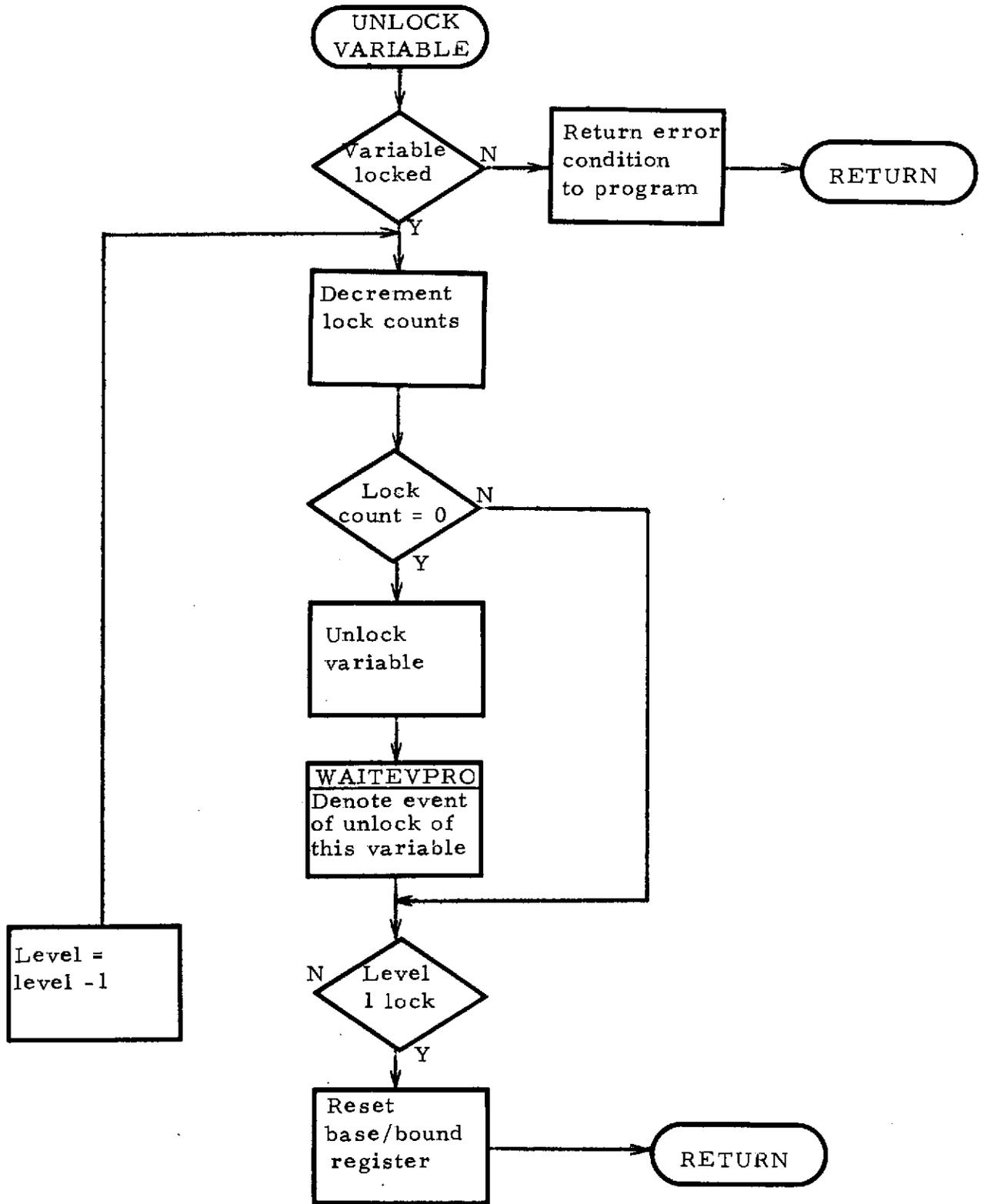
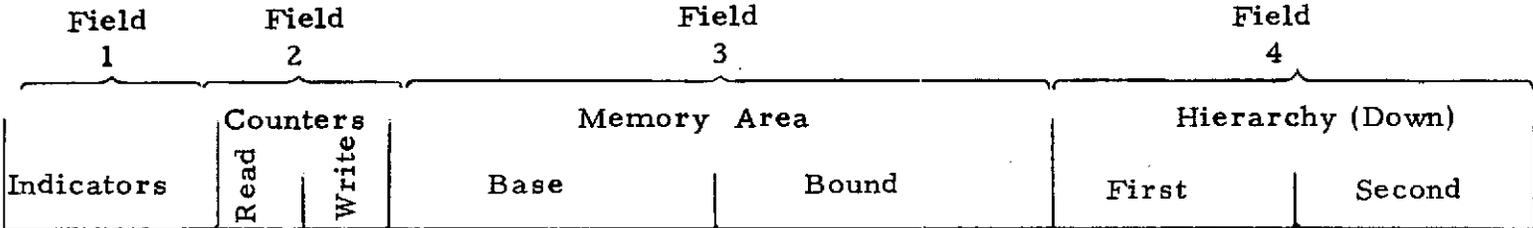


Figure 4-45

LOCK VARIABLE



981-8

Figure 4-46

Field 1 -

Lock applied.

This field contains four indicators:

- Indicator 1 - Signals whether access lock is necessary. If not on, a lock request is immediately executed without any actual locks being set.
- Indicator 2 - If on, indicates that at least one read-lock has been applied.
- Indicator 3 - If on, indicates that at least one write-lock has been applied.
- Indicator 4 - If on, indicates that multiple write-locks may be applied.

Field 2 -

Lock counters.

This field contains two counters:

- Read Counter - Multiple read-locks may always be applied. To insure that the read-lock is not removed until all UNLOCK requests have been completed, a count of active read-lock requests is kept.
- Write Counter - This serves a similar purpose as the read counter for those variables to which multiple write-locks may be applied (see Field 1).

Field 3 -

Memory area.

This field contains the base/bound of the memory area that the lock is applied to.

Field 4 -

Hierarchy pointers.

This field contains the pointers establishing the position of this variable in the lock-variable hierarchy.

4.8.6 Subtask Accessing

It is desirable that tasks may utilize common subroutines. To distinguish these from local (to a task) subroutines, they are called subtasks.

To insure that no usage conflicts occur, calls of subtasks have to go through the ACES. Subtasks could be made into tasks and be called through the normal scheduling mechanism. However, to minimize overhead, a separate mechanism must be used. The sole purpose of this mechanism is to insure that no re-entrancy problems will occur.

An ACES command will be provided: CALL SUBTASK ZZZZ

If the subtask can be used, the ACES will store all task registers and load the registers with subtask information. A subtask may, therefore, have completely different base/bound information.

4.9 Input/Output Control

The design of the Input/Output (I/O) Control is highly dependent on the type of input/output devices to be handled, the residency of the input/output functions of the Control Executive and the characteristics of the Input/Output Processors. Specifically, the latter is key to the overall design. It is, therefore, obvious that a design of the Input/Output Control is not practical at this point. Nevertheless, Input/Output Control should be considered within the current design effort to minimize potential incompatibilities and establish a basic Input/Output Control philosophy to guide future design efforts.

This section, therefore, discusses mainly input/output transmission characteristics, the input/output configurations anticipated in ARMMS, and the potential residency of the input/output functions.

4.9.1 Types of I/O Transmissions

Potentially, two types of I/O transmissions may be required in ARMMS missions.

The first type involves the transmission of relatively large volumes of data such as disk/drum input/output and telemetry. Such a bulk transmission typically requires from several milliseconds to several seconds to complete.

Conversely, the second type is characterized by short messages (say four words or less), and consequently, may require only microseconds to complete. Communication with many of the space vehicle devices is likely to fall in this category. Significant is that these may have to be performed at short, and relatively precise, time intervals (e.g., sampling and control rates).

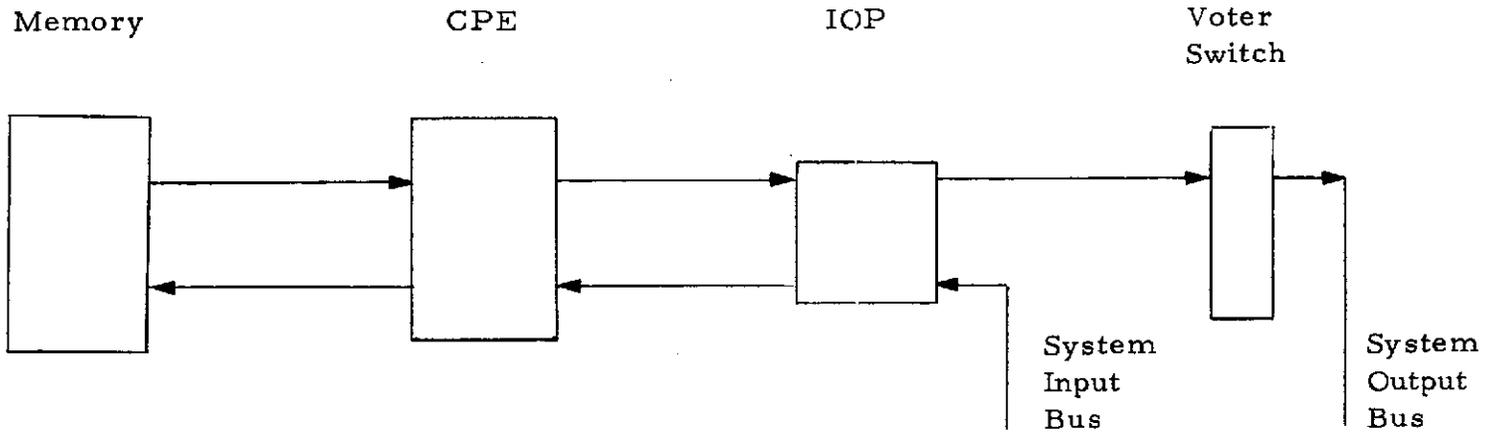
4.9.2 Types of Input/Output Streams

To support the above mentioned transmissions, two types of Input/Output Streams (i.e., input/output processing configurations) have been established: an Input/Output Processing Stream (IOPS) and a Full Processing Stream (FPS). Both of these are depicted in Figure 4-47.

The Input/Output Processing Stream (IOPS) is designed to process the long, bulk I/O transmissions. This transmission is normally independent of the CPE and, therefore, the CPE is not needed as a module

TYPES OF I/O STREAMS

Full Processing Stream



I/O Stream

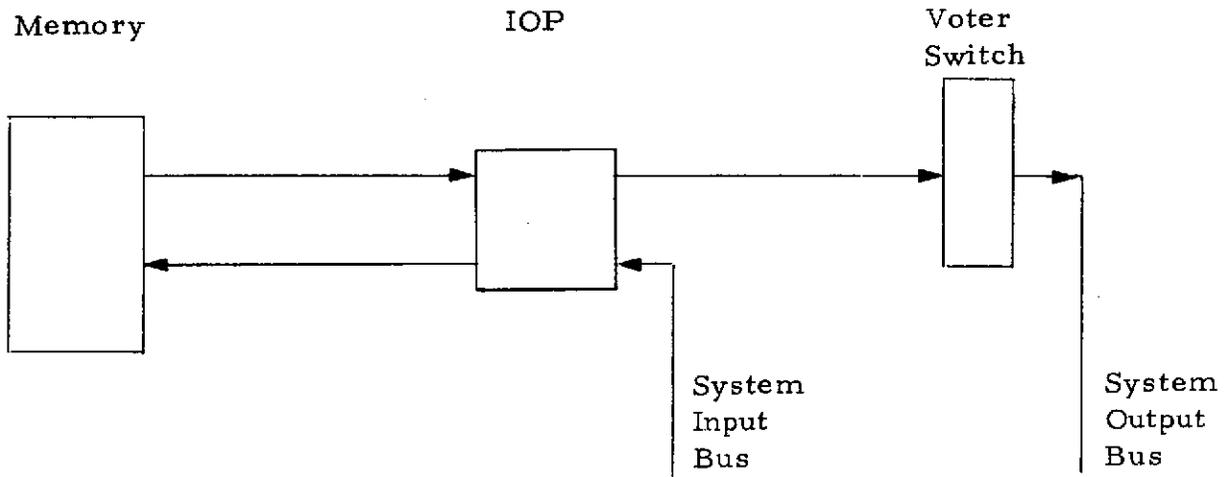


Figure 4-47

in the stream. The IOP is connected directly to memory and is started in the same manner as any other processing stream. ACES makes no distinction, for dispatching purposes, between a task to perform I/O and a computational task. The Dispatcher places any type task into execution, based upon priority and availability of appropriate modules. While Figure 4-47 depicts an I/O stream in the simplex mode, I/O streams may be simplex, duplex, or TMR, as required.

The Full Processing Stream (FPS) is designed to satisfy the need for a computational task to occasionally output small quantities of data within a limited time frame. Figure 4-47 depicts the FPS in a simplex mode. The IOP is directly connected to the CPE and is dedicated to it. The IOP cannot be time shared by other CPEs or bulk transmissions in this mode. This makes for inefficient use of an IOP since its utilization is low relative to the total task time. Another disadvantage is, that additional buses are required to connect the CPE for just this one type of processing stream. However, it will be considered a required stream until a better solution has been identified.

4.9.3 Residency of Input/Output Control

The Input/Output Control functions, such as device scheduling and manipulation, are not anticipated to reside in BOSS. There are many reasons for this.

First of all, it is desirable to off-load BOSS as much as possible to prevent BOSS from becoming unnecessarily complex, and to prevent BOSS from becoming a systems bottleneck.

Secondly, it is not necessary to control I/O from BOSS as failures of I/O scheduling and manipulation are recoverable and do not cause loss of systems control in contrast to other, previously discussed, Control Executive Functions.

Last, but not least, I/O requirements may differ considerably from mission to mission. Input/Output Control should, therefore, be as adaptable as possible and should, therefore, not be an integral part of BOSS.

The question remains whether the Input/Output Control is predominantly executed on a CPE or an IOP. This, of course, is highly dependent on the selected IOP design concept. Most desirable would be to provide these functions as an integral part of the IOP. It is most desirable to off-load the CPEs and maximize adaptability to mission requirements (by tailoring IOPs to missions).

4.9.4 I/O Summary

ACES thus contains no specific I/O Control other than the ability to configure I/O oriented streams and schedule I/O oriented tasks.

It is recommended that intelligent IOPs be considered to obtain a true and relatively simple application of modularity to mission adaptability as well as to obtain additional systems efficiency.

5. MAJOR IMPACTS ON BASELINE SUMC

5.1 Processor Speed

Speed of a single processor module within ARMMS has been estimated to be 1 - 1.8 MIPS.

(Roughly) Estimated speed of SUMC is currently .5 MIPS (add equivalents). Maximum estimated speed with near future technologies is estimated to be 1.5 MIPS. This is limited by the number of logic levels, and can therefore not be increased without a redesign.

The following information was derived early in the contract (Task II) and therefore may not match the information described in the previous sections (Tasks IV and V). However, as it fulfilled its intended purpose, no attempt has been made to update this information.

5.2 Basic Instruction Set

The currently proposed set resembles an IBM/360 subset. As the SUMC has some flexibility because of its microprogrammed control, the exact makeup of the current instruction set is not of major significance. What is of significance, is the amount of flexibility actually realized through the microprogrammed approach.

Memory interference and bus traffic considerations in ARMMS suggest that the ratio of the instruction execution time and the instruction access time should be as high as possible.

This ratio should be emphasized in the design of the instruction set by the following groundrules.

- o The instruction length of the majority of the executed instructions should be short.

As this is mainly accomplished by limiting the size of the addressing fields, the instruction set should be heavily register or stack oriented.

- o The instructions should be powerful.

By increasing the amount of "work" performed within a single instruction, the ratio mentioned above can also be improved. Potential powerful instructions, applicable to the ARMMS environment, include those that perform functions such as:

Matrix/Vector operations

Bit(s) insertion/extraction

Table Search

Local Executive functions

Mathematical functions

Logical testing of bit combinations

Although a detailed study of the microprogram flexibility of the SUMC has not been performed, initial information is available that at least

indicates that this is likely to be an area of concern. An MSFC internal report is available that lists: "Instructions which cannot be microprogrammed without hardware modification". This list includes (IBM/360 type) instructions such as:

Load Multiple

Store Multiple

Pack

Unpack

Test under mask

Translate and test

Test and set

This would indicate some limitations, inherent to the SUMC data flow, that may be too restrictive for the intended use.

5.3 SUMC/BOSS Communications

This section describes the main characteristics of the communication path(s) between BOSS and the individual processors, necessary to perform the Executive functions.

5.3.1 General Description

The communication is functionally depicted in Figure 5-1. The physical implementation can, of course, be performed in numerous ways.

Regardless of the actual implementation, the TMR (DUPLEX) mode of operation constrains the communication process as follows:

- (1) BOSS must be able to synchronously start each processor in a TMR (DUPLEX) configuration.
- (2) BOSS must be able to vote (hardware or software) on requests made from processors in a TMR (DUPLEX) configuration.

The first item above requires a means of synchronously starting a TMR (DUPLEX) task at the same microinstruction (clock time) for those processors used in the TMR (DUPLEX) mode. Each processor can be primed individually over a common data bus, but all processors must start execution at the same time to guarantee the integrity of the data for the voting elements.

The second item is more complicated. In this case a TMR (DUPLEX) set of processors needs to communicate with BOSS. BOSS must guarantee that the processors are all making the same request and no failure has occurred during execution. A hardware and a software solution are outlined. In each case assume all tasks are in a TMR (DUPLEX) mode.

In the software solution an interrupt decoder is sequentially scanning each processor for an interrupt condition (See Figure 5-2). When an interrupt is decoded, BOSS is notified which processor has sent the interrupt and conditions that processor to send the data associated with that interrupt. BOSS then places the interrupt honored processor in an idle state, and allows the interrupt decoder to search for the next sequential interrupt. In a TMR (DUPLEX) mode two (one) interrupts should be pending if no error has occurred. BOSS reads the

SUMC/BOSS COMMUNICATIONS

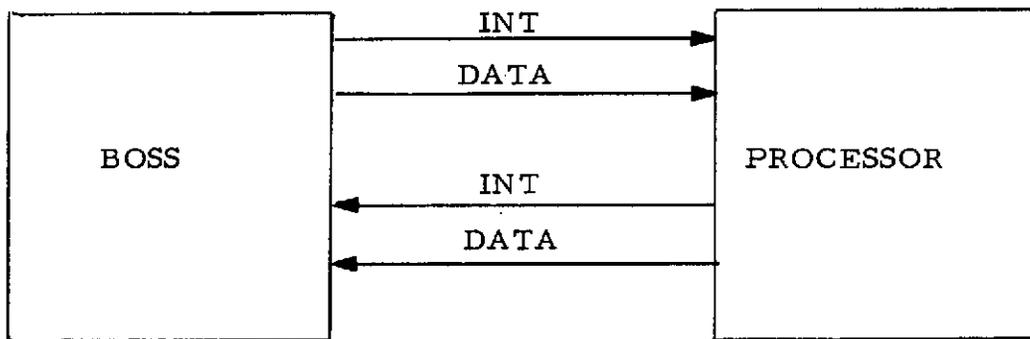


Figure 5-1

INTERRUPT DECODER

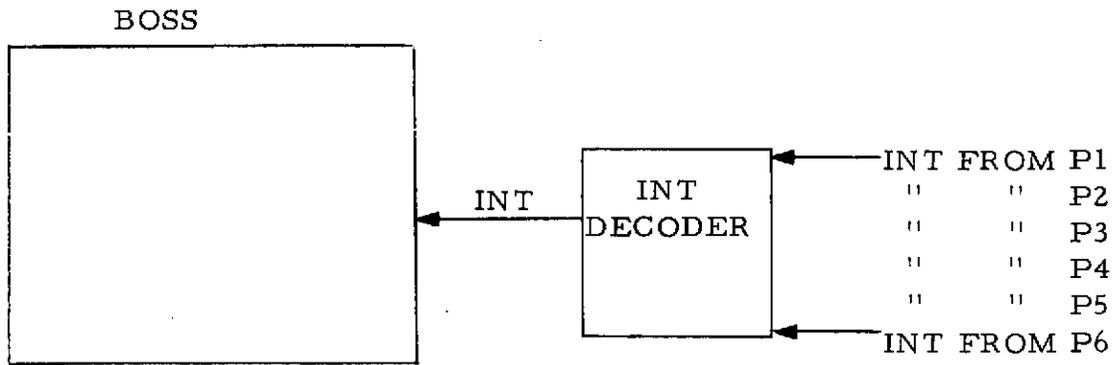


Figure 5-2

data associated with the other interrupts, forcing each processor in the idle state. BOSS then verifies via software that the TMR (DUPLEX) set has made the same request. If no error is detected, BOSS issues a start to the TMR (DUPLEX) set. If an error is detected, BOSS turns control over to the Configurator.

The hardware solution requires a switch between the processors and BOSS (See Figure 5-3). BOSS configures the switch to a TMR (DUPLEX) mode when processors enter this mode. All requests are voted (compared) by the switch before BOSS is notified of a request.

The voting of interrupts and data to BOSS could be performed by the same voting element that provides voting of data between processors and memory as outlined in baseline 1. Figure 5-4 depicts such dual use of the voting switch. In this configuration the outputs of the processors are gated to memory and BOSS. An interrupt to BOSS or a memory request determines the destination of the data from the processors. In this case BOSS would need an interrupt detecting network to decode the interrupt and a multiplexer to allow the data associated with that interrupt to be strobed into BOSS.

5.3.2 Hardware Modifications to SUMC

Figure 5-5 depicts the hardware modifications to SUMC to provide communications with BOSS.

- o Parity Generator

A parity generator should be added to generate parity for data to BOSS.

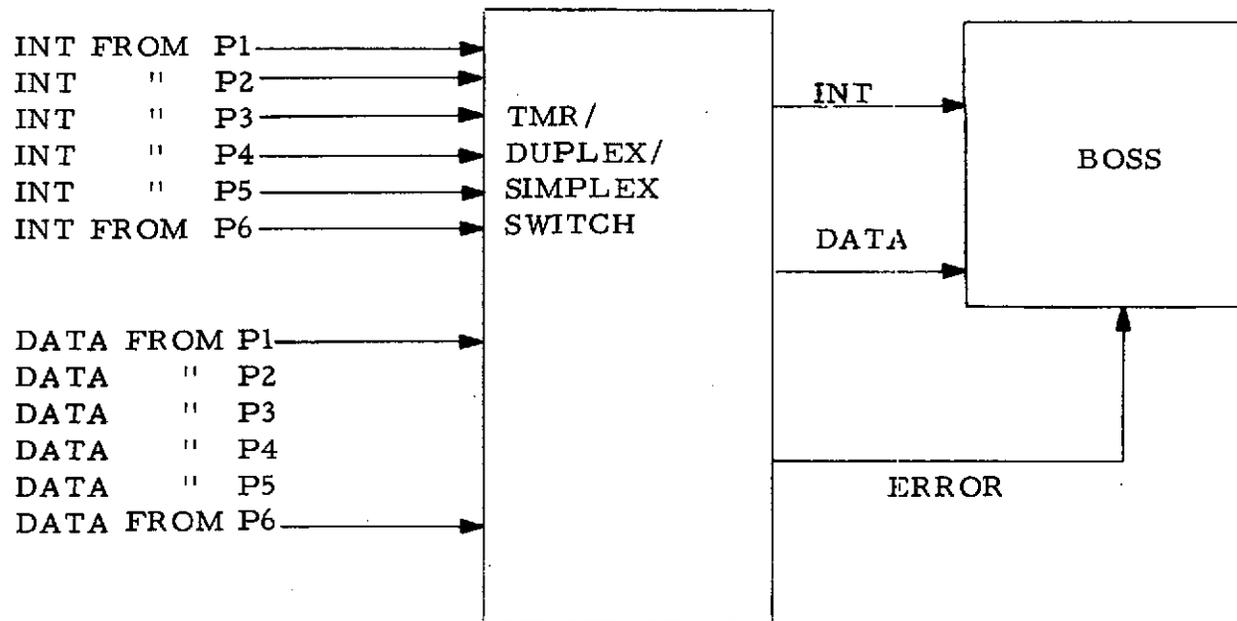
- o Parity Checker

Parity should be checked on data from BOSS and the control section should be notified in case of an error.

- o Control Section

Microprogram control to send and sense an interrupt and acknowledge from BOSS needs to be added.

PROCESSOR TO BOSS COMMUNICATIONS



3-150

Figure 5-3

D

MEMORY/BOSS/PROCESSOR COMMUNICATIONS

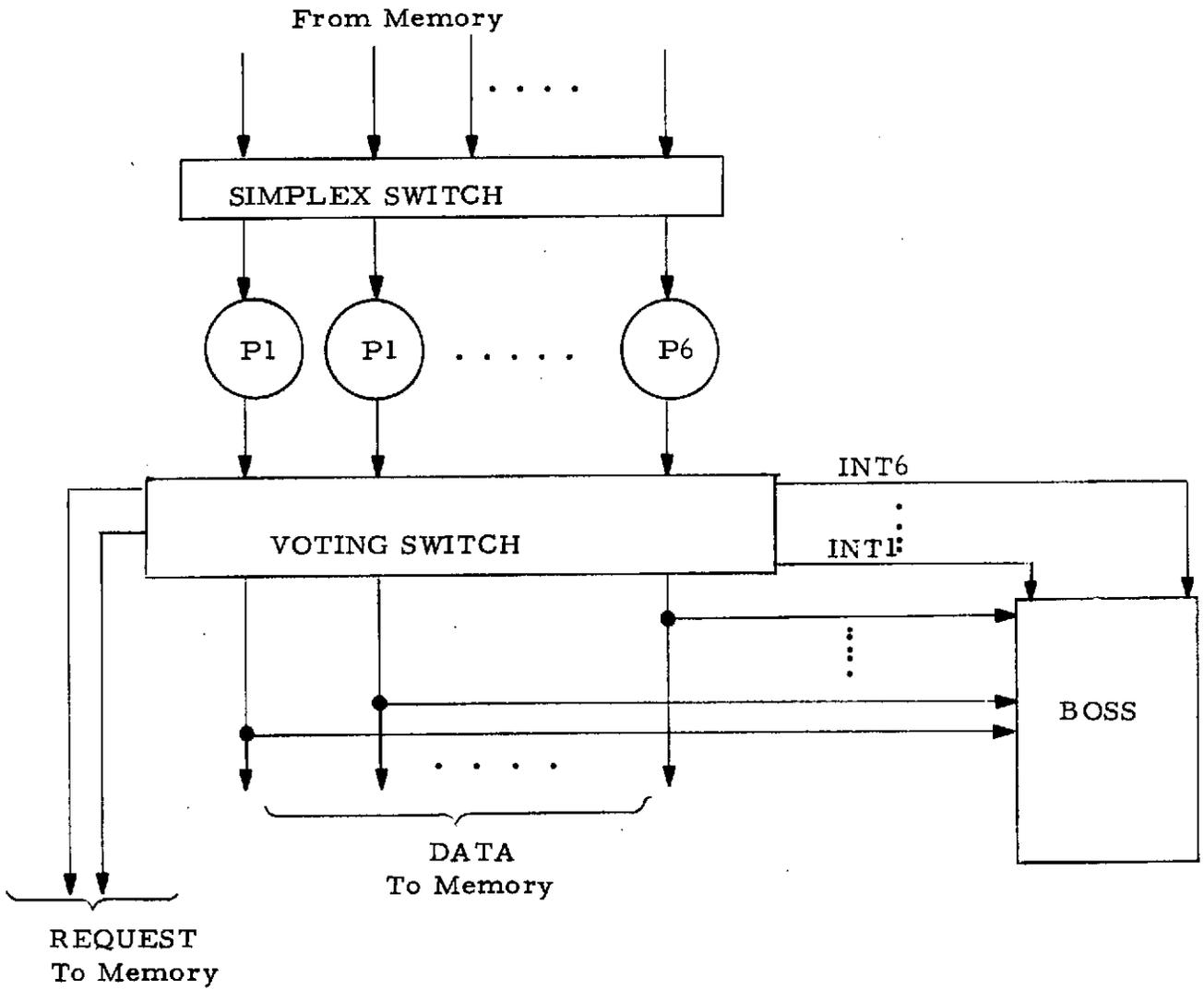


Figure 5-4

HARDWARE MODIFICATION TO SUMC
FOR BOSS COMMUNICATIONS

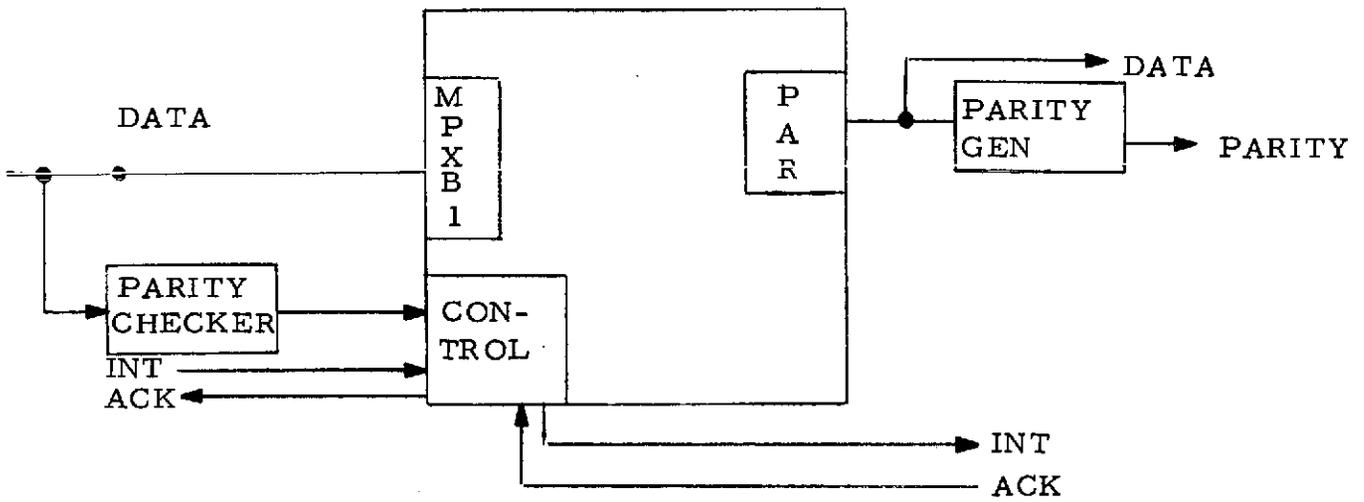


Figure 5-5

5.4 SUMC/BOSS Control Commands

Communication between BOSS/SUMC is performed through the execution of Control Commands. This section provides a potential approach to estimate potential impact on the SUMC microporgram. For simplicity of description, BOSS is assumed to be a microprogrammed unit.

5.4.1 Fetch Cycle

Figure 5-6 depicts the current microinstruction sequence for the fetching and execution of machine instructions. Before each instruction is executed the microprogram looks for an I/O (interrupt) request. At the present time three different interrupts are wired into the SUMC hardware. If BOSS processes all interrupts, these three interrupts will be sufficient for BOSS to communicate with each processor. At the present time the overhead to decode the instruction is 5 microinstructions and the execution time requires from 1 to 27 microinstructions. Therefore the maximum time to decode the interrupt would be 32 microinstructions and typical would be 4 microinstructions. These times do not include any waiting on memory.

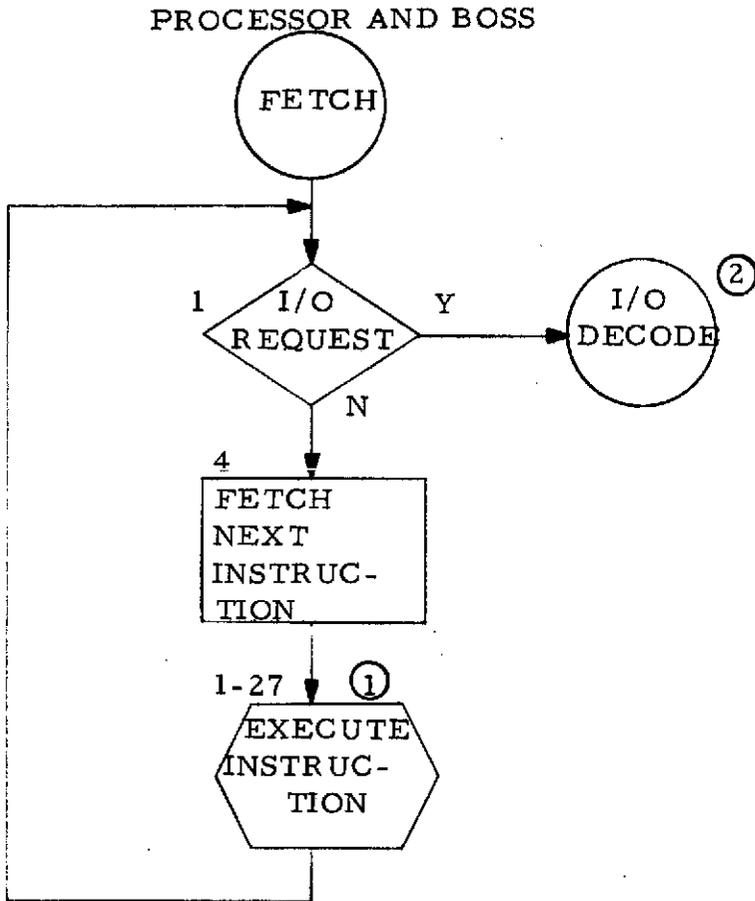
5.4.2 Task to Control Executive Commands

A preliminary list of Task Control Commands was described in paragraph 4.1.

1. SCHEDULE
2. DELETE
3. TERMINATE
4. ABEND
5. SUSPEND
6. SIGNAL

To conserve microprogram memory and provide an expandable format, the processors will communicate with BOSS by a "CALL" instruction.

FETCH CYCLE



NOTE ① NO WAIT FOR MEMORY INCLUDED
② TIME TO I/O DECODE
MIN 1
MAX 32
AVER 3 to 4

Figure 5-6

CALL, type, N

Data

.

.

.

Data

where CALL is the instruction mnemonic to transfer to the microprogram memory, type is one of the six commands listed above (or others), and N is the number of data words (conditions) associated with this command.

The CALL instruction requires 6 microinstructions in the processor [See Figure 5-7 (abbreviations listed in Figure 5-8)]. The instruction will send an interrupt to BOSS and wait for an acknowledge from BOSS. The processor will loop waiting for BOSS to honor the interrupt. When BOSS honors the interrupt an ADDRESS is gated to the MAR. The ADDRESS in this example is that of its own dedicated memory, although the structure could be changed to use SPM or main memory. The Control Executive will need to maintain this address. BOSS and the processors continue to transfer data by a sequence of interrupts and acknowledges.

At the completion of the transmission BOSS transfers to a COMMAND TYPE routine. This routine is not shown. Two approaches are possible. In the first case BOSS takes appropriate action before returning to its fetch cycle. This approach requires less hardware but has the disadvantage of locking out all external interrupts. In the second case the processor interrupt is treated as an interrupt level and the PSW would be exchanged to reflect an interrupt level. This approach would allow higher level interrupts (errors) to be honored before the CALL sequence is terminated.

5.4.3 Control Executive to Processor Commands

Several commands from BOSS to the processor have been assumed for the design of the Control Executive software.

1. SAVE
2. STOP
3. RESTORE
4. START

The microprograms for these instructions will now be described.

CALL INSTRUCTION

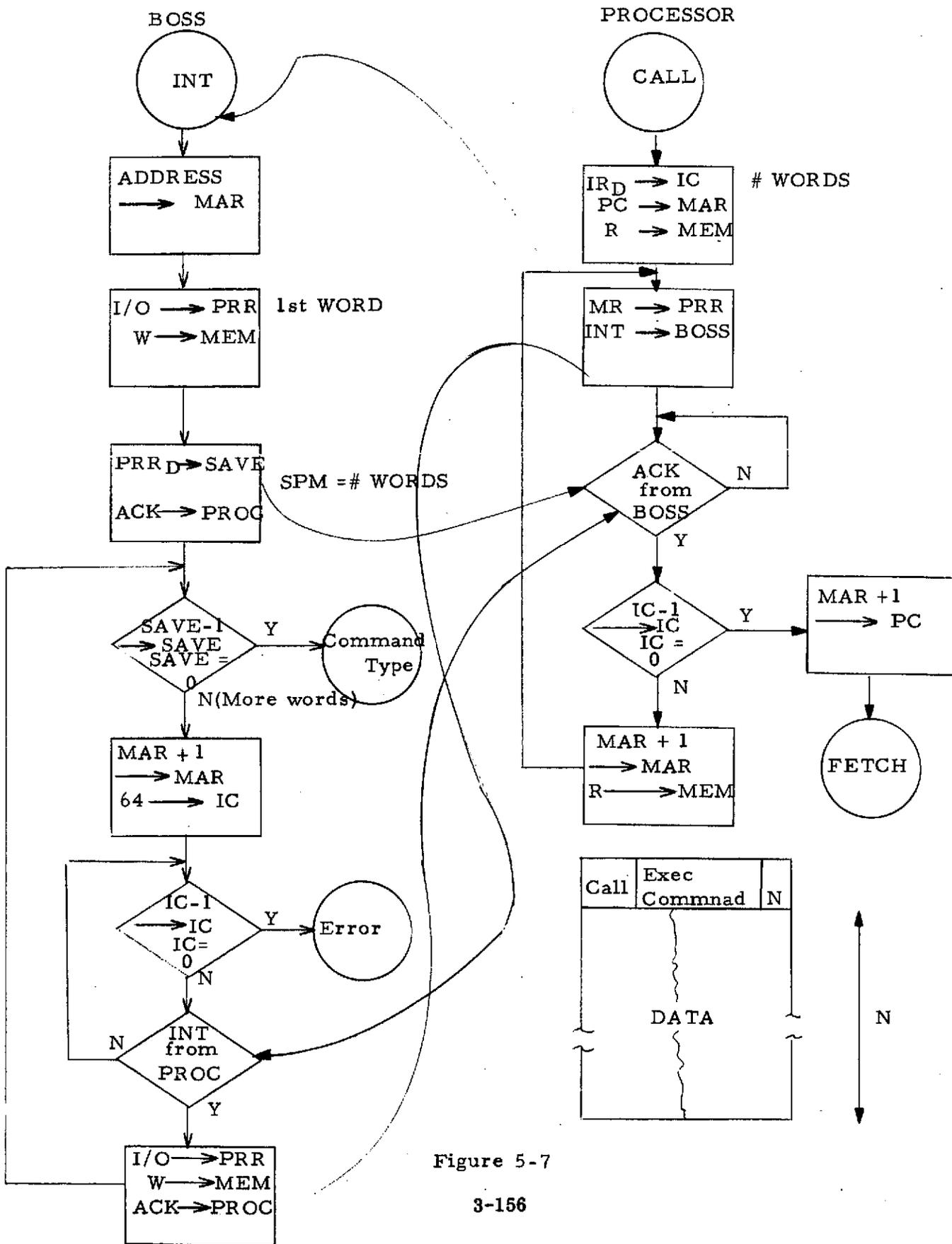


Figure 5-7

MICROPROGRAM ABBREVIATIONS

A0	-	GENERAL REGISTER 0
A1	-	GENERAL REGISTER 1
A15	-	GENERAL REGISTER 15
ACK	-	ACKNOWLEDGE
D	-	DISPLACEMENT
IC	-	ITERATION COUNTER
INT _X	-	INTERRUPT TO PROCESSOR X WHERE X = 1, 2, . . . , 6
IR	-	INSTRUCTION REGISTER
MAR	-	MEMORY ADDRESS REGISTER
MEM	-	MEMORY
MR	-	MEMORY REGISTER
PC	-	PROGRAM COUNTER
PROC	-	PROCESSOR
PRR	-	PRODUCT REMAINDER REGISTER
PSW ₁	-	FIRST PSW WORD
PSW ₂	-	SECOND PSW WORD
R	-	READ
W	-	WRITE

Figure 5-8

- o SAVE

BOSS sends an interrupt and a main memory address. The processor transfers the contents of the 16 general registers, 8 floating point registers, and PSW to main memory and enters the STOP state (See Figure 5-9).

- o STOP

BOSS sends an interrupt and stop command. The processor enters the STOP state, and loops (looking for an interrupt) (See Figure 5-10).

- o RESTORE

BOSS sends an interrupt and main memory address. The processor loads the 16 general registers, 8 floating point registers, and PSW from the specified starting address (See Figure 5-11).

- o START

BOSS sends an interrupt and a two word PSW. The processor loads the PSW registers and returns to the fetch cycle (See Figure 5-12).

SAVE TO MAIN MEMORY - 16 GENERAL REGISTERS AND
2 WORD PSW & 8 FLOATING REG.

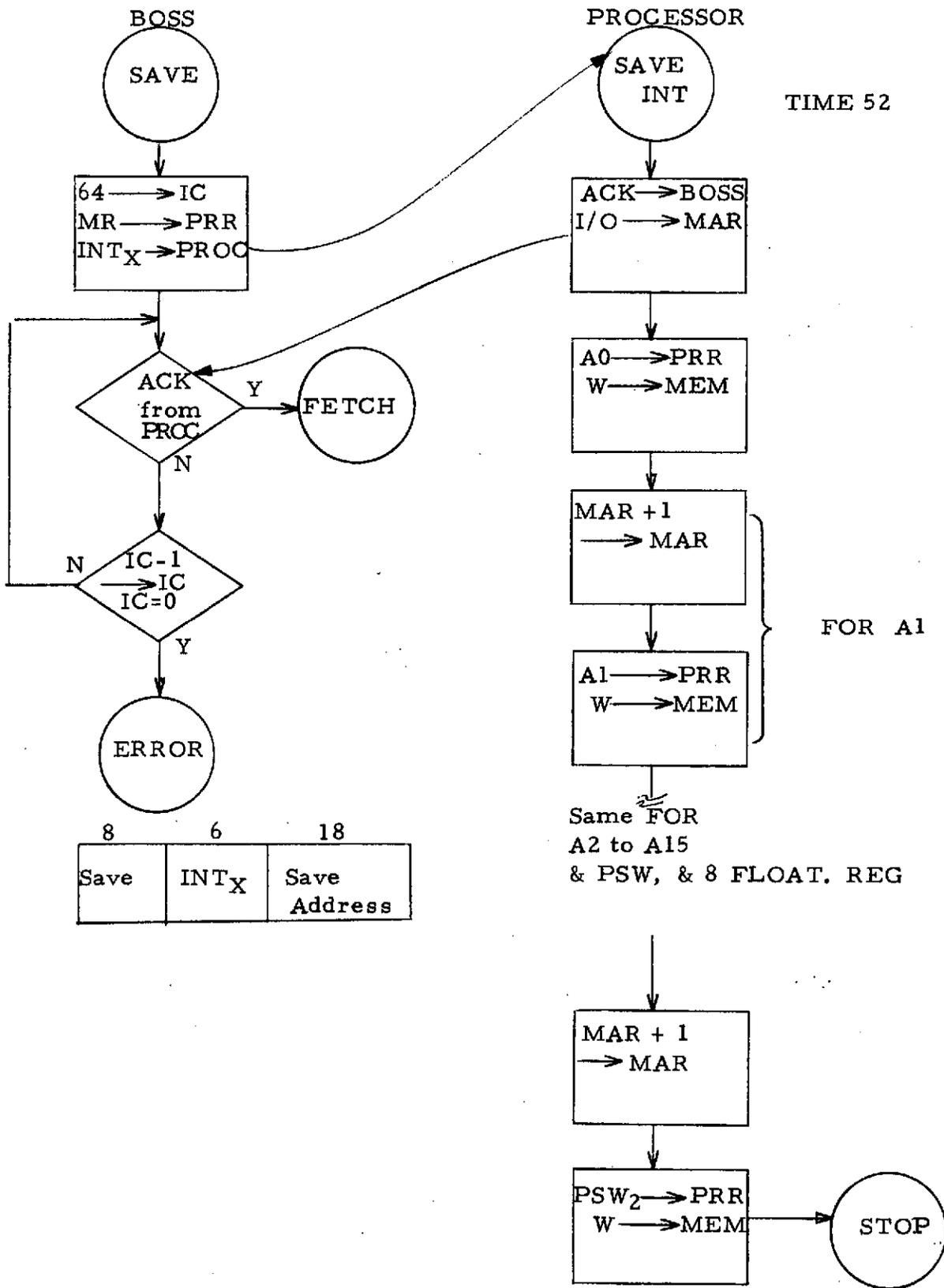


Figure 5-9

STOP INSTRUCTION

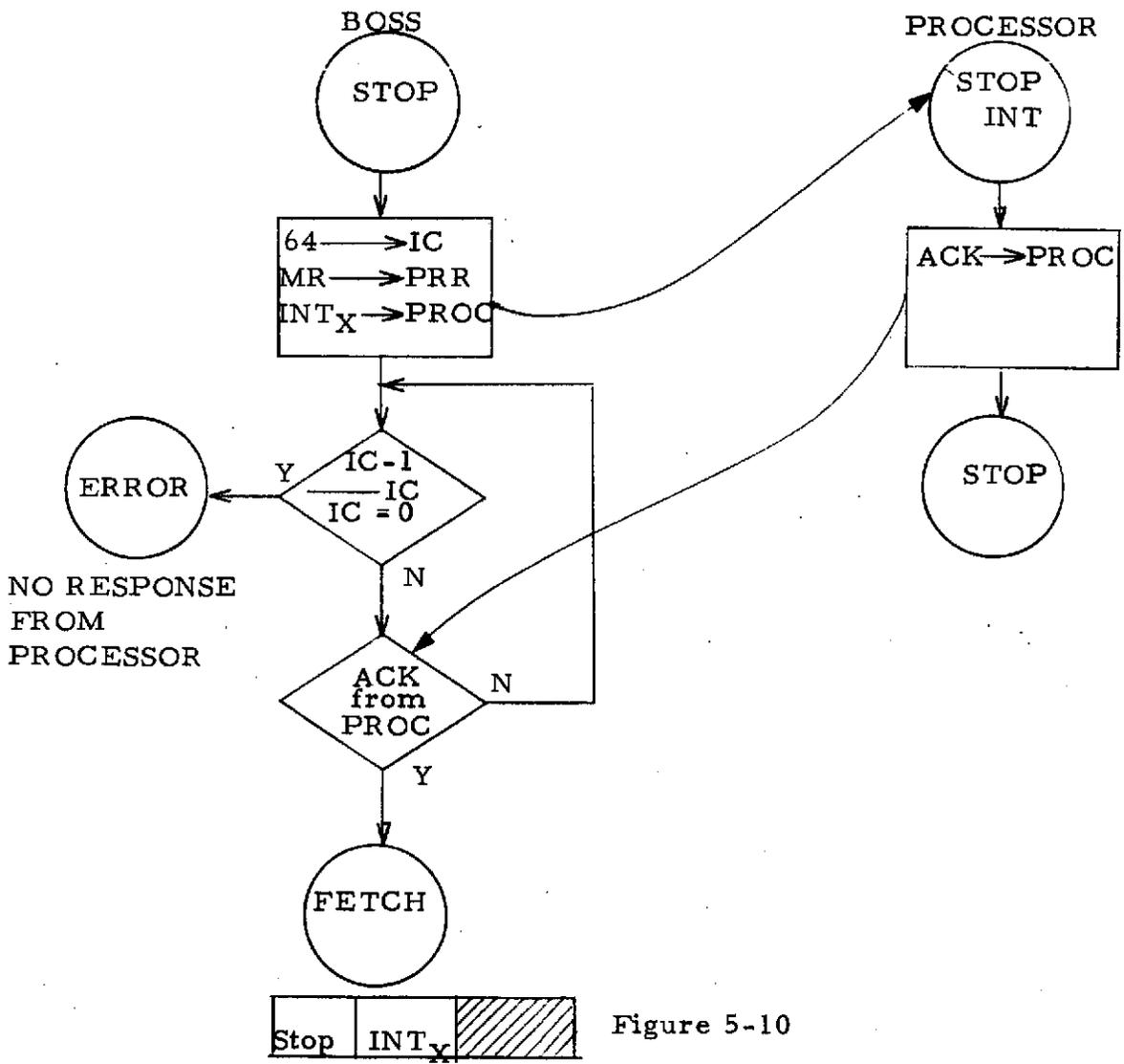
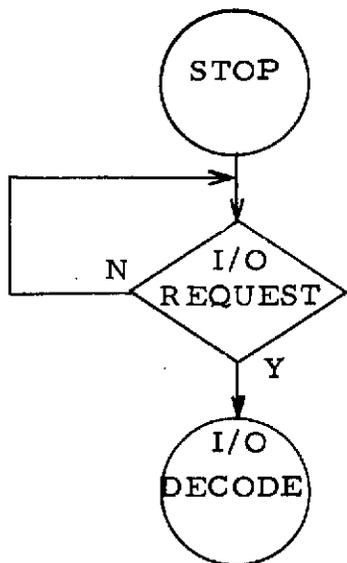


Figure 5-10

RESTORE FROM MAIN MEMORY - 16 GENERAL REGISTERS AND
2 WORD PSW AND 8 FLOATING
POINT REGISTERS

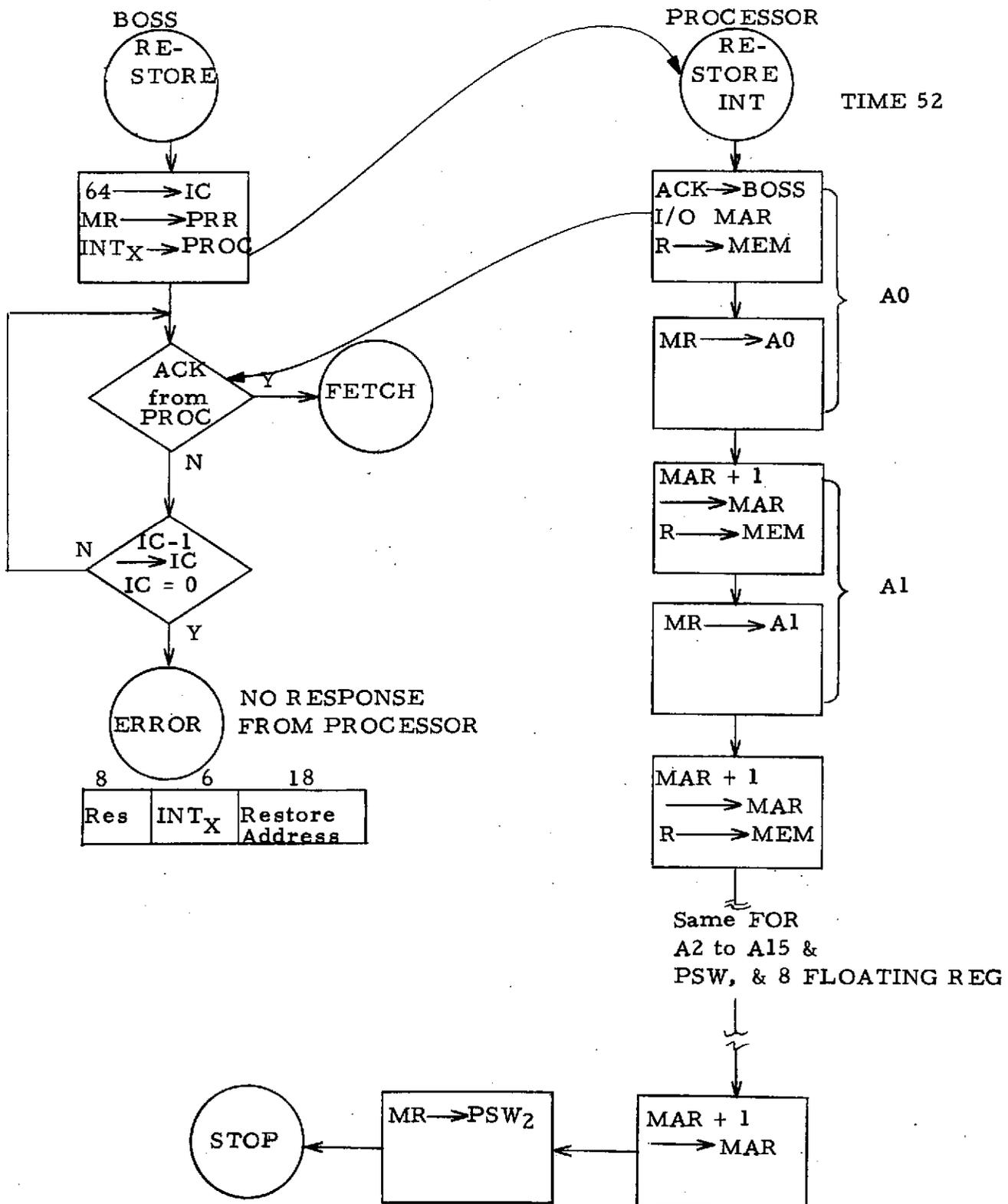


Figure 5-11

START FROM BOSS - BOSS SENDS START INTERRUPT &
2 WORD PSW

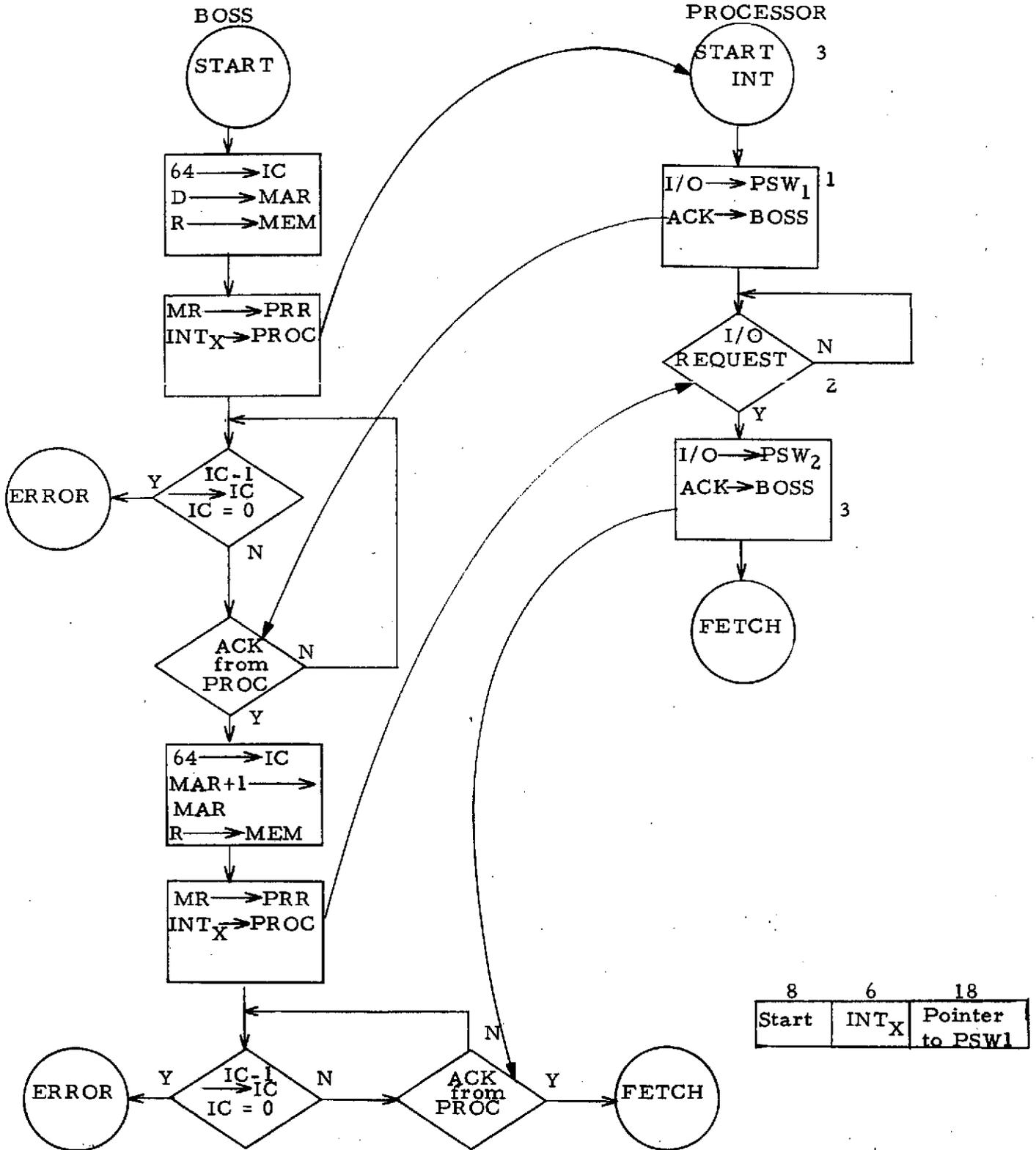


Figure 5-12

5.5 SUMC/Memory Communications

This section describes the memory/processor communications. No attempt has been made to describe the use of a buffer memory. It is assumed that the buffer memory is transparent to the processor control section.

The mapping of logical to physical memory address has not been included. The mapping function requires more study. Several alternatives are possible. BOSS could relocate programs as error conditions occur and/or maintain base registers accordingly. Memory banks could be renamed as errors are detected. Also hardware could be added to map the logical to physical address. The mapping function is embedded in the software (Compiler, Assembler, and Loader) and hardware. Regardless of the translation function the following communication paths apply.

5.5.1 Communication Paths

Figure 5-13 depicts the communication paths between the processor and memory. Parity or other error codes will be generated and checked on transmissions to and from memory. Each request to memory will also contain a key. The key and read/write lock bits in memory will be used to validate accesses. The control signals provide the timing and attention signals to allow data and address to flow between these units.

5.5.2 Error Conditions

Four types of errors can occur in the processor to memory communications.

1. Illegal Access
2. Error on Data to Memory
3. Error on Data from Memory
4. Error in Memory Access

Item 3 will be decoded by the processor. The other items will be decoded by the memory unit. All errors have to be known to the processor for two reasons.

1. In a simplex system there may not be a BOSS

PROCESSOR/MEMORY COMMUNICATION PATHS

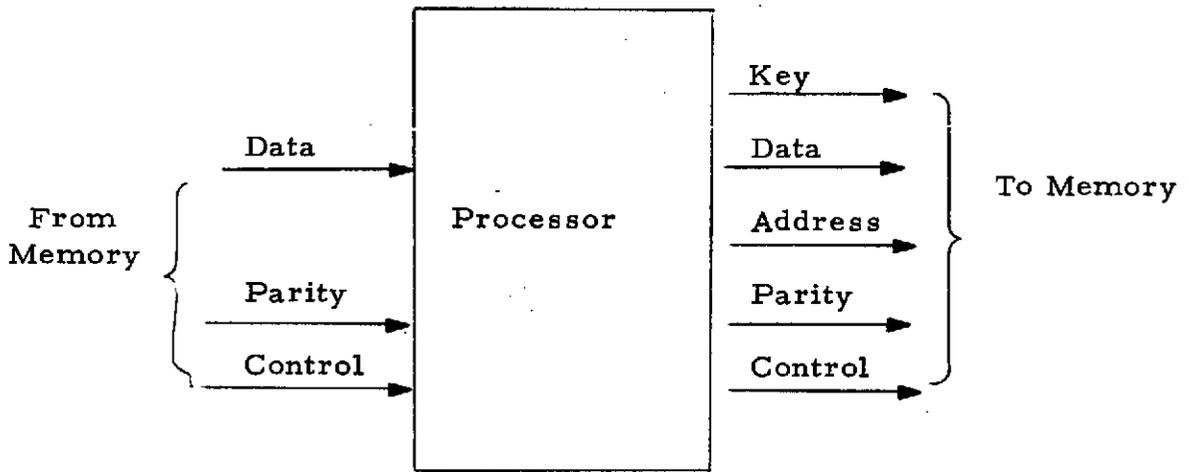


Figure 5-13

2. The exact partitioning of error recovery between BOSS and the processors has not been resolved.

This may be modified (such as only notify BOSS of a memory error) as the error recovery and reconfiguration schemes become more definitive.

5.5.3 Hardware Modifications to SUMC

Figure 5-14 depicts the memory to processor communication.

- o Parity Generator

A parity generator has to be added to generate parity on data and address memory.

- o Protect Key

A method of providing a protect key for the processor and sending the protect key with each memory request needs to be added.

- o Ready Line

A microinstruction control bit has to be added, so the processor will know when memory data is available. Presently the microprograms allow enough time but, in a multiprocessor system, memory contention factors must be recognized.

- o Parity Checker

Parity from memory has to be checked and the control section notified in the event of an error.

- o Memory/Processor Errors

The control section should be modified to allow for recognition of the four memory errors. Probably the four errors should be combined into a memory error violation and further decoded if an error is detected.

MEMORY/PROCESSOR COMMUNICATIONS

3-166

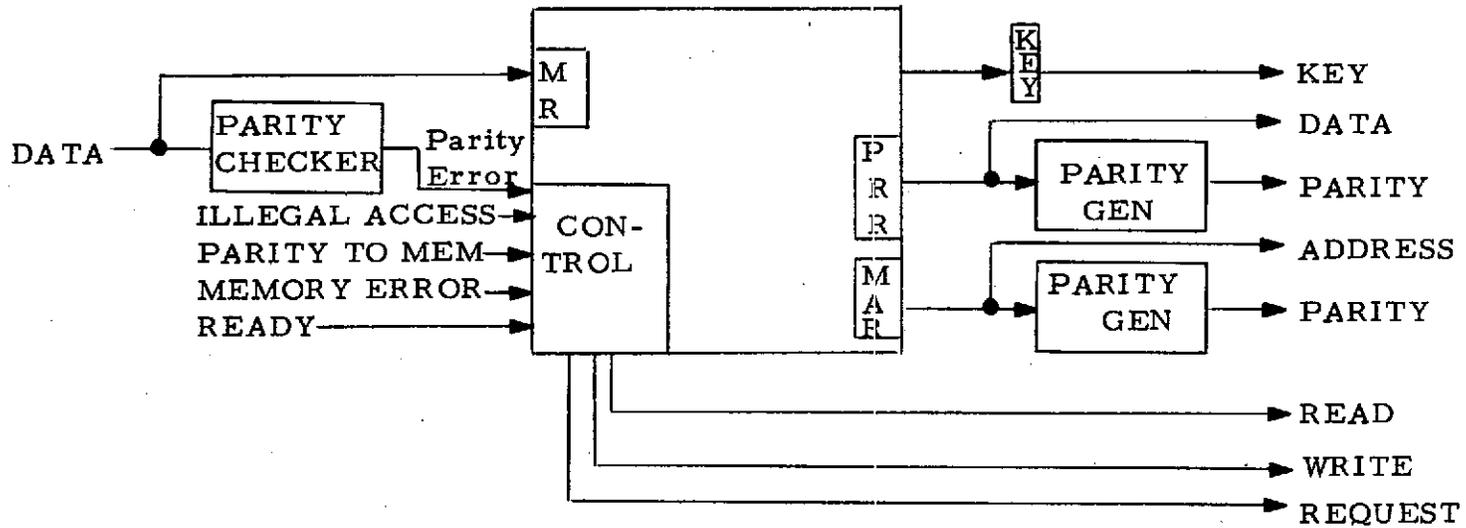


Figure 5-14

5.6 SUMC/I/O Communications

This section describes the I/O/processor communications. The processors will have the ability to set up the I/O units, but all I/O complete interrupts will be honored by BOSS. Figure 5-15 describes the processor communications with I/O. The dotted INT (interrupt) and ACK (acknowledge) signals would be necessary if BOSS was not processing all interrupts. In this case, an interrupt structure for each processor would need to be designed. Therefore the processor design for a BOSS and non-BOSS system is different.

The physical connection of processors to I/O units will be performed by BOSS. BOSS will provide the I/O units for a task before the task is turned over to the processor for executing. BOSS will configure the switches to provide the mode (TMR, DUPLEX, or SIMPLEX). In the event a task requires more than one I/O unit to simultaneously transmit different sets of data, BOSS will notify the processor (task) the address of each I/O unit connected to the processor. The address will allow the task to monitor the state of the different I/O units. If only one I/O unit is needed per task, the task need not know (except for error checking) the physical connection.

5.6.1 I/O Commands

At least three I/O commands are needed.

1. START
 2. STOP
 3. STATUS
- o START

The START instruction will contain the source, destination, number words, keys function code, or pointers to those items which are necessary to initiate an I/O transaction.

- o STOP

This instruction will terminate an I/O operation.

- o STATUS

This instruction will allow the task to determine the state

PROCESSOR/ I/O COMMUNICATIONS

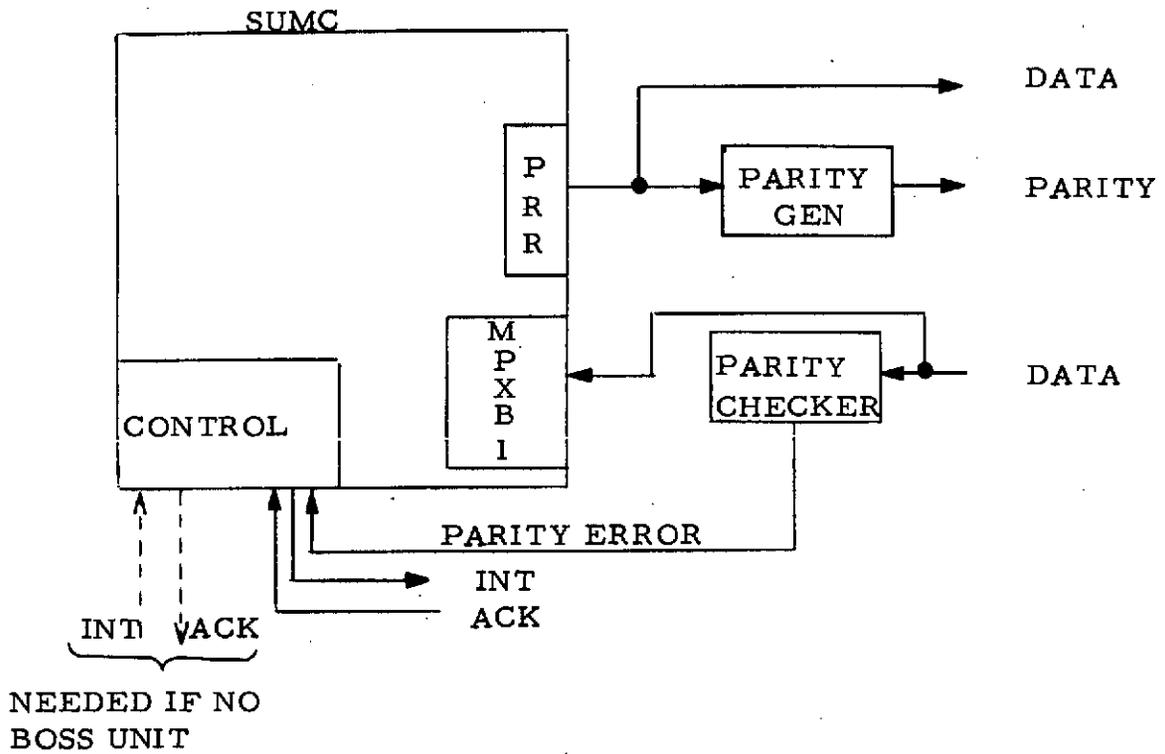


Figure 5-15

(busy, idle) and error conditions for those I/O units connected to that processor.

5.6.2 Hardware Modifications to SUMC

Figure 5-16 depicts the modifications to SUMC.

- o Parity Generator

A parity generator has to be added to generate parity on data to the I/O unit.

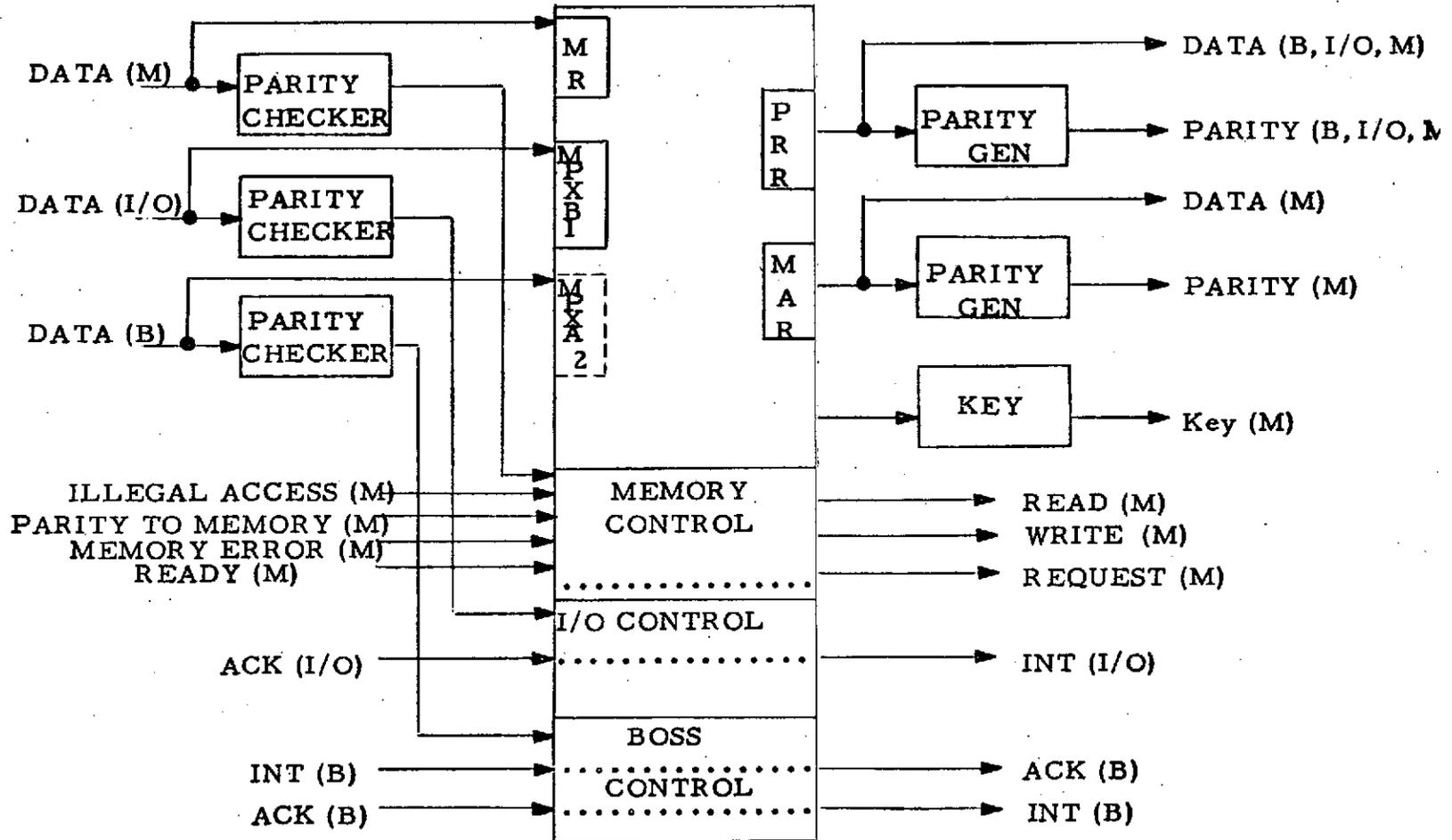
- o Parity Checker

A parity checker for data (status) from the I/O unit needs to be added. Also the control section will be modified to record the parity error.

- o Control Lines

The control section, under microprogram control, should be modified to allow an interrupt to be sent to the I/O unit and a microprogram Sense line on acknowledge needs to be added. The Acknowledge line allows the timing for more than one processor communicating with one I/O unit. The control section should allow for sensing of parity errors on status data from I/O.

SUMC MODIFICATIONS



3-170

B (BOSS)
M (MEMORY)

Figure 5-16

5.7 Summary of SUMC Modifications

This section summarized the modifications and investigates the sharing of functions (see Figure 5-16). The rationale for these changes have been described in previous sections.

A port to multiplexer MPXA2 has been added to allow BOSS to communicate with SUMC.

Parity generation can be shared between units at the expense of speed, however, therefore three parity checkers are shown.

All interrupt lines remain on until an acknowledge is received or sent from the processor. Each interrupt has an acknowledge associated with it.

The read, write, and request control lines already exist in SUMC. The interrupt from BOSS control line can be used under the present interrupt structure.

The following additions are recommended:

1. Generate parity for PRR
2. Generate parity for MAR
3. Check parity on MR data
4. Check parity on MPXB1 data
5. Check parity on MPXA2 data
6. Add control for MPXA2
7. Add control for interrupt to BOSS
8. Add control for interrupt to I/O
9. Add ready from memory sense line
10. Add ACK from BOSS sense line

11. Add ACK from I/O sense line
12. Add Illegal access sense line
13. Add parity to memory sense line
14. Add memory error sense line
15. Add acknowledge to BOSS control
16. Add key on requests to memory

SECTION 4

ARMMS COMPONENT TECHNOLOGY STUDIES - PHASE II

This section deals with ARMMS component technology studies involved in choosing a logic family, data bus technology, and power supply configurations. CMOS is the recommended choice for module internal logic after consideration of all major logic families' projected characteristics for the 1975 time frame. The major advantages of CMOS include: 1) lowest power dissipation of any logic form, 2) excellent noise immunity, 3) high packing density, 4) wide temperature operation, 5) high fanout, 6) easy interface with bipolar circuits, and 7) operation over a wide power supply voltage range. CMOS will be assumed in developing ARMMS packaging characteristics during Phase III.

Bus technology studies placed an emphasis on loading considerations, detection theory, module interconnection methods, and reliability. A current source driver operating into a single-ended isolated receiver over a 50 Ω microstrip line was chosen to provide best power-speed characteristics with simple technology and minimum pin count.

Power supply configurations ranging from a single centralized supply to individual power supplies per module were considered. Since no module was to depend on one power supply, modularization must be effective over a range of ARMMS configurations, and it must be possible to switch supplies on and off under BOSS control, a partially centralized regulator supplying power to up to 5 modules each of which incorporates a DC/DC converter was selected for further detailing in Phase III packaging studies.

4-1

I. ARMMS TECHNOLOGY ALTERNATIVES STUDY

This report has as its objective the evaluation and recommendation of a logic family to be used in the ARMMS computer. This report will consider the adaptability of present logic families in the 1975 time frame. It will review several device technologies, consider interconnection complexity, thermal control, electrical characteristics and blue sky trends in the semiconductor industry in making its recommendation. Both bipolar and MOS technologies will be considered for possible use.

INTERCONNECTION COMPLEXITY

Future generations of computers will most likely be designed utilizing two separate component philosophies.

1. The first will most likely be the simple translation of some of the more commonly used logic functions into MSI or LSI arrays. Those arrays will, of necessity, be flexible, universal and limited to those specific elements which enjoy wide spread industry use. Logic implementation would be accomplished by hard wire interconnection of the standard building blocks to give the desired input output requirements. This system is essentially the same as used today except that the gate or flipflop is replaced with more complex logic functions. The major computer functions will still be constructed with at least one level of interconnection removed from the chip.
2. Concept two will rely on the development of highly specialized monolithic devices with interconnection designed to perform a complete computer function or subfunction (on the chip).

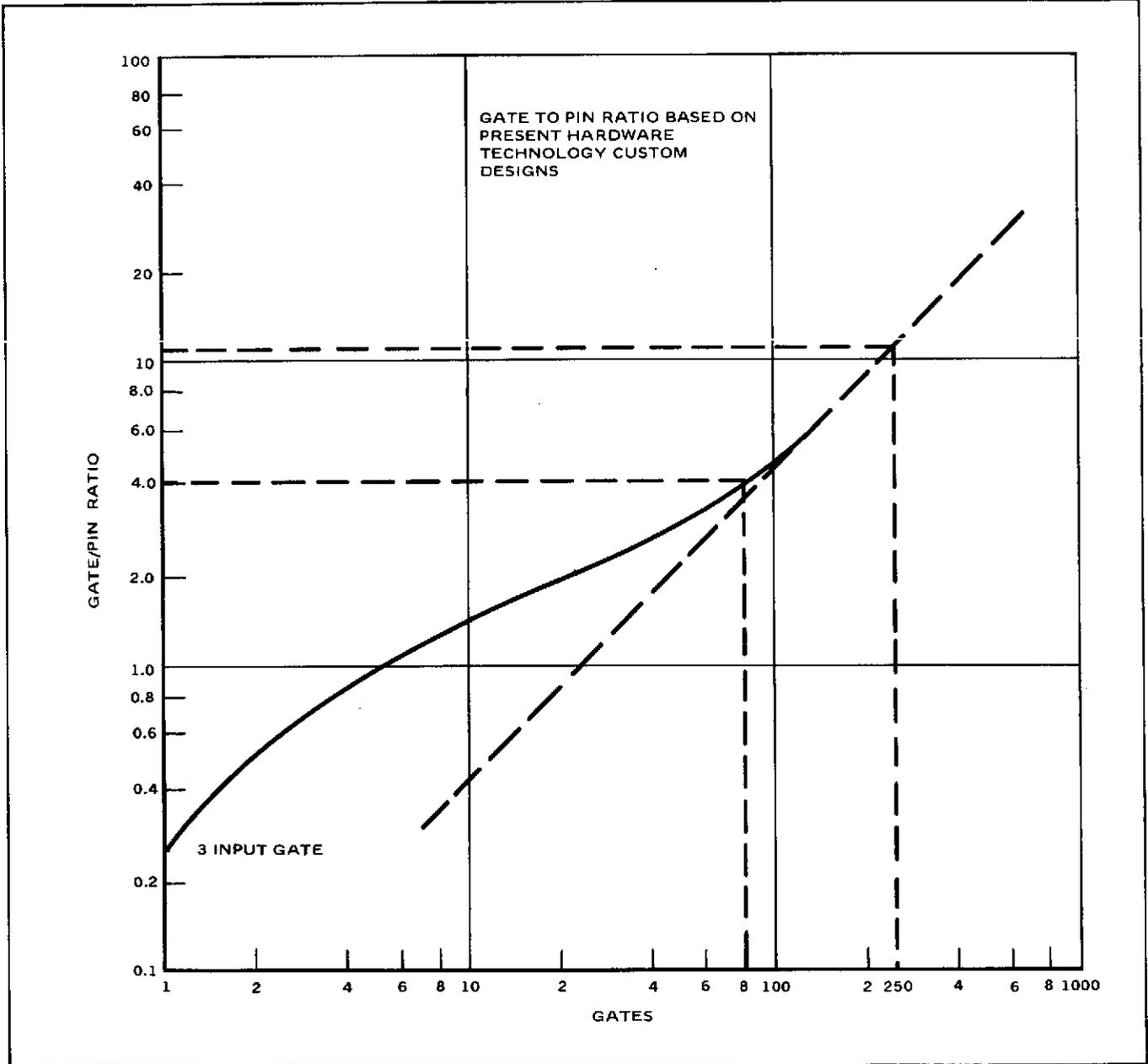
Clearly the two systems have applications where each stands out. System one is particularly useful where design costs, weight, volume and power are not a prime requirement. Applications such as ground equipment, ground test equipment and most airborne computers will use this concept.

System two tends to excel where volume power and weight are of prime importance. Because stray capacitance is significantly reduced, propagation delays are reduced and system speeds increase. Power is reduced by designing each device for its optimum characteristics and not a general set of ground rules.

Estimates of device complexity which will become available in the post 1973 time frame vary widely. This tradeoff will therefore take a conservative approach. For bipolar elements, we shall assume 80 equivalent gates per chip. For MOS devices 250 gates per chip appear to be a practical level of complexity.

Graph I reflects present (1972) gate to pin ratios for several custom and noncustom devices being manufactured today. Reviewing this information, Table I reflects a three-to-one decrease in interconnection points for a 250 gate chip vs. an 80 gate chip. In addition to define a 15K gate processor module, the chip count drops from near 200 to approximately 60 with the higher complexity devices.

32140-25



Graph 1

TABLE I. INTERCONNECTION COMPLEXITY

Logic Family	Gates/ Chip	Gate/ Pin Ratio At Chip	Chips	Interconnection Points Per Chip	Total Interconnection Points Between Chips
T ² L (All)	80	4	200	20	4,000
P. MOS	250	4	60	23	1,380
CMOS	250	4	60	23	1,380

A precise estimate of unit volume cannot be given until the overall packaging concept is defined. The only safe statement at this time is that the interconnection media is the prime user of volume within a unit, and the fewer number of points to be interconnected, the smaller and lighter the hardware will be.

Since low power, weight, and volume figures will be prime consideration in this system, large scale MOS devices are preferred.

POWER DISSIPATION

While increasing the number of gates on a die will tend to increase device speed and decrease interconnection complexity, there does exist definite limitations on the number of gates which may be placed on a die without exceeding tolerable power dissipation levels. The assumption will be made here that this hardware must operate in an uncontrolled (vacuum) environment and, therefore, all component cooling must be accomplished by conduction.

Table II reflects the power dissipation per gate (@ 5 megahertz) and total power dissipation per processor module based upon 15K gates. While the total dissipated per Schottky T²L chip may represent a design problem, it is small compared with the total thermal control problem. The total dissipation of Schottky T²L or 5400 T²L will probably prove unmanageable within the assumed processor volume of approximately 12 cubic inches.

Assuming a maximum case temperature of 100°C on a particular semiconductor and a worst case unit temperature of 55°C (131°F), a Schottky T²L implemented processor module would require a structure thickness of one inch. Clearly, this would present an intolerable situation. For MOS or lower power Schottky T²L, this structure thickness reduces to 0.1 inch, a somewhat practical number. Even if more exotic cooling systems were employed, such as heat pipes, managing the power densities of T²L or Schottky T²L will be a near impossible task.

TABLE II. POWER DISSIPATION SUMMARY

Logic Type	Dissipated Power Per Gate	Processor Module Dissipated Power	No. of Chips/Processor	Dissipated Power Per Chip
Schottky T ² L	19 MW	285 W	200	1.4 W
SN5400	10 MW	150 W	200	.75
Low power Schottky T ² L	3 MW	45 W	200	.225
PMOS			60	
CMOS	2 MW	30 W	60	.50
5% Schottky T ² L	20 MW	15 W		
95% CMOS	2 MW	30 W	45 W	

From a strict power point of view, the low power Schottky T²L, CMOS or PMOS are the only logic families which appear practical.

Electrical Characteristics

Without question, the speed and propagation delays of Schottky T²L, 5400 series T²L, and low power Schottky T²L are within the limits needed for the processor logic family. Consequently, the ARMMS computer could be implemented today using any one of those three devices.

The present CMOS devices, however, have propagation delays several times those of the slowest T²L family. This fact could make processor logic implementation difficult and limit the processor operating space. Figure 1 presents typical speed characteristics for CMOS devices.

The future of CMOS does look bright, however. Semiconductor houses are today developing ION implantation techniques and silicon gate technologies for CMOS to reduce parasitic and junction capacitances. Silicon on sapphire and silicon on spinal substrates will dramatically reduce substrate capacitance associated with bulk silicon CMOS devices. Device speeds in the range of 100 MHz will be possible without effecting the speed-power relationship established in present day hardware. However off chip capacitance considerations limit chip to chip logic speeds to approximately 40 nsec.

FIGURE 10 – TYPICAL RISE TIME versus LOAD CAPACITANCE

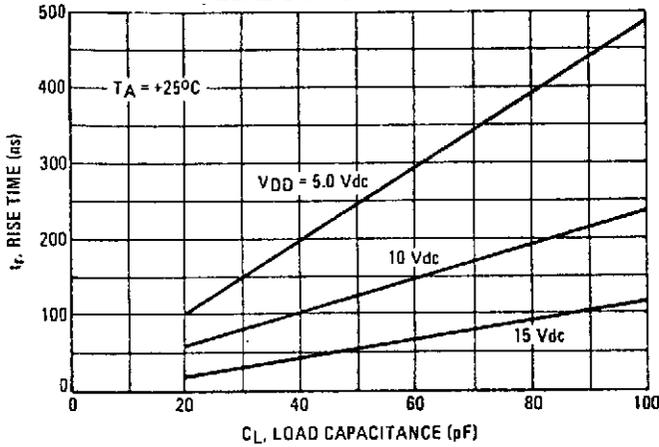


FIGURE 11 – TYPICAL FALL TIME versus LOAD CAPACITANCE

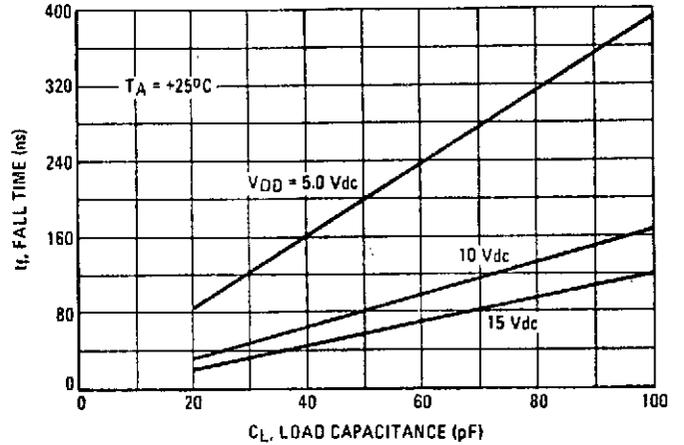


FIGURE 12 – TYPICAL TURN-ON DELAY CHARACTERISTICS

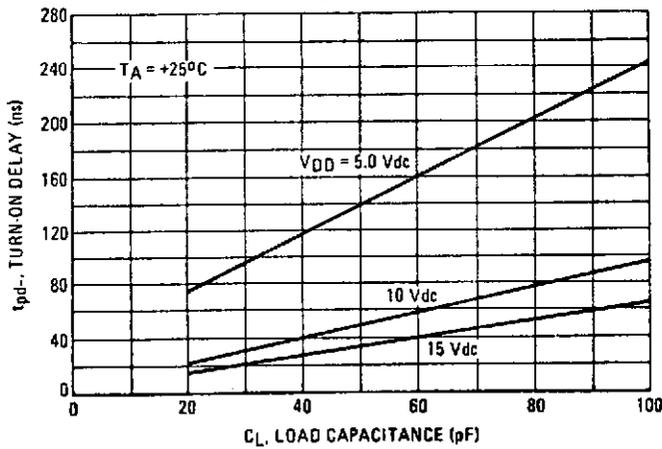
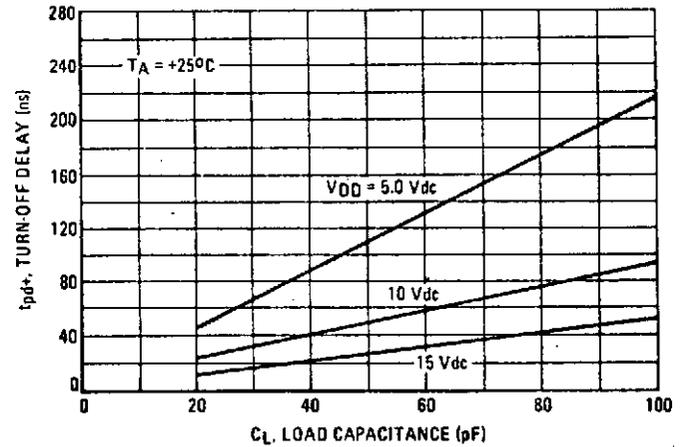


FIGURE 13 – TYPICAL TURN-OFF DELAY CHARACTERISTICS



Typical CMOS propagation, rise time and fall time vs. load capacitance.

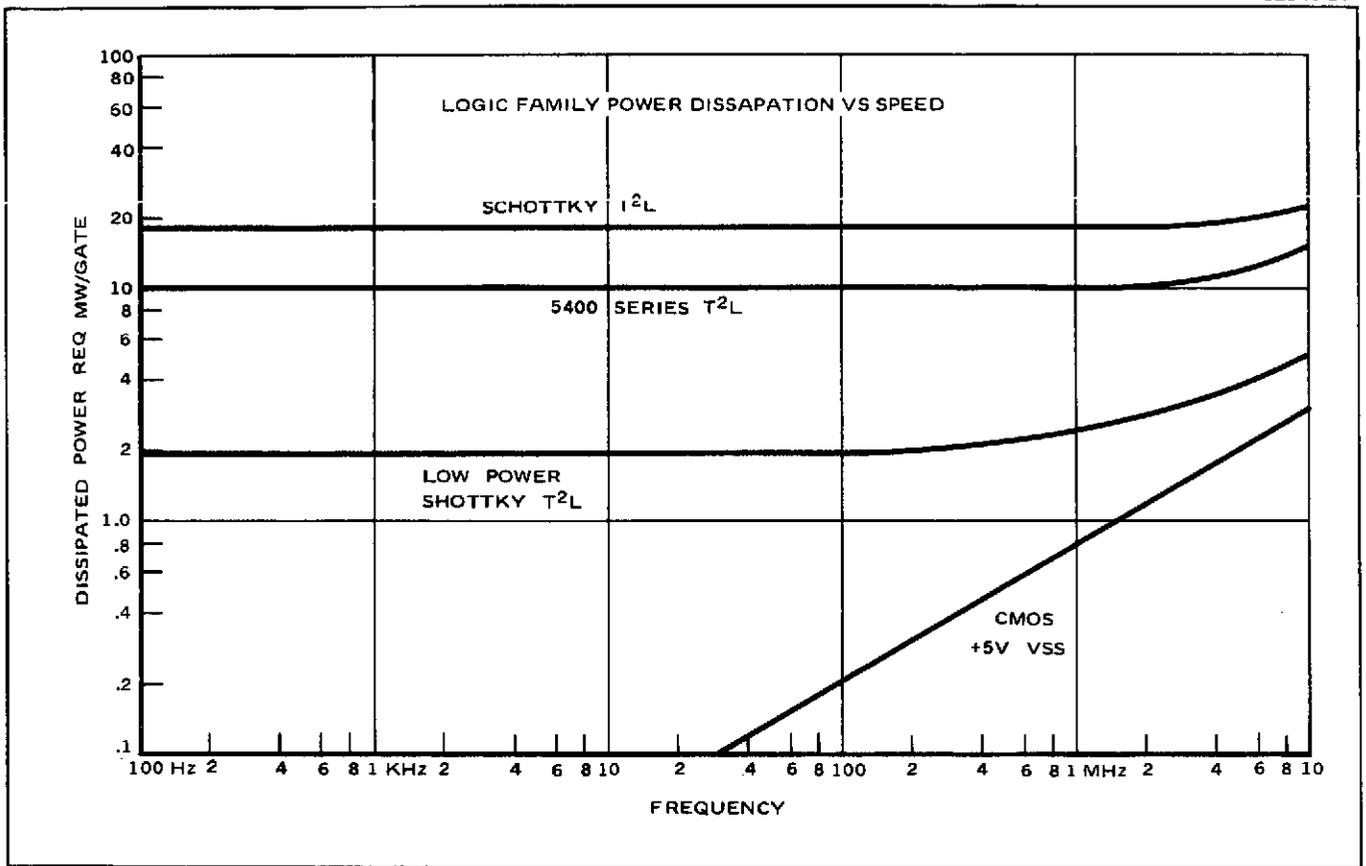
FIGURE 1

F

DTL and RTL logic lines will not be discussed because it is thought that their future in new design is somewhat limited because of their relatively low speed and lack of interest within the semiconductor houses themselves. PMOS is definitely a possible choice, but has been rejected because of the superior speed, lower power dissipation and greater interest in CMOS. The high power levels of ECL combined with the fact that their speed is not needed has caused the elimination of this logic line from consideration.

The advantages of three lines of T²L logic and CMOS logic will be discussed and some of their more important technical characteristics will be explored. Graph 2 and Table III compare the speed power characteristics of these elements and list some of their important electrical parameters.

32140-26



Graph 2

TABLE III. PARAMETER COMPARISON

Parameter	CMOS	Schottky Low Power T ² L	Schottky T ² L	5400 Series T ² L
Power Supply Voltage	4.5-18V	5.0 ± 0.5V	5.0 ± 0.5V	5.0 ± 0.5V
Power Dissipation				
Qui	.01 μw	2 mw	19 mw	10 mw
Dynamic	.4 mw/MHz		.7 mw/MHz	.3 mw/MHz
Propagation Delay	50 ns	10 ns	5 ns	10 ns
Input Current				
Logic 1	10 pa	5 ua	50 ua	40 μa
Logic 0	10 pa	-.1 ma	-2 ma	-1.6 ma
Input Capacity	2.5 pf	2 pf	-	-
Output Impedance Ω's				
Logic 1	750	550	50	70
Logic 0	750	20	10	10
Switching Threshold	.45 V _{dd}	1.4V	1.4V	1.5V

T²L logic has become an extremely popular logic element over the past several years. Its speed and the availability of complex devices from numerous sources, combined with the ability to manufacture devices with as many as 80 equivalent gates, makes T²L a possible logic family for the ARMMS computer. T²L's advantages and disadvantages are listed in Table IV.

T²L is the fastest saturated logic family of digital elements available today. As a result of its speed, transmission line problems such as ringing and line reflections can cause false triggering problems. Many manufacturers now incorporate a clamp on the input lines to clamp any negative signals to a level which will not cause false triggering.

The 5400 series (Figure 2) of T²L has typical propagation times of 10 ns per gate and is capable of operating with clock speeds of 20 megahertz. Power dissipation of 15 mw/gate is common at this speed and decreases at a rate of approximately 0.3 mw/MHz to a quiescent level of approximately 10 mw/gate.

Typical power dissipation of approximately 12 mw per gate can be expected at clock rates of approximately 5 megaHz, the operating speed of the bulk of the logic within a processor module.

TABLE IV. ADVANTAGES AND DISADVANTAGES OF T²L LOGIC DEVICES

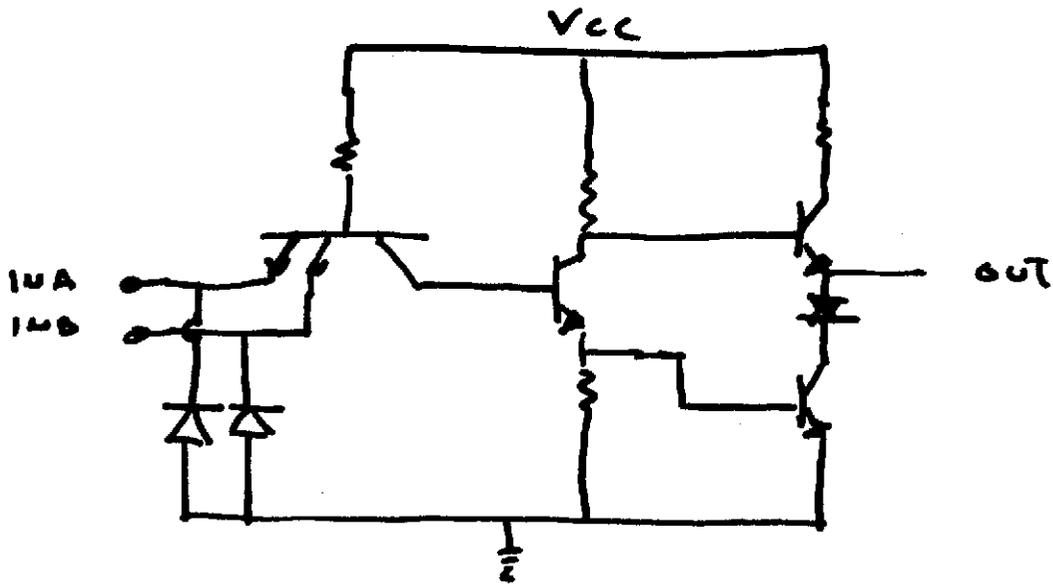
TTL Advantages:

1. A large number of different devices and complex functions are available.
2. Low output impedance results in superior drive capability.
3. TTL has very good immunity to externally generated noise.
4. TTL results in a better system speed-power product than is obtainable with other saturated forms of logic.
5. High-speed capabilities allow more work to be accomplished in a given amount of time.
6. Multiple sources and extensive competition have resulted in low prices.
7. Compatible specifications allow the mixing of various families for optimum designs.
8. Long history of reliability.

TTL Disadvantages:

1. TTL has high values of di/dt and dv/dt, and therefore more care is required in the layout and mechanical design of systems.
 2. TTL generates "glitches" when switching, and thus additional capacitors are required for bypassing.
 3. The "implied and" function is not available by tying outputs together.
 4. Power dissipation in high density in all but low power Schottky T²L.
-

Sizing a processor module at 15K gates, all operating at 5 megabit, we may calculate a power dissipation of approximately 180 watts. Assuming a module size of 12 cubic inch, it is obvious that this amount of dissipated power would make thermal control a near impossible problem. For this reason alone, 5400 series T²L will be unacceptable for use as the logic family for the bulk of the logic within the computer.



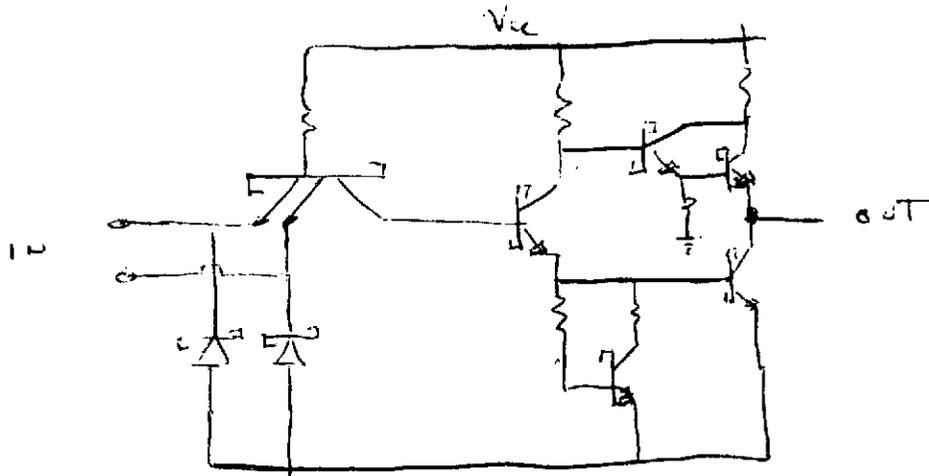
Input voltage	$5 \pm 1/2$ V.
Fanout	10
Propagation delay	
0-1	22 ns max.
1-0	15 ns max.
Dissipated power	10 mw/gate quiescent .3 mw/MHz/gate dynamic
Noise immunity	1V typical

TYPICAL 5400 SERIES T²L GATE

FIGURE 2

At the data bus interface, extremely high speed logic elements are desirable (Figure 3). To live within the 73 ns worst case time budget, high speed Schottky T²L may be necessary at the interface between the processor logic and the data bus.

		Low Power
Input voltage	5 ± 1/2 V	5 ± 1/2 V
Fanout	10	10
Propagation delay 0-1	4 1/2 ns max.	
1-0	5 ns max	
Dissipated power quiescent	19 mw	2 mw
dynamic	.7 mw/MHz	



Noise immunity 1 volt typical

TYPICAL SCHOTTKY T²L GATE

FIGURE 3

Quiescent power dissipation of Schottky T²L is approximately 19 mw per gate and increases at a rate of approximately .7 mw/megahertz. At 5 magabit clock rates, gate dissipation is approximately 22 milliwatts. Typical propagation delays of 3 ns are common with Schottky T²L with clock rates up to and exceeding 125 megahertz.

The increased speed of the Schottky device is accomplished by not allowing transistors within the device to go into saturation. With the transistors kept out of saturation, the storage time can be eliminated. The incorporation of a Schottky diode (during the collector isolation diffusion) across the base to collector junction performs the function of not allowing the transistor to saturate. The Schottky diode has the advantage of eliminating storage time but does add a significant amount of capacitance to the circuit. Because storage time is eliminated, primary delays are due to RC time constants on the chip. To overcome these delays, Schottky devices operate at somewhat higher current levels than 5400 series T²L elements. This, combined with unequal turn on and turn off times of Schottky transistor, results in a higher power dissipation with frequency than a standard T²L.

Lower Power Schottky T²L

The low power Schottky T²L logic elements are the only bipolar devices which can compete with MOS for the bulk of the logic in the low speed portion of the processor module. This line of logic elements is fully capable of operating at 30 MHz with a reasonably low gate dissipation of approximately 5 mw at this speed. At speeds of approximately 5 MHz, gate dissipations should run approximately 3 mw rivaling that of CMOS. Arrays of 60 gates are already available and prospects for 80 gate arrays for custom chips still seems good.

Assuming a device complexity of 80 gates, approximately 200 chips would be necessary to complete a processor module. The large number of devices required, and the interconnections necessary, tend to rule out the use of this logic family in the ARMMS computer.

Aside from the interconnection problems, the low power Schottky T²L element would be an ideal device for systems use. It does not seem likely, however, that arrays in excess of 250 gates will become available in the near future. Nor does it appear likely that processing yields will allow anything other than discretionary wiring techniques for reaching this level of complexity.

CMOS

The advent of CMOS digital elements has given system designs a new feel which solves many of the problems of bipolar hardware. CMOS has the

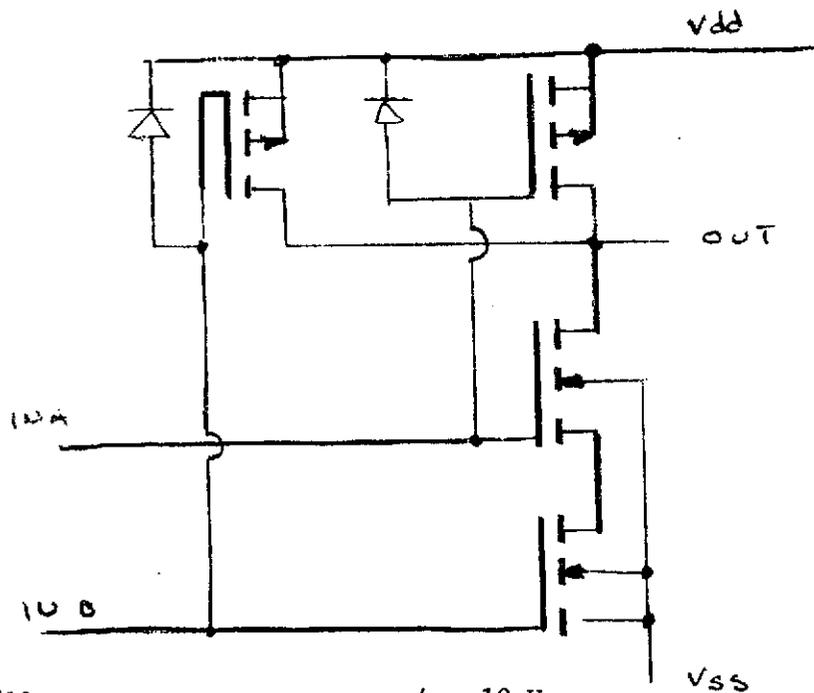
advantage of extremely low power, high noise immunity, high fan out and wide tolerance to power supply voltage. Table IV-A lists CMOS's most desirable features and characteristics. Figure 3A presents the characteristics of a typical CMOS gate.

TABLE IV-A. CMOS ADVANTAGES AND FEATURES

1. The lowest power dissipation of any logic form, thus lowering cost.
 2. Excellent noise immunity that increases with increased supply voltage (.45 V_{dd}).
 3. Operation over very wide supply voltage range (1.2 volts to 18 volts).
 4. Has packing density greater than bipolar technology, resulting in lower cost MIS and LSI functions.
 5. Has lower output impedance than PMOS, thus simplifying interfacing with saturated bipolar logic.
 6. Operates over wide temperature extremes with minimum performance degradation.
 7. Very high impedance results in the highest fanout of any logic form.
 8. Logic swing is between power supply and ground.
 9. Propagation delay is faster than PMOS, and speeds will soon approach those of TTL.
-

Present CMOS elements, with light capacitive loading, are substantially faster than PMOS elements are comparable with slower speed T²L elements. See Figure 4.

One of the most desirable characteristics of CMOS is its low power dissipation. Under quiescent conditions either the p channel or n channel device is off; consequently, the device is dissipating virtually no power. Only during the transition between states does the device dissipate power. Quiescent power



Vdd	4 - 18 V		
Fanout	750		
Propagation delay	0-1	45 ns	$C_2 = 20$ pf
	1-0	30 ns	
Dissipated power	quiescent	.01 μ w	
	dynamic	.4 mw/MHZ	
Noise immunity		.45 Vdd	

TYPICAL CMOS GATE

FIGURE 3A

FIGURE 4(a) - TYPICAL GATE POWER DISSIPATION CHARACTERISTICS

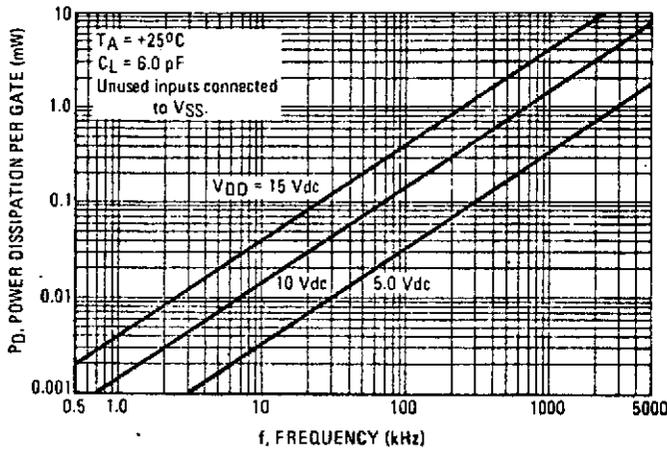
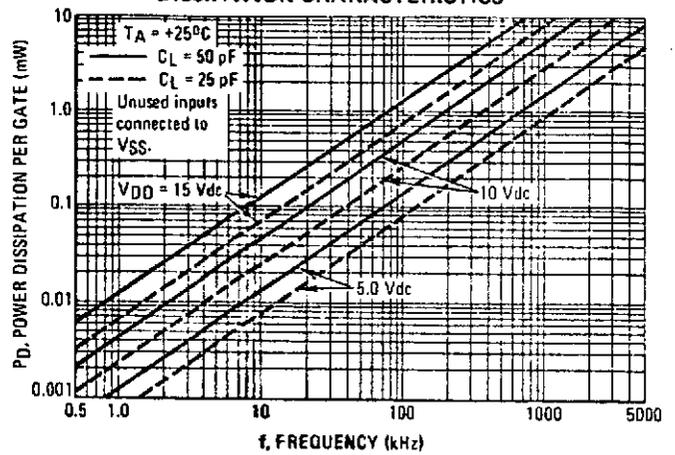


FIGURE 4(b) - TYPICAL GATE POWER DISSIPATION CHARACTERISTICS



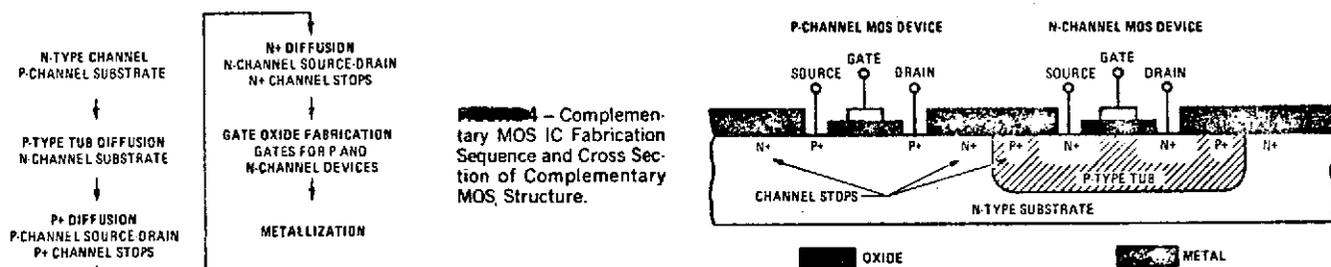
CMOS GATE DISSIPATION VS. FREQUENCY
FOR VARIOUS
VARYING LOAD CAPACITANCES

FIGURE 4

dissipation is typically $0.01 \mu\text{w}$ per gate; dynamic power dissipation is $.4 \text{ mw/MHz}$. With a lightly loaded (6 pf) line, a CMOS gate will dissipate approximately 2 mw at 5 megahertz.

The ability to operate CMOS from a single, relatively wide tolerance supply bus significantly eases system power supply design requirements. Most CMOS logic is fully capable of working from a supply voltage from as low as 4 volts, to as high as 18 volts. Noise immunity of CMOS elements is correspondingly high. Noise immunity is typically $.45 V_{DD}$ and increases with increasing supply voltage.

Another significant advantage of CMOS is simplicity of fabrication. See Figure 5. CMOS requires three major diffusion steps compared to five for bipolar devices. Device geometries for CMOS are significantly smaller than for bipolar elements, and linear resistors are not used. Consequently, a CMOS gate may be as much as a factor of eight smaller than its bipolar counterpart. This, combined with simpler fabrication processes, will allow high complexity chips with moderate yields.



CMOS DIFFUSION PROCESSES & TRANSISTOR STRUCTURES

FIGURE 5

CMOS elements are now fully capable of operating over the entire military temperature (-55 to +125°C) with only minimum variations in device performance.

Because of its relatively low output impedance and high input impedance, CMOS has the largest fanout capability of any logic form. Fanouts of greater than 50 are readily achieved. Interface with bipolar logic elements is also relatively simple. CMOS will interface with T²L directly, and open collector T²L will directly interface with CMOS. A pull up resistor is normally required to interface T²L elements to CMOS inputs.

By eliminating the parasitic capacitance in CMOS junctions and substrates, speeds near 100 megahertz can be expected. Presently, ION implantation techniques, silicon gate CMOS, silicon on sapphire and silicon on spinnel techniques are being developed in the industry. These techniques will push CMOS speed substantially above that needed for the ARMMS processing equipment.

Conclusions

If a processor module was to be constructed today, and if the processor had to operate at speeds in excess of 5 MHz, low power Schottky T²L would be selected as the best logic choice. CMOS would have to be rejected because of its somewhat lower speed. This would be the only reason for its rejection.

Since the design of this computer is being projected into the 1974-1976 time frame, it is the author's belief that CMOS processing will have proceeded to the point that its speed characteristics will equal or surpass that of low power Schottky T²L. With the higher speed, lower power, greater fanout, ease of T²L interfacing and greater device complexity, the CMOS logic element should be chosen as the basic logic element in the ARMMS computer.

II. ARMMS Data Transmission Line Study

In the Phase I report hardware speed and performance characteristics of data base transmission systems were discussed. It was noted that for bus speeds below 50 MHz current source drivers and differential line receivers are the optimum bus interface elements. Presently available bipolar integrated circuits can manipulate data at rates up to 15 MHz at reasonable power levels. A transmission line with uniform characteristic impedance and short stubs for interconnecting the modules was recommended. During Phase II data buses were studied further with emphasis placed on loading considerations, detection theory, module interconnection methods, and reliability. The following ground rules were adapted:

1. Maximum inter-module cable length \leq 6 feet.
2. To reduce pin counts single ended rather than differential current source drivers and receivers will be used.
3. Maximum stub length from module to bus \leq 3 inches.
4. Each connector pin has a capacitance of about 7 pf.
5. Bus will be contoured microstrip line (propagation delay equals 2.25 ns/ft).
6. The number of modules to be connected will not exceed 48.
7. Transmission power shall be minimized as much as possible without degrading the data transmission quality.

8. A synchronous clock system with a period greater than worst case delays in bus and module interfaces is required to allow lock-step operations in duplex and TMR modes.

Since data is bussed to many modules, it is important that a failed module does not short the signal bus. If a module fails open, the module is not available for use by the rest of the system, but this does not result in a complete system failure. There are many approaches to signal bus isolation; most fall into either of two classes, parallel bus redundancy or use of series isolation elements.

Using parallel busses would multiply the number of connector pins and the number of drivers and receivers by the number of parallel busses. Series redundancy may be accomplished with series elements in each bus line or with a single series element in series with a group of elements.

To isolate receivers, resistor isolation may be sufficient. For most driver schemes a switch is necessary. For current drive transmission it is necessary only to provide a switch in series with the high state supply for the drivers to isolate a failed or unused module from the signal bus, as shown in Figure 6. This is because current mode drivers have a low impedance path from the signal bus to the +5.0V supply, and no low impedance path from the signal bus to ground. A short circuit failure in both driver output transistors Q1 and Q2, shown in Figure 6, is necessary to disable the signal bus. If the two receiver input transistors short circuit fail, the bus is not disabled because of the isolation resistance. This would only present a load, and not a short, to the signal bus. If a transistor between the signal bus and the +5.0V supply fails shorted in either a receiver or a driver, the signal bus would still not be disabled unless the dc-to-dc converter output switch fails shorted also, in keeping with the "no one component failure disables the signal bus" concept.

Resistor R at the driver input in Figure 6 must be provided in the circuit design to keep Q1 leakage current from forward biasing Q1. Since the dc-to-dc converter output is at ground potential when turned off, Q1 could cause Q2 to conduct from the signal bus to the converter in the event that the +5.0V switch is shorted.

Each transmission line in the system interfaces with several modules through connector pins, each with a capacity of five to seven picofarads. The effect of these discontinuities on the performance of the data transmission is a function of the transmitting frequency, the average length between these capacitive loads, the standard deviation of these separations, the characteristic impedance of the transmission line, and the detection scheme and the signal to noise ratio. One of the objectives of this study is to minimize the transmitting power without degrading the data transmission. This implies that pulse fidelity should be maintained throughout the transmission line.

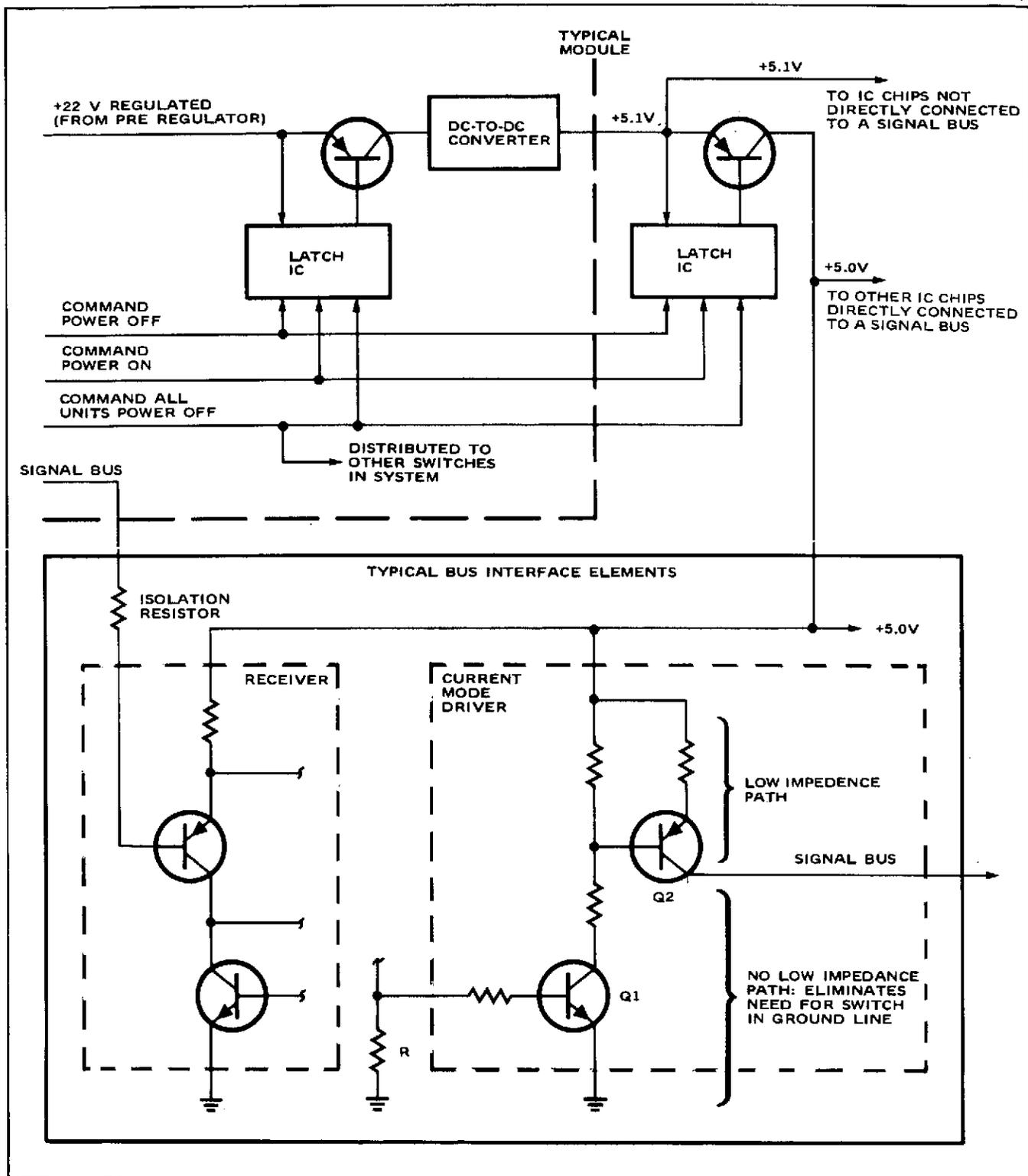


Figure 6. Dual Isolation System. Two transistors must short circuit fail before the signal bus is disabled. Use of receiver input resistors and current mode type drivers eliminates the need for switching the power ground to the modules.

Detection theory is applied to the problem to define the transmitted levels and the receiver configuration for a low probability of error. In this problem the receiver in each module is synchronous or coherent to the transmitter via a common clock. A pulse $S(t)$ of duration T seconds may be transmitted for a logical one, and no pulse (no signal) transmitted over the duration of T seconds for a logical zero or for no data transmission. The transmitted signal is masked by noise over the data link channel. In ARMMS, the noise will mainly be ground noise between modules since each receiver is referenced to a ground not identical to the transmitter ground. Other noise sources will be crosstalk or coupling of power lines and other transmission lines to the line of interest.

One of the design requirements is to reduce the pin count at the interface of the modules and the transmission line. This requirement implies single ended instead of differential transmission. The greatest advantage of this interconnection technique is the pin count reduction derived by the single ended transmission and reception. However, this approach presents several weaknesses. First, the transmitting current is returned to the transmitting module via the signal ground path. This in turn will develop a potential difference between the transmitting module and the receiving modules. Since the receiving modules have threshold detectors referenced to a ground more noisy than the transmitter ground, the received voltage will be interpreted by the receiver as signal plus noise.

Figure 7 plots transmitter power vs bit rate for a single bit bus as a function of the number of modules on the line. These numbers will be multiplied by up to 104 times to obtain peak transmission power for 8 thirteen bit buses. Figure 8 and Table V illustrate and list the sources of worst case transmission delays. Voter and control logic implemented in the modules would add additional delays.

The baseline bus design characteristics for configuration C are as follows:

1. 100 nsec clock rate (10 MHz)
2. A maximum of 25 memory modules, 7 processor modules, 4 I/O modules, 1 BOSS module (37 total)
3. Line characteristic impedance = 50 ohms
4. Line driver current requirement = 100 ma
5. Signal voltage amplitude = 2 volts
6. Average driver power dissipation (50% duty cycle) = 250 mw leading to a peak bus transmission power requirement of 26 watts (3.25 watts per active module).

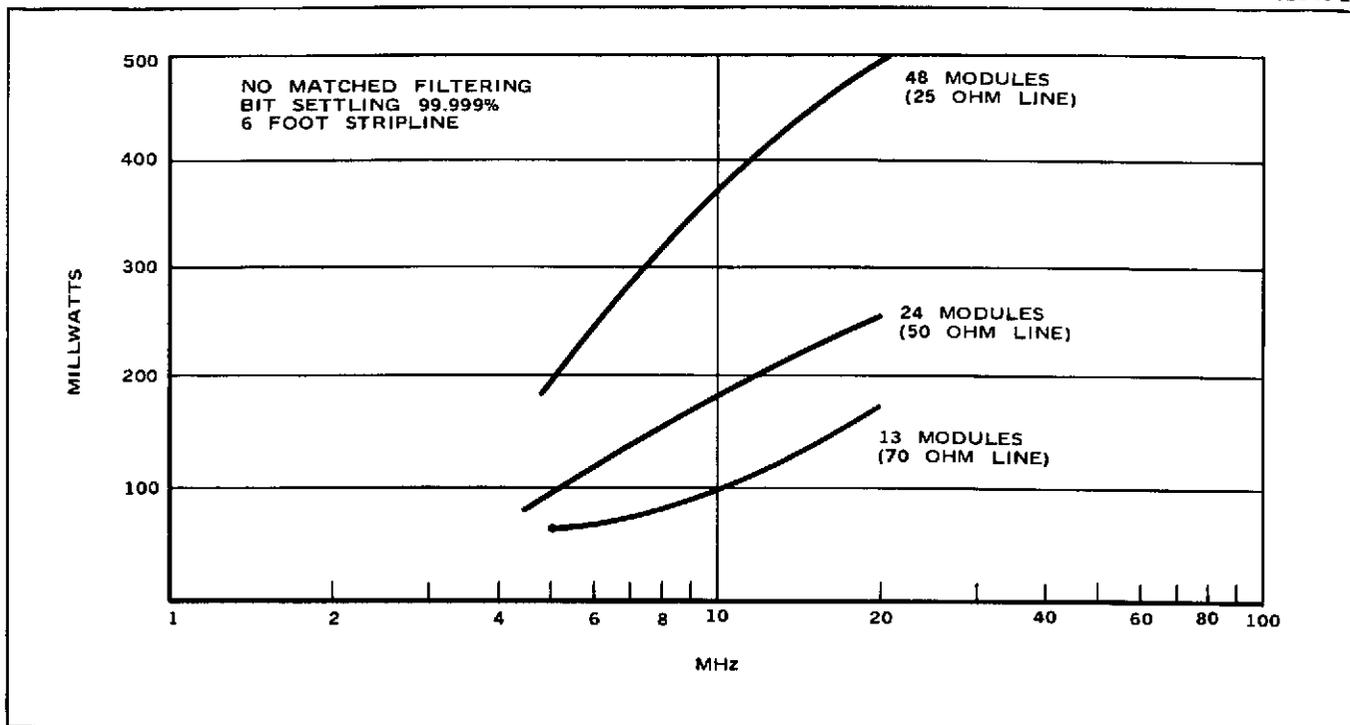


Figure 7. Transmitter Power versus Bit Rate

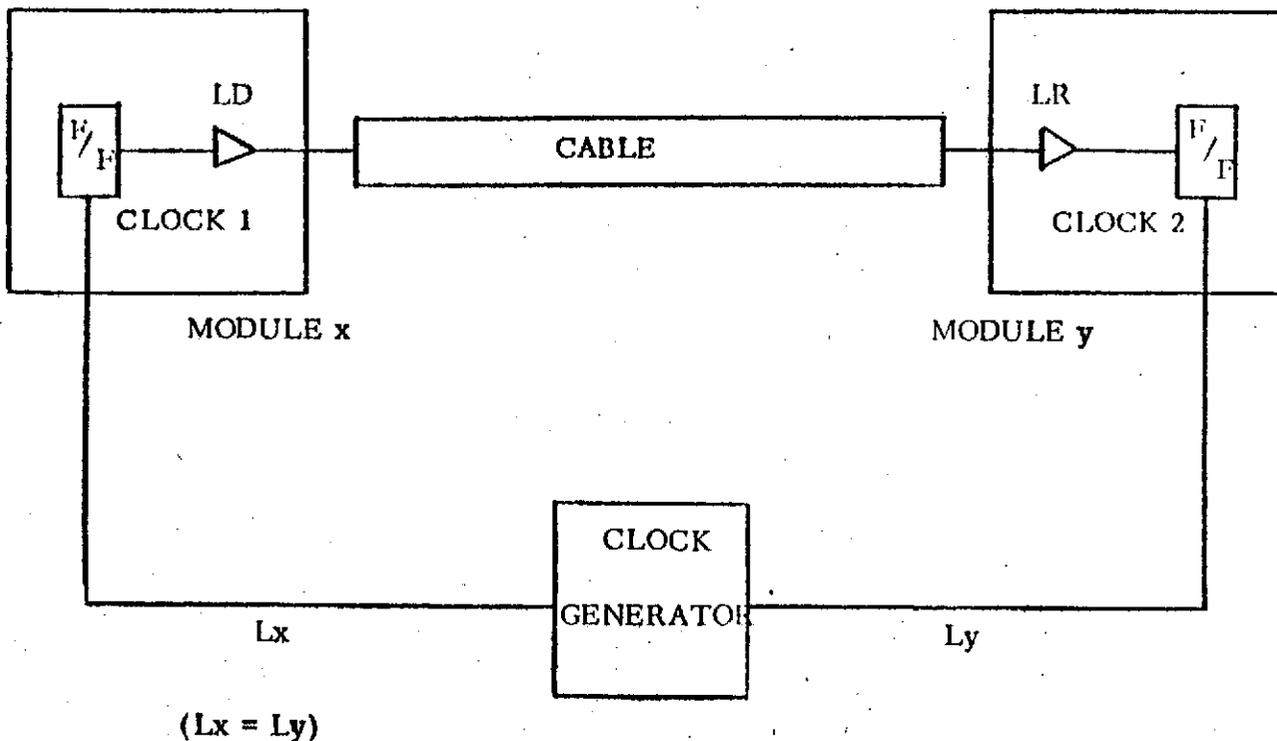


Figure 8. Typical Intermodule Communication

TABLE V. SUM OF WORST CASE DELAYS

Clock Skew	10 NS
(Difference between clock 1 and 2)	
F/F Resolution	7 NS
Driver Delay	20 NS
Cable Delay	13 NS
(Assumes 6 feet)	
Receiver Delay	15 NS
F/F Set-Up Time	8 NS
Total Worst Case Times	73 NS

Development of a current source line driver with the required characteristics may prove to be a problem area. Possible approaches to reducing bus power include reducing speed, reducing the number of modules in the system by using matched filters as receivers instead of threshold detectors. This latter alternative would require more effective bit synchronization and would reduce the driver power by a factor of approximately four. A typical current source driver is shown in Figure 9. A matched filter receiver block diagram is shown in Figure 10.

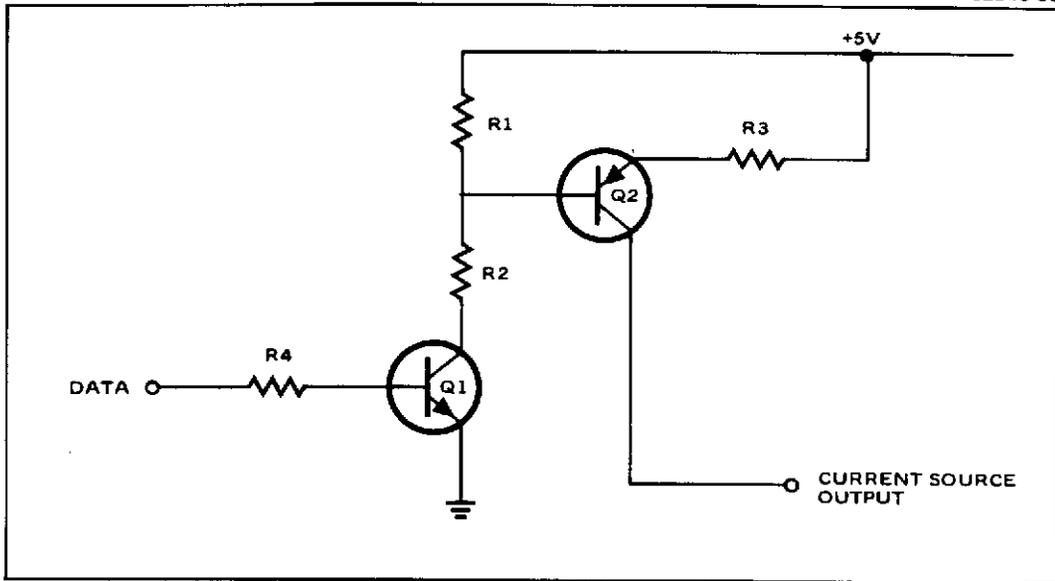


Figure 9. Design of a Current Source Line Driver. Current Output is Determined By Selection of R1, R2 and R3

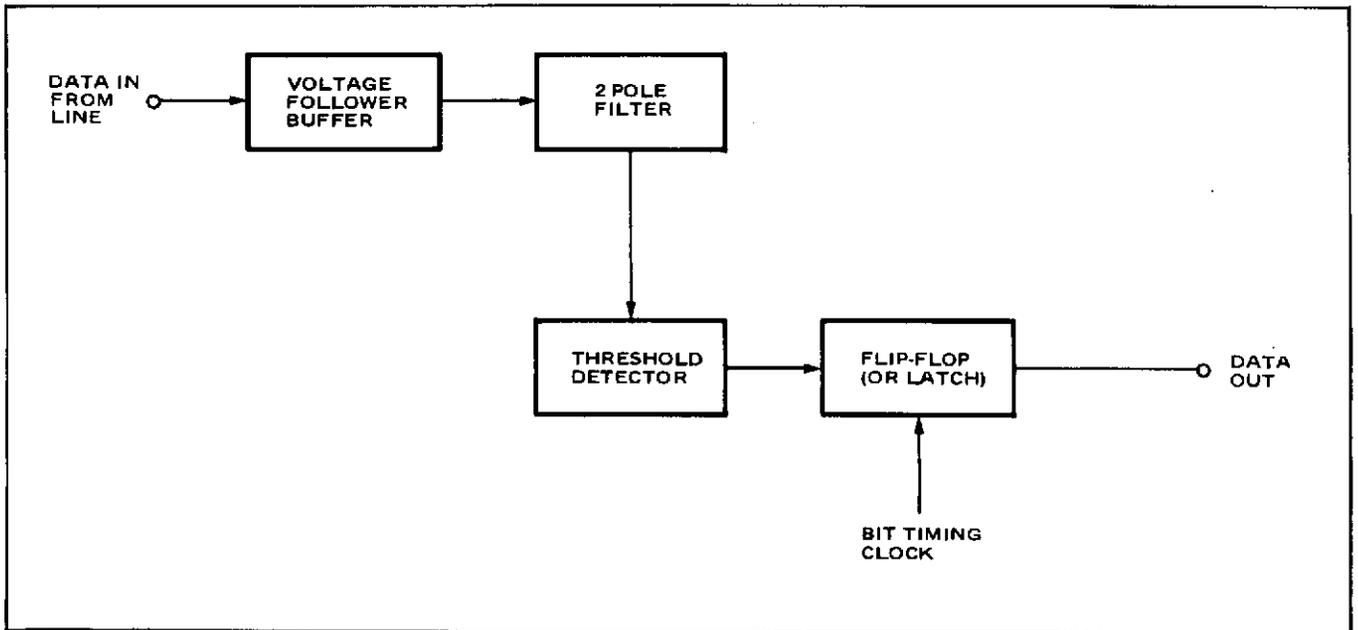


Figure 10. Matched Receiver Block Diagram.

III. ARMMS POWER DISTRIBUTION STUDY TASK REPORT

The purpose of this study is to accomplish the following:

1. Identify basic alternatives for supplying power to the computer modules.
2. Discuss the advantages and disadvantages of each alternative.
3. Select a baseline power distribution implementation.
4. Generate detailed circuits illustrating how the baseline design can be implemented.
5. Provide part count, weight, and power numbers for the baseline design.

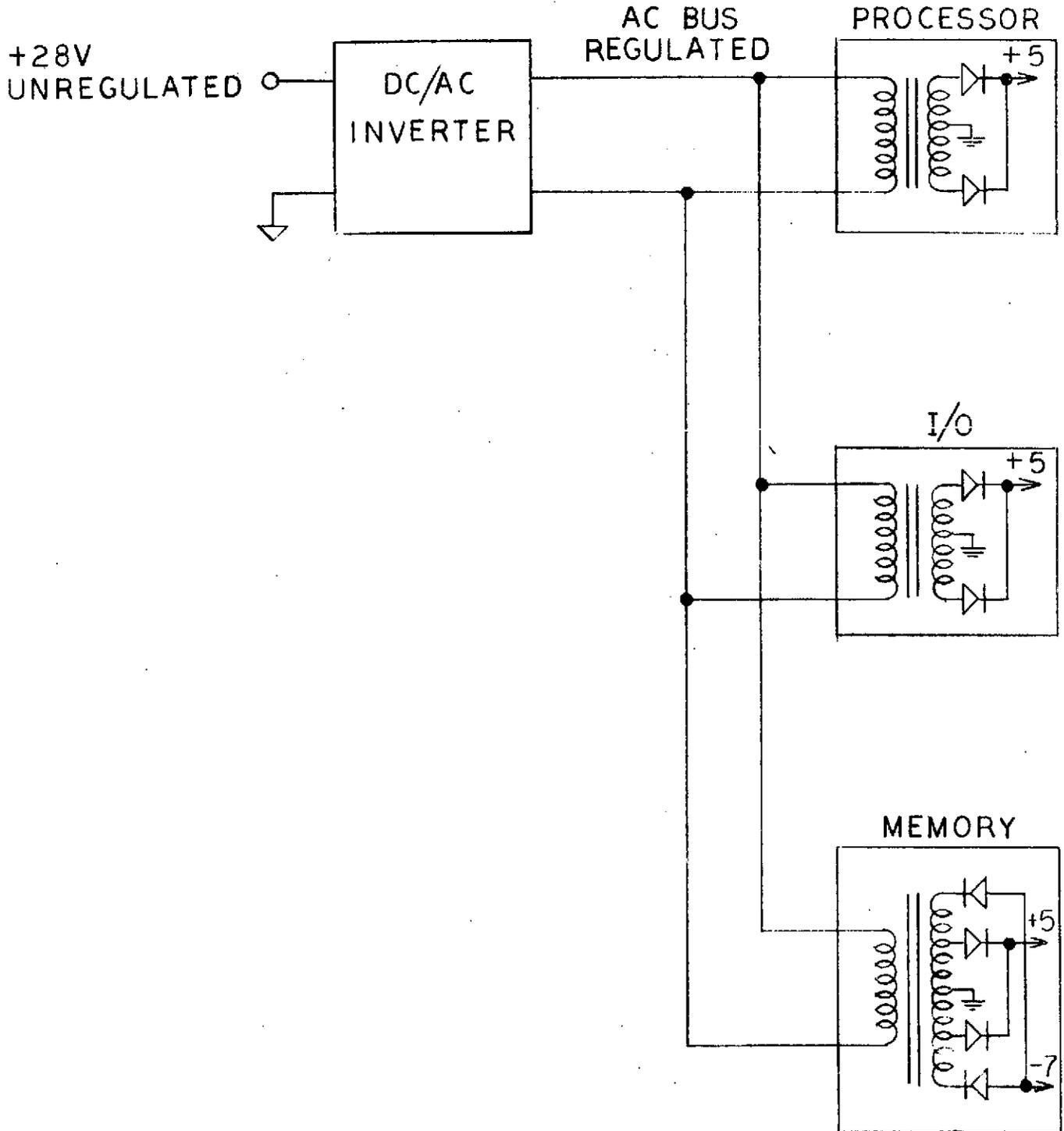
The following ground rules shall be used for this study:

1. Power supply reliability should not limit system reliability.
2. Power supply modularization and standardization shall be used.
3. No module class may depend on one power supply.
4. Reliability modeling will be performed by Division 11.

Primary Power Distribution

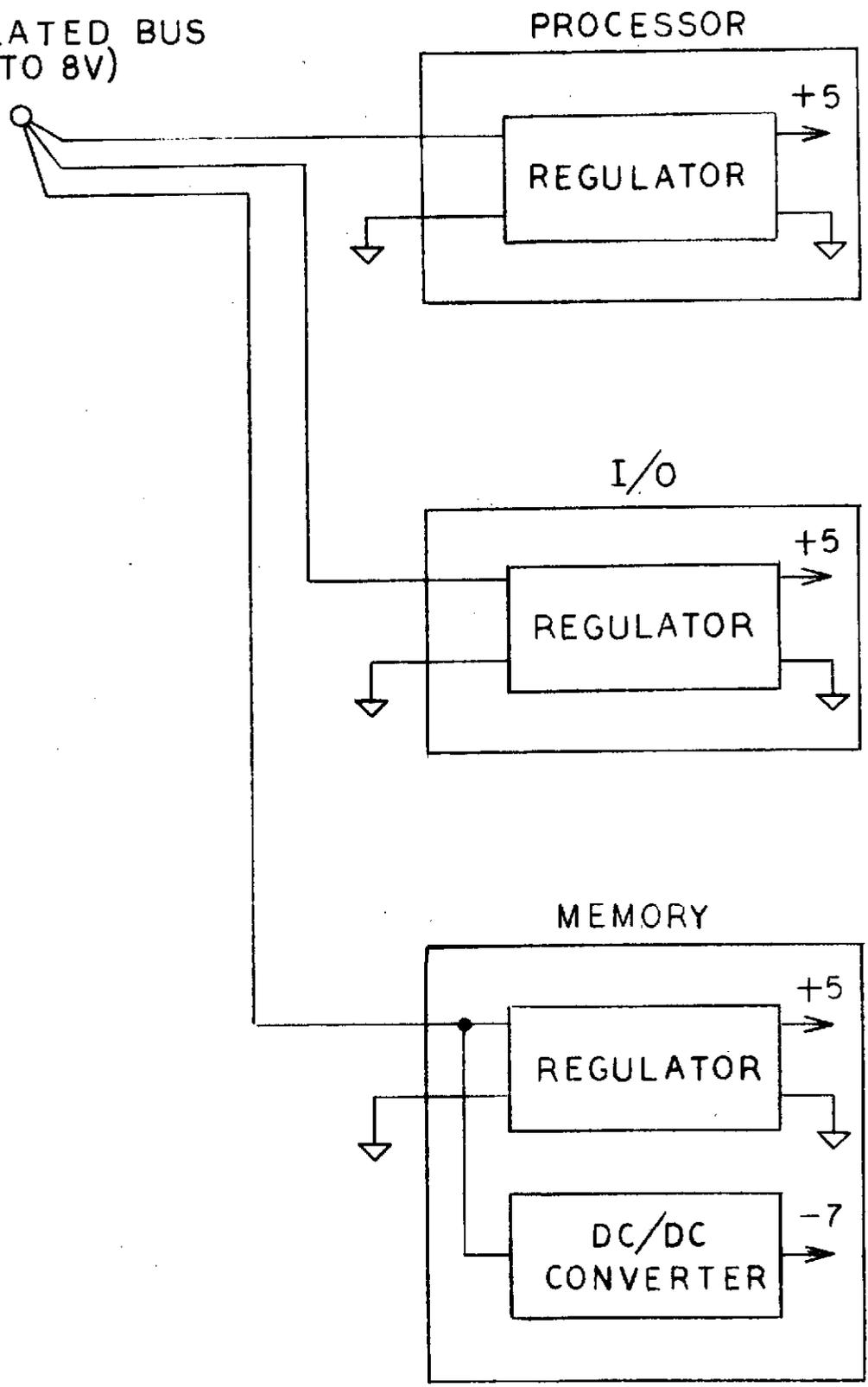
Before investigating secondary power distribution it is worthwhile to briefly discuss the various ways to distribute primary power. Three bus systems were investigated: a) Regulated AC Bus, b) Low Voltage DC Bus, and c) Conventional DC Bus. The advantages and disadvantages of each alternative are summarized as follows:

BUS ALTERNATIVE A REGULATED AC BUS



BUS ALTERNATIVE B LOW VOLTAGE DC BUS

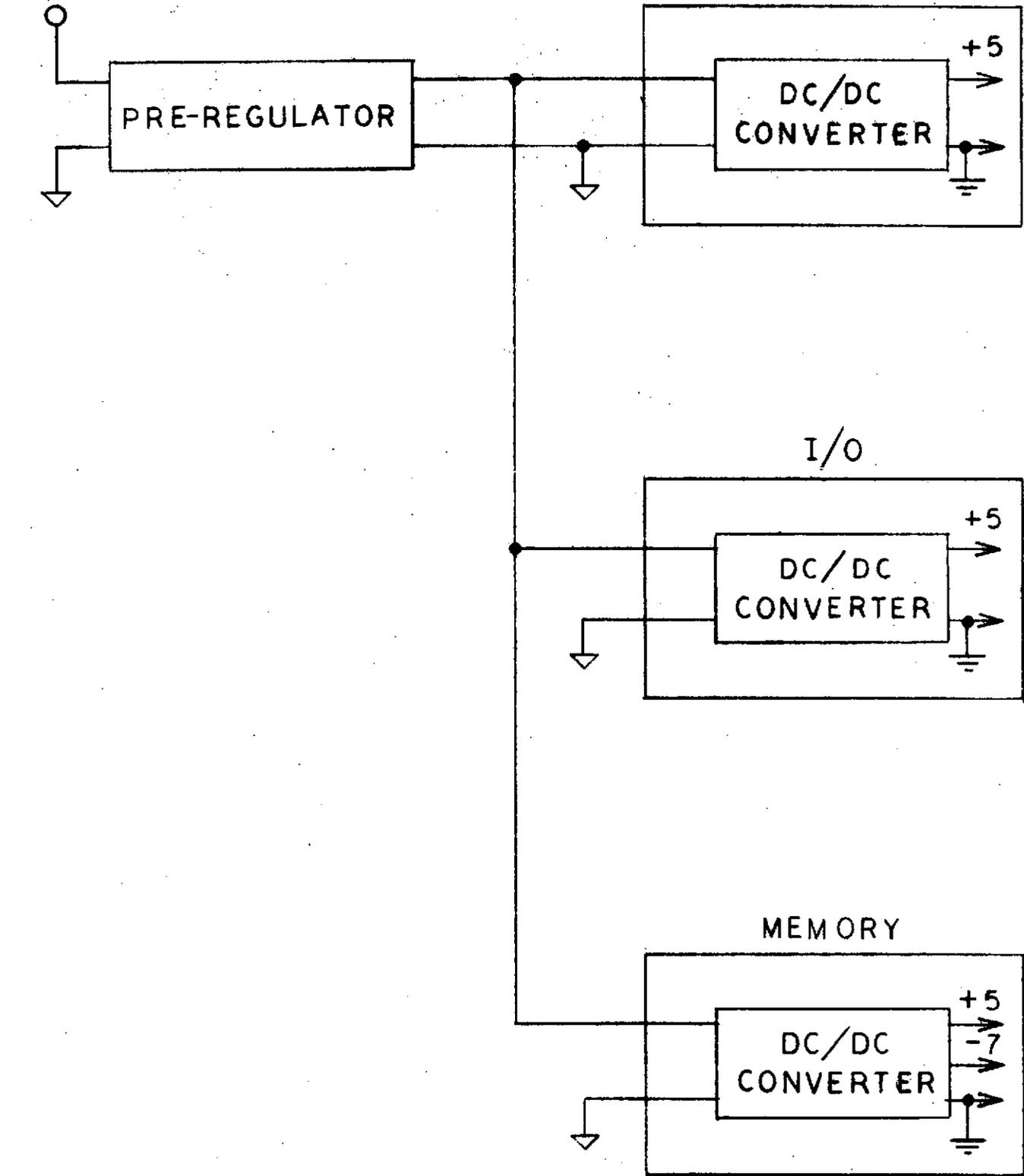
UNREGULATED BUS
(6V TO 8V)



BUS ALTERNATIVE C CONVENTIONAL DC BUS

+28 UNREGULATED

PROCESSOR



Alternative A – Regulated AC Bus

Advantages:

1. The module power supplies can be kept very simple.
2. Separate ground isolation for each module is provided.

Disadvantages:

1. The DC/AC inverter would involve some difficult and time consuming design effort.
2. To maximize efficiency, the AC bus would have to be a medium frequency (about 4 kHz) square wave. This would result in a very noisy bus.
3. AC power is more difficult to switch than DC power. Therefore the power switching would have to be done past the transformers and rectifiers. This would lead to a slight loss of efficiency and reliability

Alternative B – Low Voltage DC Bus

Advantages:

1. Virtually no DC/AC inverters or DC/DC converters would be required.
2. A minimum part count power distribution system would be achieved.

Disadvantages:

1. No module ground isolation.
2. Poor efficiency. The efficiency gains made by eliminating DC/DC converters would be more than offset by large line losses due to large line currents and by the drops across the linear regulators.
3. Other equipment on the spacecraft might have to be modified to use a low voltage bus.

Alternative C – Conventional DC Bus (+28V nominal)

Advantages:

1. A +28V nominal DC bus is an industry standard. Therefore a great deal of experience has been accumulated on how to design equipment with a +28V DC bus.
2. Module ground isolation.
3. Clean (non noisy) bus.
4. Power switching is easy to implement.

Disadvantages:

1. Large part count.

A conventional DC bus will be assumed as a baseline, since it has a high advantages to disadvantages ratio.

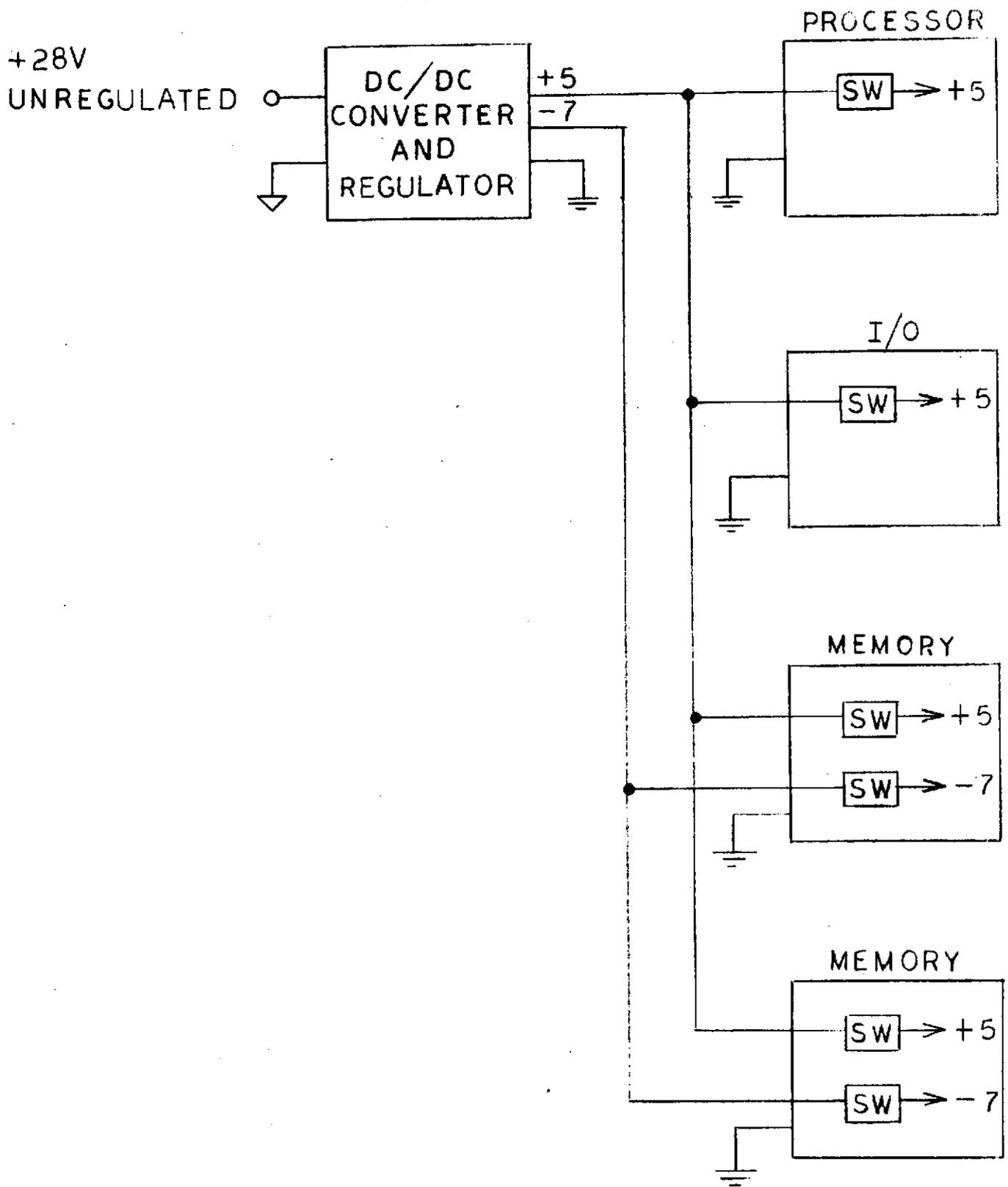
Secondary Power Distribution

The following assumptions will be made before analyzing various power distribution alternatives.

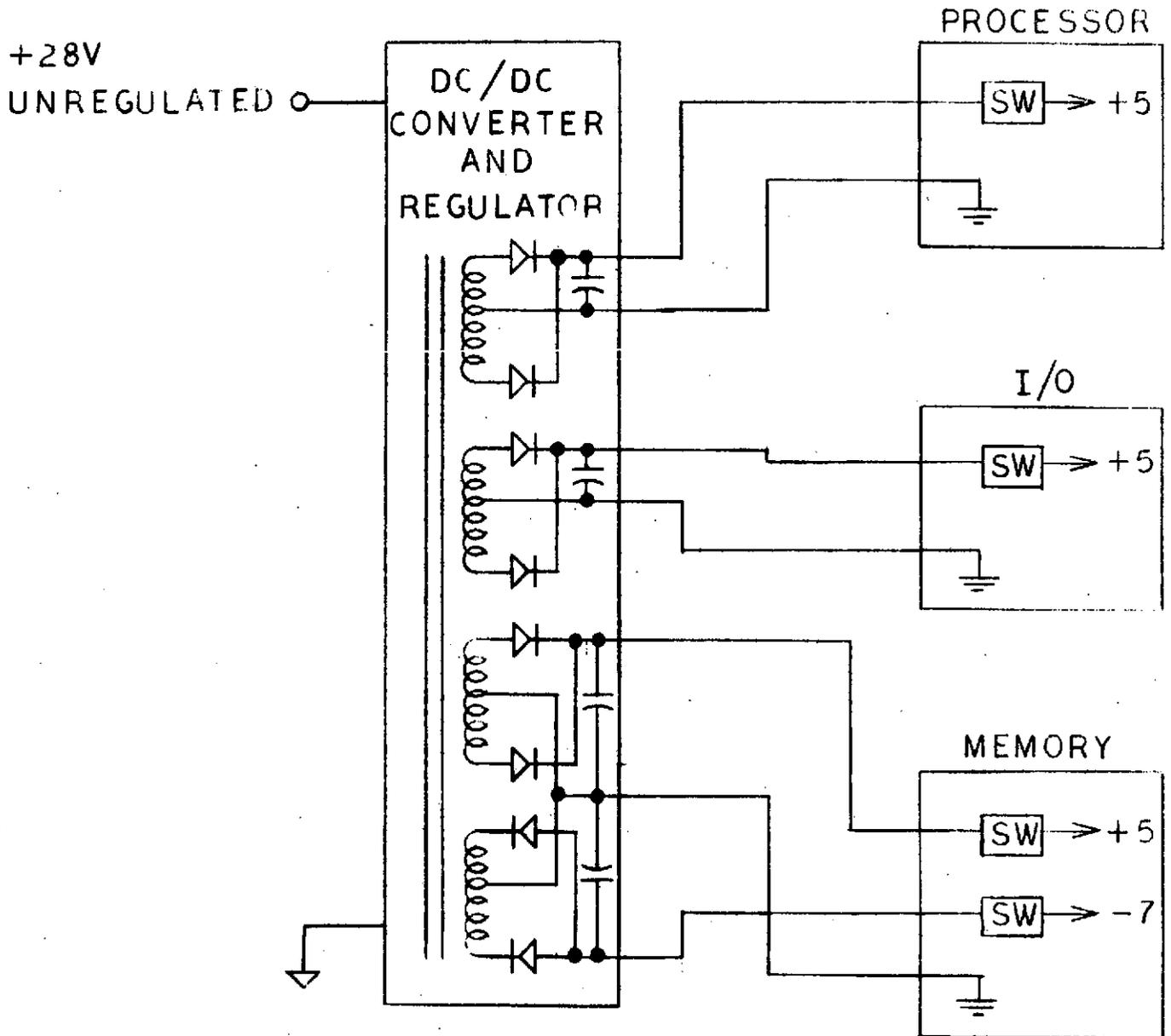
1. The BOSS modules will be ON all the time.
2. The other ARMMS modules will be pulse commandable. The ON/OFF command pulses will be generated by the BOSS modules. Therefore all the non-BOSS modules will contain power switching and power memory.
3. No sub-module partitioning will be used.
4. DC ground isolation in each module will be provided.

Several secondary power distribution alternatives are illustrated in the following figures.

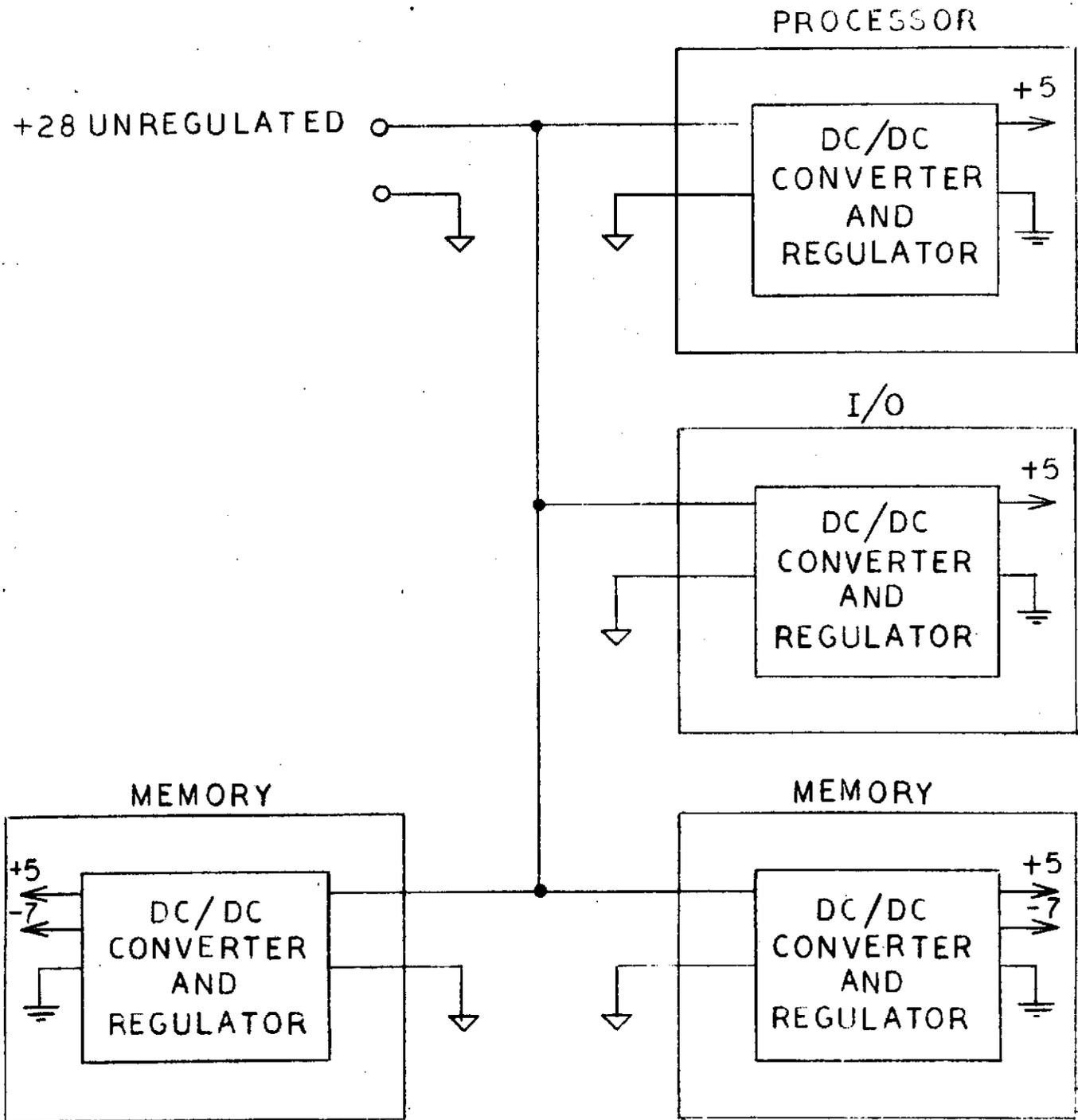
ALTERNATIVE A CENTRALIZED POWER DISTRIBUTION, NO BACKUP



ALTERNATIVE C CENTRALIZED POWER DISTRIBUTION, WITH ISOLATED OUTPUTS

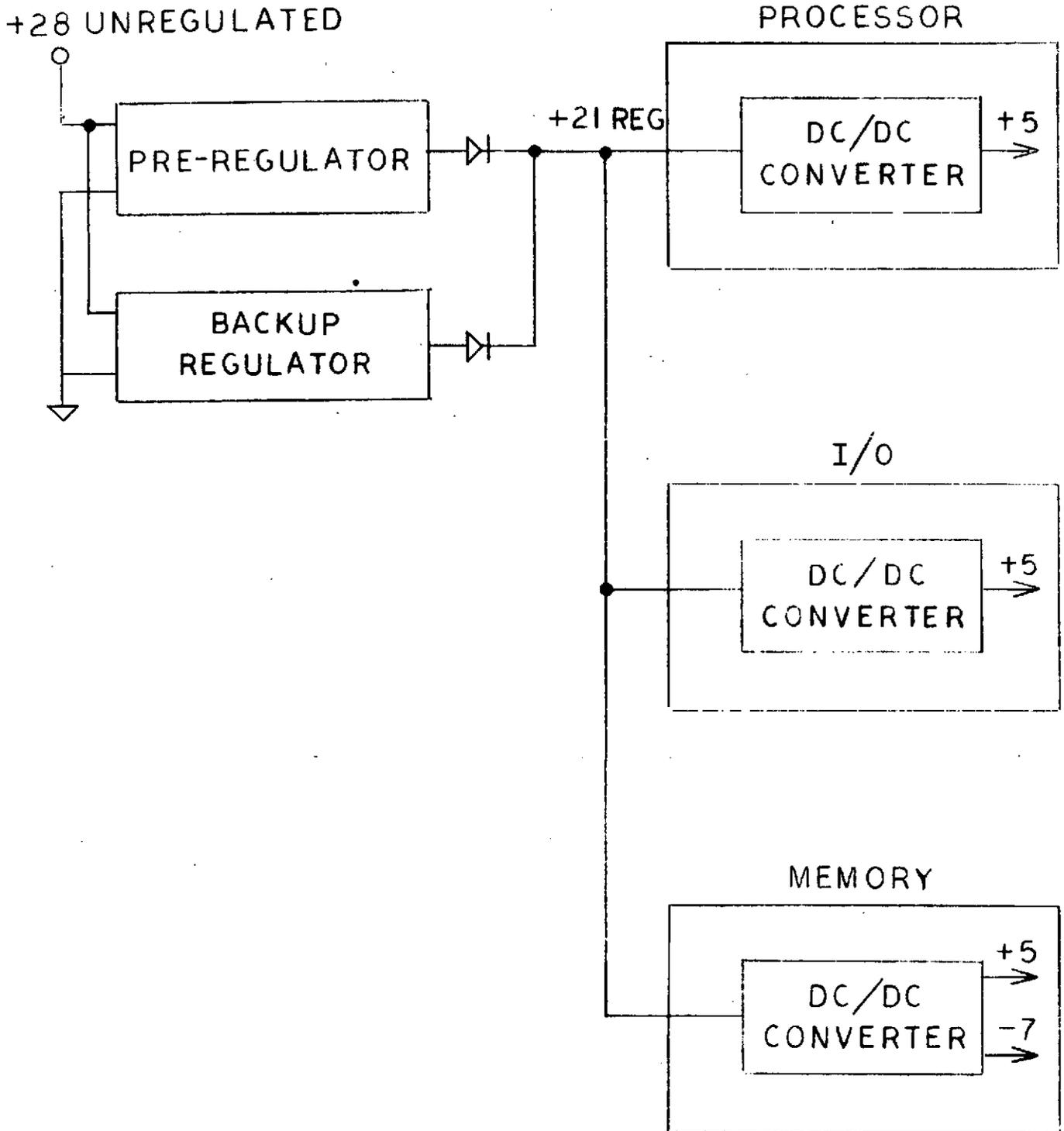


ALTERNATIVE D DECENTRALIZED POWER DISTRIBUTION, NO PRE-REGULATION

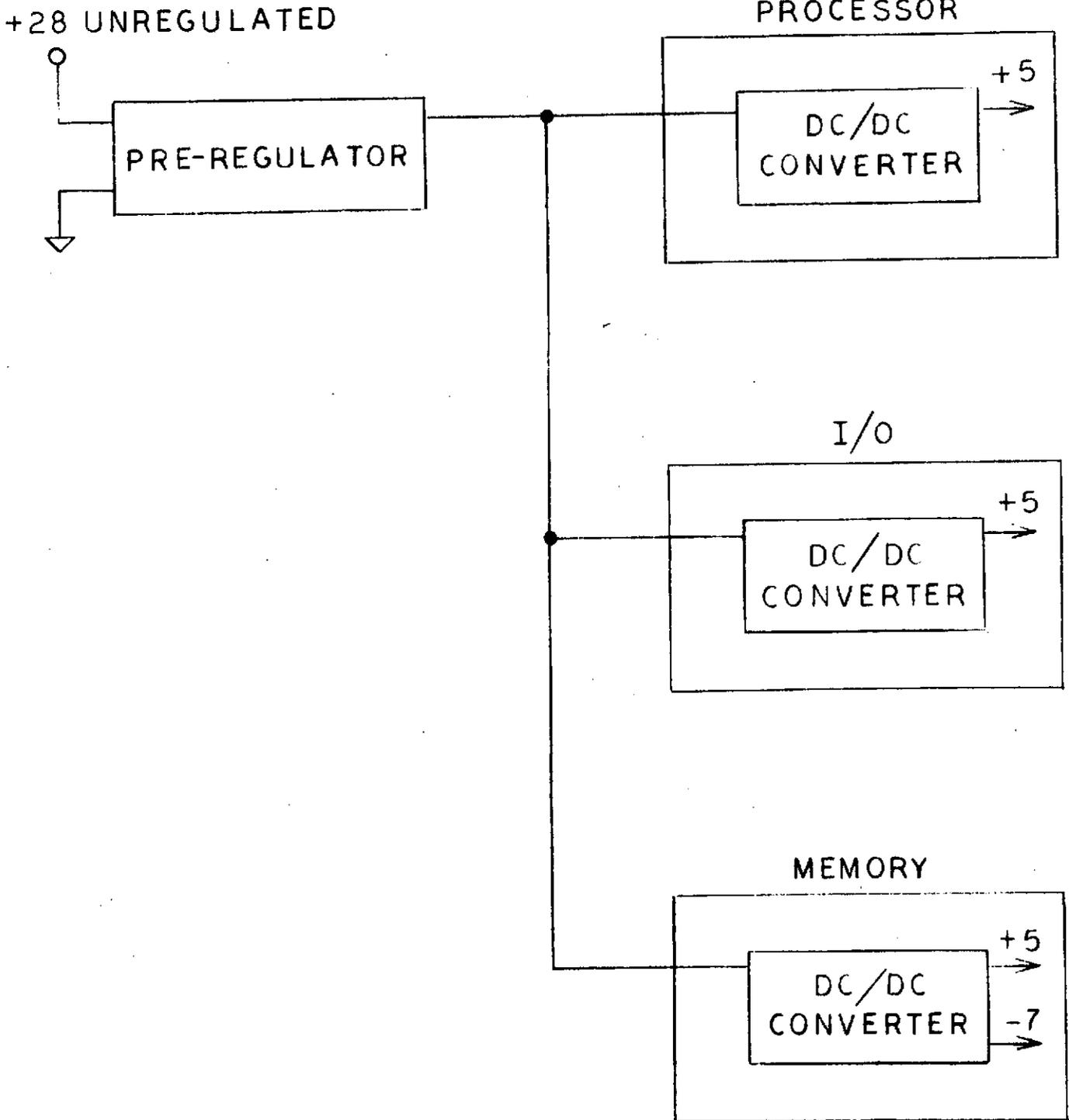


ALTERNATIVE E

DECENTRALIZED POWER DISTRIBUTION, WITH PRE REGULATOR AND REDUNDANCY



ALTERNATIVE F
DECENTRALIZED POWER DISTRIBUTION,
WITH PRE-REGULATOR, NO REDUNDANCY



CA

The various secondary power distribution alternatives can be grouped into four basic groups:

1. Fully centralized supply
2. Partially centralized supplies
3. Fully decentralized supplies
4. Partially decentralized supplies

Fully Centralized Supply

In a fully centralized power distribution system, a single power supply (possibly with one backup supply) would supply power to all the ARMMS modules. The ARMMS modules (with the exception of the BOSS) would only contain power switches and power memory. The BOSS modules would receive unswitched power. Alternatives A, B, and C are examples of centralized power systems.

The advantage of a centralized power supply is minimum parts count. But a centralized system has many disadvantages. A single component failure could cause the entire system to fail. This eliminates it from consideration since it violates one of the study ground rules.

A centralized system is very inflexible since it must be designed for maximum load conditions. Under minimum load conditions the power supply would be oversized.

Partially Centralized Supply

In a partially centralized system several central power supplies are used, one for each set of modules. Under these conditions if a power supply fails it does not cause the failure of an entire ARMMS system. A partially centralized system also provides greater flexibility at the cost of a greater part count. Alternatives A, B, and C are examples of partially centralized systems if these alternatives are repeated several times.

One remaining problem with a partially centralized supply system is providing DC ground isolation on the module level. One way of doing this is shown as alternative C. But alternative C introduces regulation problems. There does not appear to be any way to achieve ground isolation and good regulation with a centralized or partially centralized system.

Fully Decentralized Supplies

In a fully decentralized power distribution system, each module contains its own complete power supply. An example of a fully decentralized power distribution system is shown as alternative D.

A decentralized system provides extremely good flexibility, good efficiency, good regulation, and DC ground isolation. The main disadvantage of a decentralized system is the high part count that is required.

Partially Decentralized Supplies

In a partially decentralized power distribution system, each module contains its own power supply. But instead of each supply operating from an unregulated bus, each supply would operate from a regulated bus supplied by a common pre-regulator. Each pre-regulator would supply power to several modules. The advantage to doing it this way is that now it is possible to remove the voltage regulating circuitry from each module supply thereby saving a significant number of parts. Alternatives E and F are examples of partially decentralized supplies.

In a partially decentralized system a small amount of flexibility is sacrificed to reduce power supply part count.

Conclusion

A partially decentralized system will be used as a baseline. Specifically alternative F will be used as a baseline. Alternative F was chosen for the following reasons:

1. Low part count

The part count of alternative F will not be the lowest possible, but it will be lower than alternatives D or E.

2. Ground isolation

DC ground isolation for every module can easily be provided.

3. Good regulation

Since most of the power supply will be inside the module, good voltage regulation can be provided.

4. Good configuration flexibility

Alternate F provides greater configuration flexibility than alternatives A, B and C but less than alternatives E or D.

If the total number of modules per configuration are added up, we get the following summary:

<u>Configuration</u>	<u>No. of Modules</u>
Simplex	5
Duplex	10
TMR	15
TMR & Spare	20
MAX	30

The above table shows that the different configurations are separated by steps of five modules. By using one pre-regulator for five modules, it is possible to insure that each pre-regulator will be operating near its designed output capability, thereby avoiding one of the weaknesses of a partially decentralized system.

5. Good bus flexibility

Use of a pre-regulator makes it easier to adapt an ARMMS system to changing bus characteristics. For example, if an AC bus was used on a future program, it would only be necessary to modify the pre-regulators. On the other hand if alternatives D or E were used all the modules would have to be redesigned.

6. Good reliability

Packaging most of the power supply circuitry inside the module increases reliability, because failure of one supply will only cause one module to fail.

7. Good thermal characteristics

By packaging the regulation circuitry outside the modules, the losses associated with regulation are removed from the modules. This will help to keep the heat rise within the modules down to a reasonable level.

8. Good Output Voltage Flexibility

Since power is switched to the primary side of a DC/DC converter, it is possible to add new secondary voltages without incurring much of a penalty. In a centralized or partially centralized system secondary voltages are switched. Therefore the addition of extra secondary voltages are very costly in a centralized system, but inexpensive in a partially decentralized system.

Power Distribution for the Various ARMMS Configurations

Assuming alternate F as a baseline, the power distribution for the various configurations would appear as shown in Figures 11 to 15. The pre-regulators will not be cross-strapped. Cross-strapping helps to protect against "open" failures but can cause reliability degradation against "short" failures. For example, if two pre-regulators are diode "ORed" together, a single short at the output would cause both pre-regulators to fail. By not cross-strapping pre-regulators, the effect of "short" failures can be isolated. In addition, not cross-strapping pre-regulators leaves the way open to the effective use of multiple primary busses for even greater reliability. The greatest reliability can be achieved if each pre-regulator has its own primary bus.

The configurations in Figures 1 to 5 are "first cut" configurations. It may be possible to interconnect the pre-regulators and modules in a more optimum manner.

Detailed Design

The detailed designs for the module power supplies and for the pre-regulator are shown in Figures 6 to 8.

Design Discussion

Figure 16 shows the detailed design for the processor, I/O, and memory power supplies. The Figure 16 power supply is commandable. A +15 volt command pulse will be used to turn on IC₁. IC₁ will then turn on Q₁ via Q₂ and Q₃. Q₁ will then feed back power to IC₁ thereby keeping it on. Q₁ is a current limit switch. Should a module fail in a "short" mode, Q₁ will go into current limiting. If the collector voltage of Q₁ drops down to a critical threshold level, it will starve power to IC₁ and therefore the power supply will unlatch. This means that if a module fails in a "short" mode, Q₁ will automatically turn off thereby disconnecting the failed module from the regulated bus. If the module fails in an "open" mode, the BOSS can disconnect the module from the regulated bus via IC₁ and Q₁.

FIGURE 11
SIMPLEX CONFIGURATION

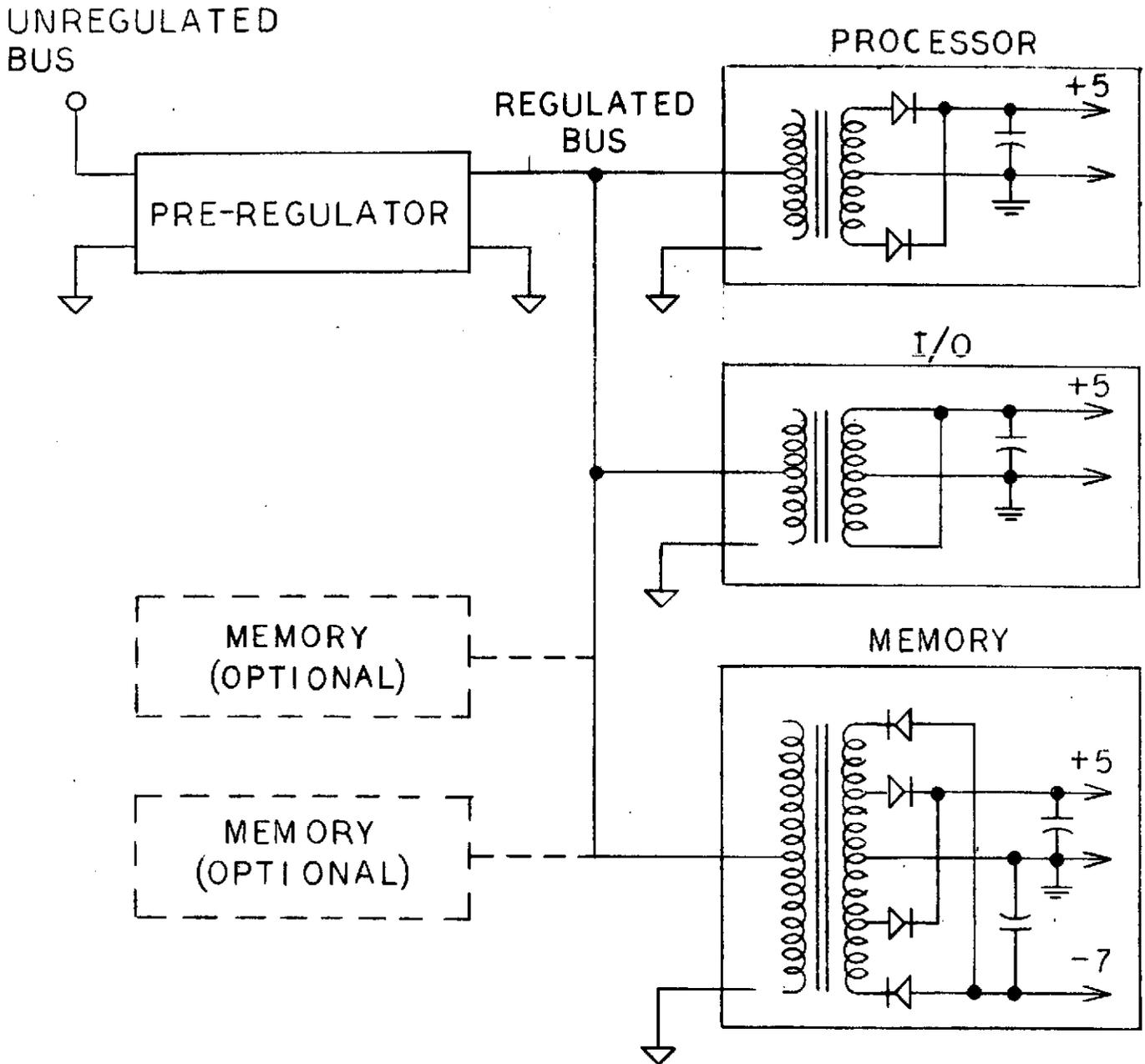


FIGURE 12
DUPLIX CONFIGURATION

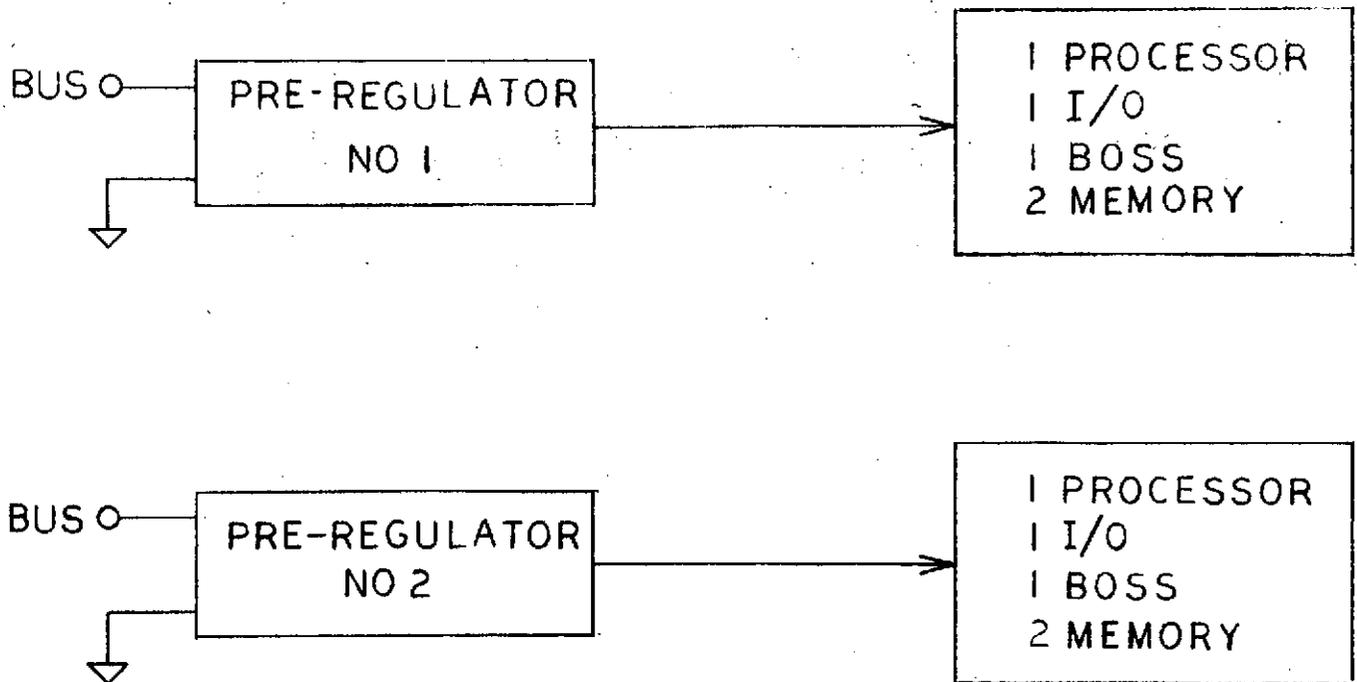


FIGURE 13
TMR CONFIGURATION

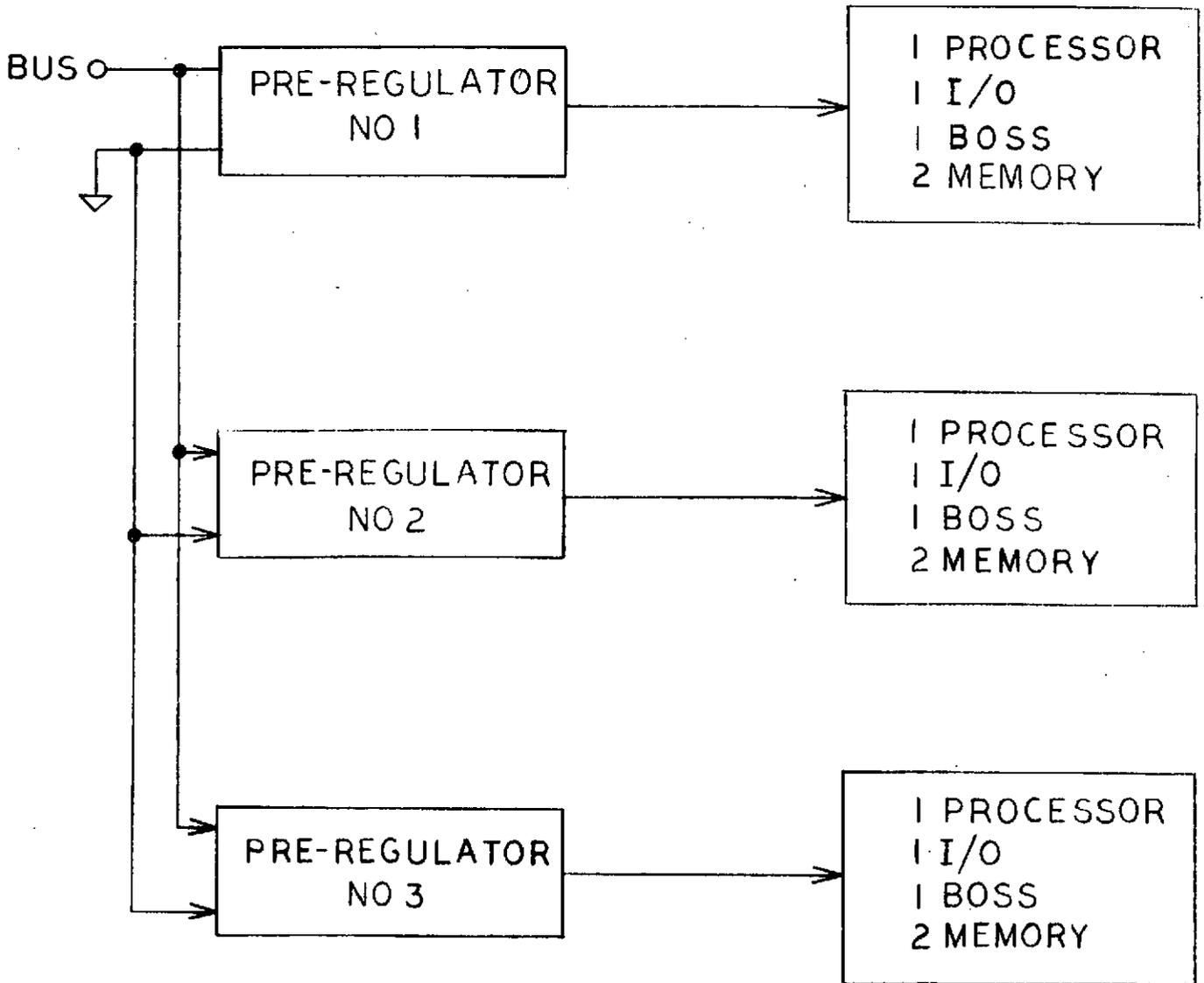


FIGURE 14
TMR + SPARE CONFIGURATION

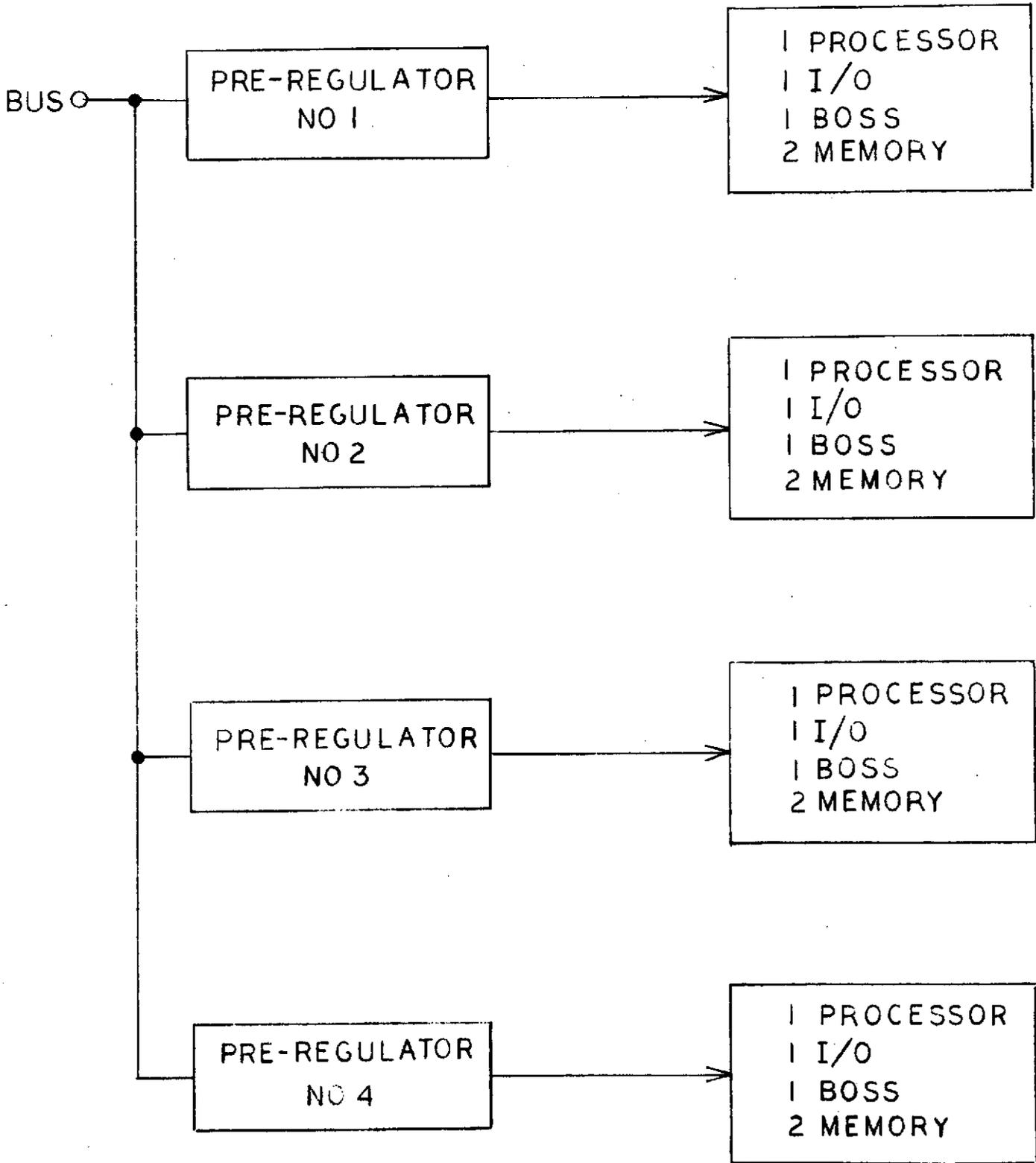


FIGURE 15 MAX CONFIGURATION

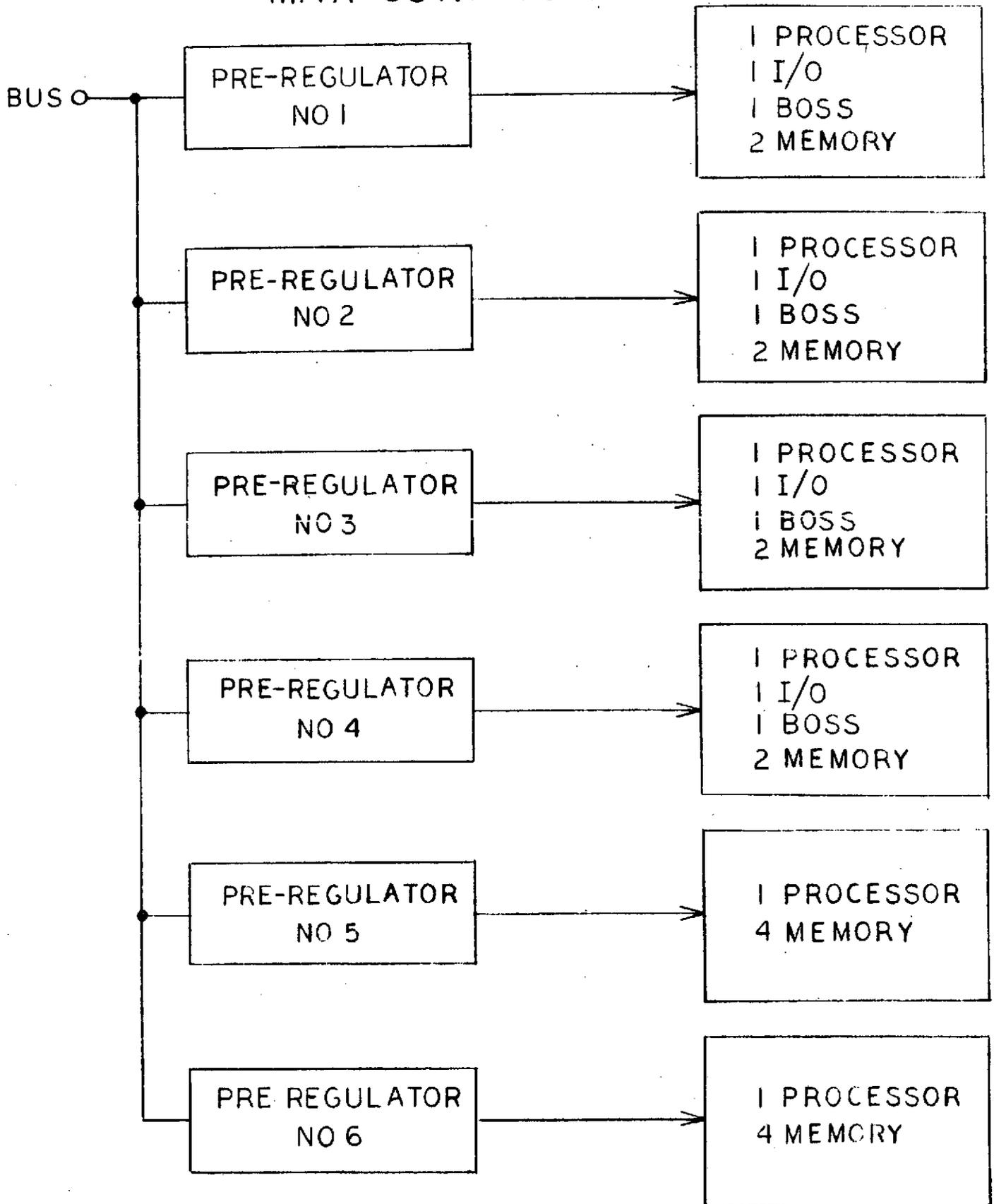
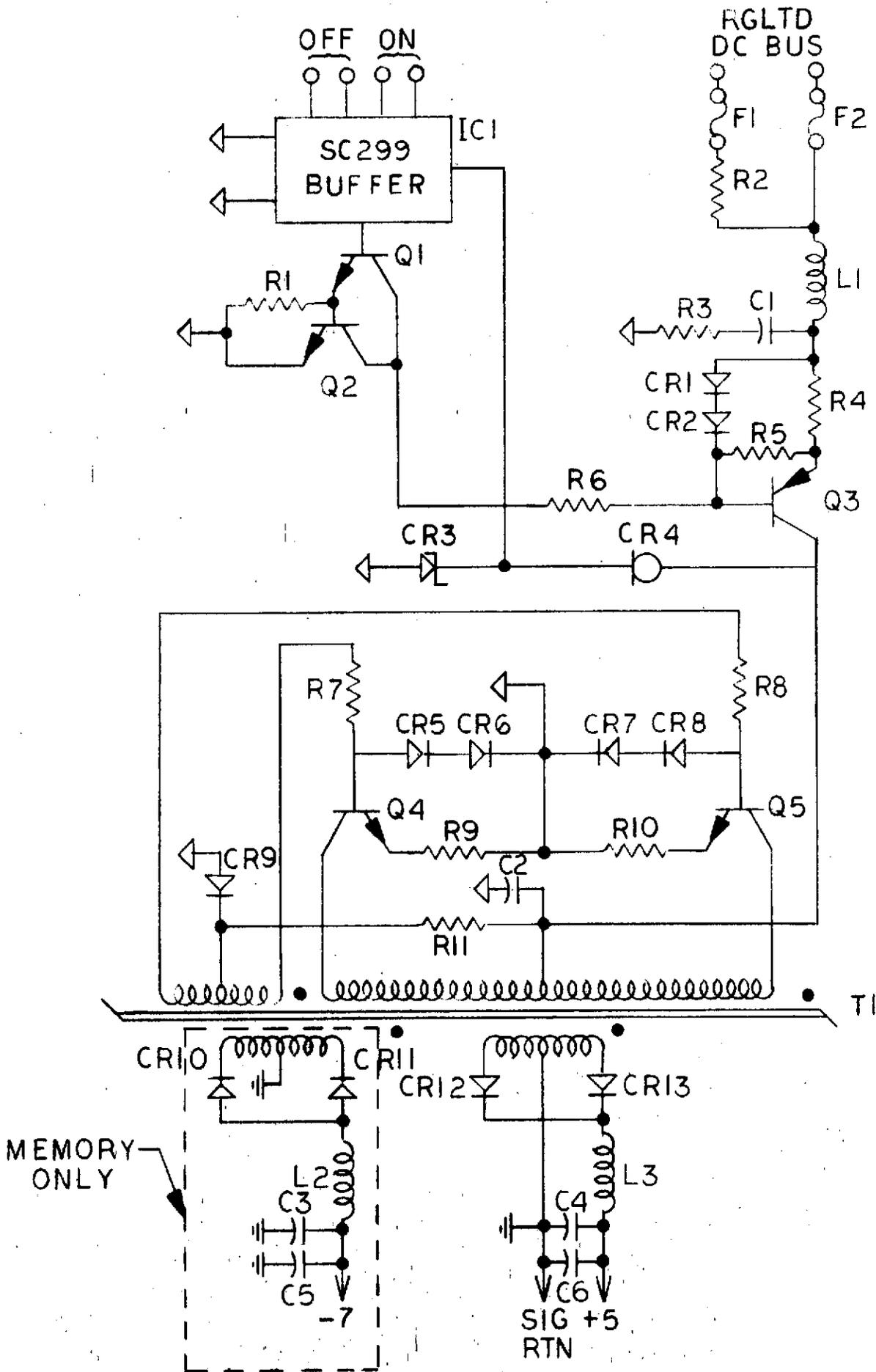


FIGURE 16
PROCESSOR, I/O AND MEMORY POWER SUPPLIES



The DC/DC converter consists of Q₄, Q₅, and associated circuitry. It is a conventional saturating square loop converter. Current surges through Q₄ and Q₅ are limited by R₁, R₂, and CR₁ through CR₄.

Figure 17 shows a detailed circuit design for the BOSS power supply. This power supply is very similar to the Figure 16 supply. The main difference is that the Figure 17 supply is self starting while the Figure 16 supply is commandable. Since the BOSS sends commands to other modules, it must be self starting. The BOSS power supply has an extra +15V output. This +15 volts is used by the BOSS to generate the ON/OFF command levels.

Figure 18 shows a detailed design for the pre-regulator. This pre-regulator is designed as a switching regulator in order to save power. L₁ and associated capacitors form the input filter. Q₁, Q₂, Q₃ form the series switch. IC₁ contains the reference, comparator, and driver stages. Q₄, Q₅, and associated circuitry protect the pre-regulator from an output short. L₂ and associated capacitors form the output filter. CR₁ is the "fly back" diode.

Weight, Part Count, and Power Summary

The Table VI summary is based on the following assumptions:

1. The baseline power distribution implementation will be used.
2. The module secondary power (excludes power supply losses) dissipations are:

Processor	45 W
I/O	15 W
Memory	20 W (Read, Write) 2 W (Standby)
BOSS	45 W

3. The efficiency of a module power supply is 75%. The efficiency of the pre-regulator is 85%. Overall efficiency is 64%.
4. The CMOS logic will be powered by +5 volts. At present CMOS logic can only toggle up to 2.5 MHz at +5 volts. The assumption is that a 3 to 1 improvement in speed will be achieved in the next 3 years. If a greater than 3 to 1 improvement is achieved, it may be possible to operate the CMOS at less than +5V thereby saving a significant amount of power.

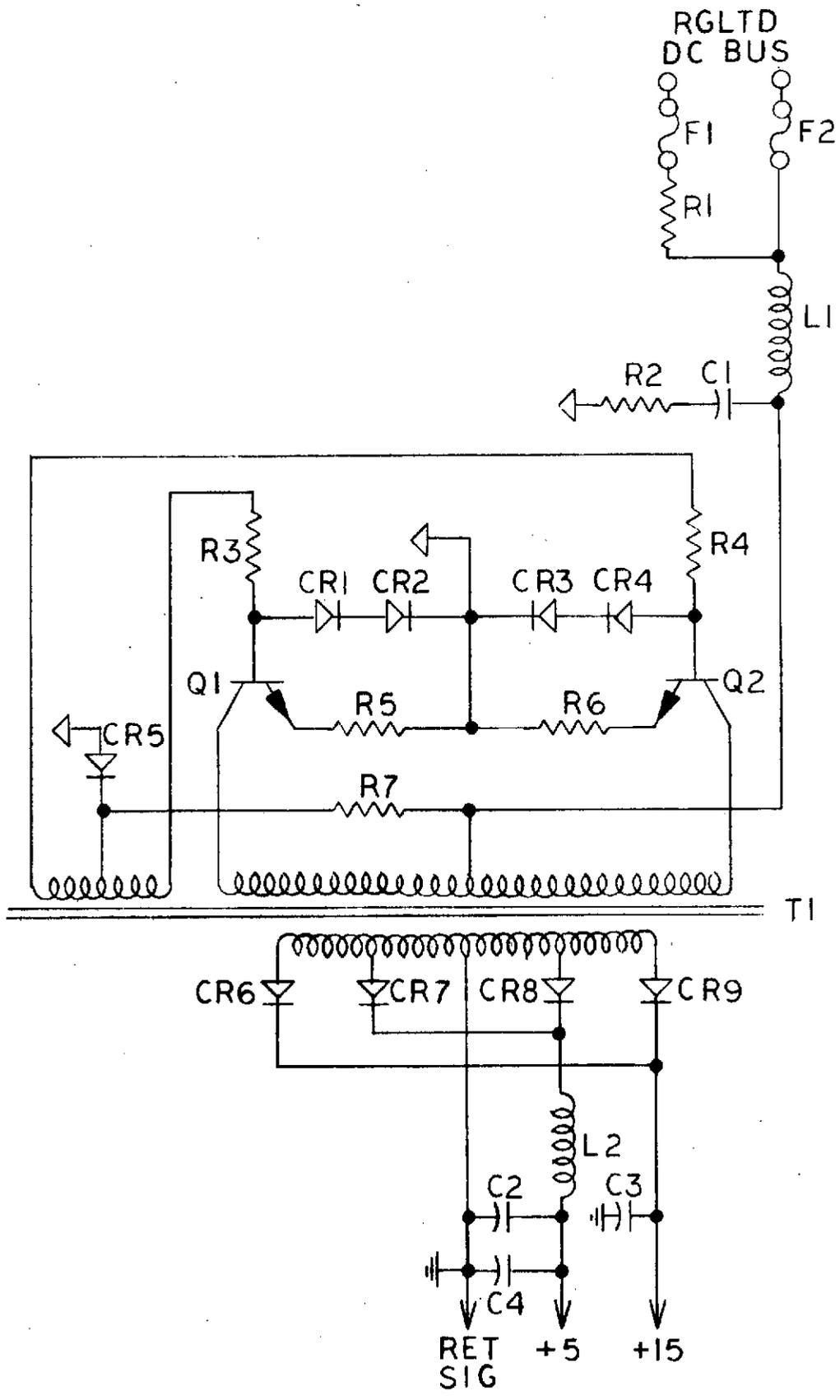
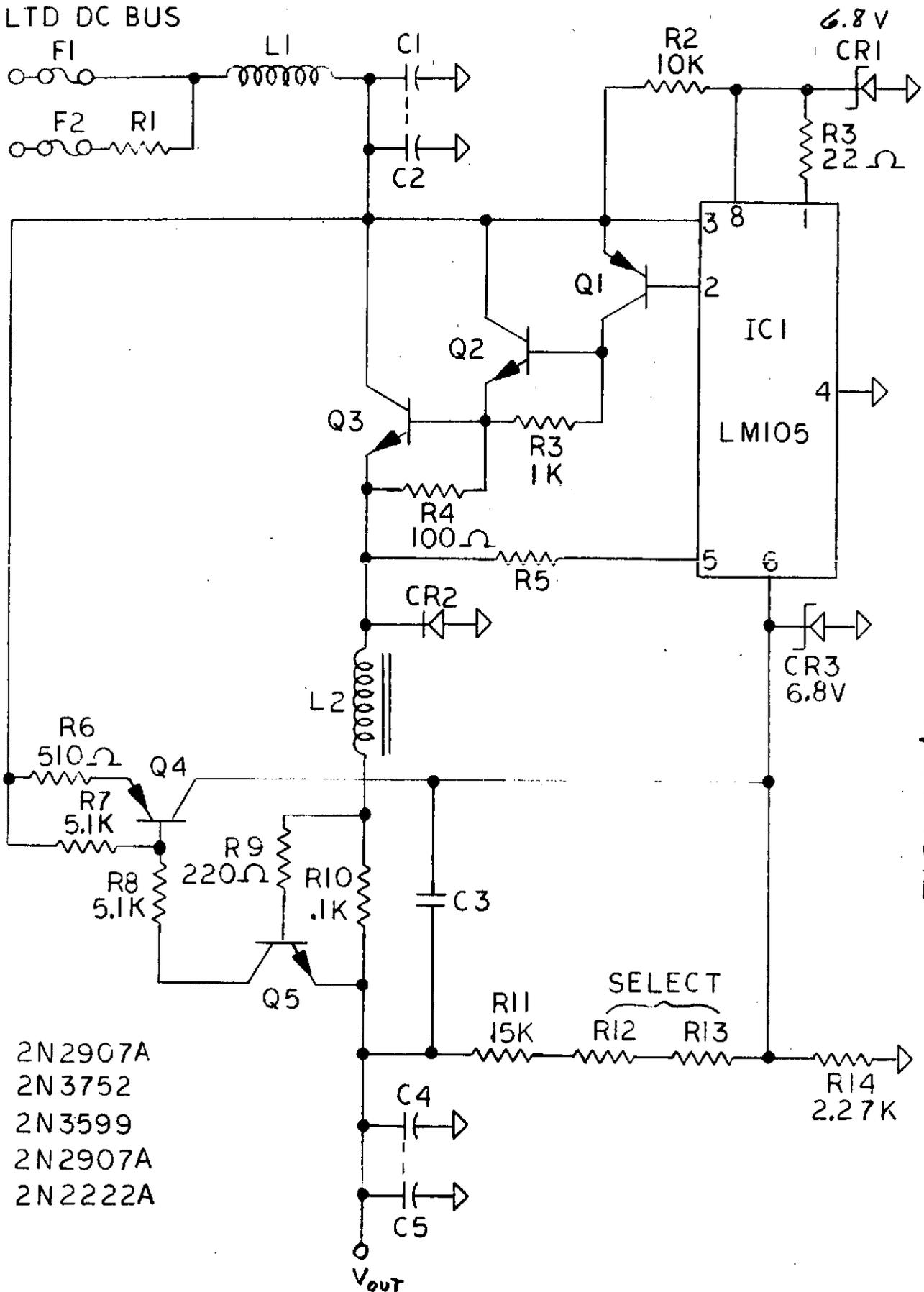


FIGURE 17
BOSS POWER SUPPLY

RGLTD DC BUS



- Q1 2N2907A
- Q2 2N3752
- Q3 2N3599
- Q4 2N2907A
- Q5 2N2222A

FIGURE 18 PRE-REGULATOR

B

5. The module power supply part counts are:

Processor	36
I/O	36
Memory	40
Boss	26
Pre-Regulator	41

TABLE VI.

Configuration	Total Power Supply Part Count	Total P.S. Weight* Pounds	Total Power (Read, Write) Mode		Total Power (Standby)	
			Secondary	Primary	Secondary	Primary
Simplex (1 Memory)	153	10.1	80W	125W	62W	97W
Simplex (3 Memories)	233	14.1	120W	188W	66W	103W
Duplex	438	24.2	290W	453W	218W	340W
TMR	657	41.6	435W	680W	327W	510W
TMR & Spare	976	55.6	580W	907W	436W	680W
MAX	1368	76.2	830W	1300W	542W	845W

NOTE: Secondary power excludes power supply losses.

Primary power includes power supply losses.

*Power supply weight does not include unit structure weight.

Power Supply Reliability Computation

Using the failure rate numbers from the ARMMS Phase I report power supply reliability is very high. Based on the sample circuit design just discussed, the processor and I/O power supply has a failure rate of only .2658 failure per million hours, about 30% as high as its failure rate for the processor itself. The power supply failure rate broken down by components is listed in Table VII.

TABLE VII. POWER SUPPLY RELIABILITY IS VERY HIGH
 BASED ON SAMPLE CIRCUIT DESIGN

Processor and I/O Power Supply	Parts	Failure Rate (per 10 ⁶ hrs.) (each)	Total
Transformer	1	.0065	.0065
Diodes	9	.0004	.0036
Zener Diodes	1	.0008	.0008
IC's	1	.0066	.0066
Fuses	2	.1000	.2000
Resistors	11	.0021	.0232
Transistors	5	.0025	.0125
Capacitors	4	.0007	.0028
Inductor	2	.0049	.0098
	36		.2658

SECTION 5

RELIABILITY MODELING WITH VARYING CONFIGURATIONS AND LOADS

The first part of this section is the manuscript of a paper written by J. J. Bricker and W. L. Martin which describes the most recent reliability model developed for ARMMS which allows the reliability of modular computer systems to be predicted considering configuration and computation load requirements which can vary at deterministic times during a mission. This paper was delivered by Bricker at the 6th annual IEEE Computer Society International Conference (COMPCON) in San Francisco, California in September 1972.

The remainder of this section gives the results of a computer analysis of tradeoffs concerning whether to place ARMMS voter switches internal or external to ARMMS processor and memory modules. The choice as reflected in Configuration C in the ARMMS hardware design section of this report was to use internal voters.

RELIABILITY OF MODULAR COMPUTER SYSTEMS WITH VARYING CONFIGURATION AND LOAD REQUIREMENTS.

Modularity at the level of major functional units (i. e., processors, memories, I/O units) in computer systems serves three primary goals: to enhance the speed achievable with available components through operating modules in parallel; to optimize the configuration used in a given application; and to enhance reliability by the presence of redundant modules. This paper describes a model which allows the reliability of modular computer systems to be predicted considering configuration and computation load requirements which can vary at deterministic times during a mission.

NASA's Marshall Space Flight Center, as an extension of its Space Ultra-reliable Modular Computer (SUMC) program, is studying the potential of modular computer systems in space missions for the post-1975 time frame. The vehicle for this study is called ARMMS for "Automatically Reconfigurable Modular Multiprocessor System". As design objectives, ARMMS is intended to meet the reliability and speed requirements of mission types ranging from launch vehicles to orbital space stations to deep space probes. A peak computation speed of several million instructions per second is desired in a multiprocessor configuration, and in addition modules of a given type are to be usable in Triple Modular Redundant (TMR), Duplex, or other redundant modes to enhance reliability at the expense of computing speed in support of critical tasks. The system consists of processor, I/O, and memory module classes each of which may be required to be further subpartitioned if dictated by reliability considerations. The tradeoff between extensive module subpartitioning and increasing the required number of modules per class was one factor which necessitated the analysis described here. A dedicated executive approach is presently envisioned, with the Executive Control Module responsible for configuration control, scheduling, and I/O management.

To assist in the system design process, a reliability model was desired which would estimate system reliability over the range of cases of interest. A brief description of the model follows.

Model Description

The ARMMS reliability model [1] allows the analysis of a modular computer composed of an arbitrary number of module classes, each containing a specified number of identical modules per class, with active and passive failure rates, each based on a negative-exponential failure distribution law per module per class, and a multiple-phased mission. The time duration of each phase may be arbitrarily given, but it is assumed to be deterministically known. Also the desired configuration of the computer and the minimal allowable configuration

level (below which that phase and the resulting mission would be deemed a failure) may be specified on a per phase and per module class basis. It is then possible to compute the mission reliability at the end of any phase, for each module class and for the computer as a whole, with the basic assumption being that failures are statistically independent. A second key assumption is that intra- or inter-modular switching, failure detection and location, and switch-off and switch-on transients (the effects of which are sometimes designated by the term coverage) [2] behave perfectly and that the voter reliability is equal to one for each voter in the computer which is not internal to a module. Any voter internal to a module is treated as part of that module.

Some of the novel aspects of this reliability model (in addition to providing a powerful design tool) are that 1) the existing treatments of TMR, NMR, and active-standby redundancy, as given in the literature, are unified and simplified; 2) the hybrid concept of redundancy [3] is extended to allow for degraded modes of operation; 3) an entire mission profile consisting of an arbitrary number of distinct phases may be studied and the reliability predicted, via a simple Markov chain approach, so that the numerical evaluation of reliability is reduced to the computation of the product of a finite number of matrices.

In cases where the mission phases are periodic (as might be imagined for a space station, for example), the numerical analysis is further reduced by utilizing the periodicity of the process. If the requirements in time, modular configuration and minimal allowable modular configuration are periodic, then with the aid of basic matrix formalisms (e.g., the Jordan Canonical form) one may simplify the numerical analysis so that only a few Markov matrix products need be computed. This is important for otherwise, for example, in a five-year mission, if there were two periods per day, the number of matrices that must be processed would be 3650×4 for a 4 module class computer. The order of these matrices depends on the initial number of modules per class and if this initial number were M_i for the i -th module class, the associated matrix would be of order $(M_i+1) \times (M_i+1)$.

In non-periodic mission profiles, configuration and minimal allowable levels of module class degradation may vary widely, depending on the specific tasks which the computer must perform over the time spectrum of the mission. While periodicity is not present, the feature of relatively few major phases (in the ranges 30-50 for proposed mission profiles) [4] keeps the reliability prediction process within manageable limits.

Sample Uses of the Model

The model may be used either to establish design requirements or to evaluate proposed configurations versus specific mission requirements.

The effects of various design parameters may be explored over the range of possible values. This is the less taxing use of the model since each module class can be considered individually. For example, module failure rates ranging from 10^{-4} to 10^{-6} failures per hour represent reasonable upper and lower bounds of individual module failure rates in a space environment. Published estimates of ratios of passive to active failure rates range from 0.01 to 1 although the most convincing data suggests that a realistic value for space quality components is closer to the latter than the former [5]. Mission times of interest range from a few hours for booster operations to several tens of thousands of hours for deep space probes. The designer faces the two problems of determining what module reliability can be achieved given the available or projected technology at the time of interest and of estimating the system reliability which can be achieved for given mission characteristics by employing the modules in various possible configurations. The model addresses this second problem.

As a simple example, suppose it is desired to compare the module class reliability over periods of time ranging from 1 hour to 5 years as a function of module active failure rate for three basic configurations: 1) a simplex system in which only one module per class is available; 2) a TMR system in which three modules are available and the system gives correct results as long as 2 of the three agree; and 3) an active TMR set with one active, standby spare. Table 1 tabulates module class reliability as a function of time for each of the three configurations for module failure rates of $\lambda = 10^{-4}$, 10^{-5} , and 10^{-6} failures per hour.

Of greater interest than the specific problem or the data in Table 1 is the sort of conclusion which can be reached using the model. For example, if a module class reliability of 0.999 is required for a 1000 hour mission, then a simplex approach can be considered only if very low module failure rates are achievable. Or, if the same reliability were required for a short term mission (e.g., 100 hours) and if module failure rates of 10^{-5} were believed possible, then the added hardware cost of a highly redundant system would be subject to question. (Of course, separate requirements for back-up systems would also affect conclusions of this type.) Finally, the data shows that high long-term mission reliability requires redundancy even under the most optimistic active failure rate assumptions.

The second, and more demanding use of the model, is to describe and evaluate specific mission profiles and modular computer configurations required during each mission phase. The model has been structured to encompass the three major cases of interest: 1) boost phase or short term missions in which all computer resources may be assumed to be active and available for use; 2) space station or long term earth orbital missions in which the computers

TABLE I. MODULE CLASS RELIABILITY BASED ON CONFIGURATION AND MODULE FAILURE RATE

	λ	1 Hr	10 Hrs	100 Hrs	1000 Hrs	4380 .5 Yr	8760 1 Yr	43800 5 Yrs
M=N=D=1	10^{-6}	.999999	.99999	.9999	.999	.995	.991	.957
SIMPLEX	10^{-5}	.99999	.9999	.999	.951	.957	.916	.645
	10^{-4}	.9999	.999	.990	.904	.645	.416	.0125
N=3 M=3 D=2 TMR	10^{-6}	*	*	*	.999997	.999943	.999773	.99464
	10^{-5}	*	*	.999997	.9997	.9946	.9801	.7119
	10^{-4}	*	.999997	.999705	.974556	.7119	.376	.0004
N=4 M=4 D=2 TMR+1 Spare	10^{-6}	*	*	*	*	*	.999997	.9997
	10^{-5}	*	*	*	*	.9997	.9978	.869
	10^{-4}	*	*	.999996	.9968	.8690	.5530	.0009

* > 0.999999

activity might be periodic, with the periods of activity perhaps determined either by the activity of the on-board personnel or by the period of earth orbit.

3) Deep-space missions which consist of relatively few phases of widely varying time duration.

As a highly simplified example, suppose that a Mars orbiter mission were to consist of an initial 10 hour boost and earth orbit phase (Phase 1) a 3000 hour Earth/Mars cruise phase (Phase 2) and a 500 hour Mars orbit phase (Phase 3). The computer system is to contain 4 module classes; processor ($\lambda = 20 \times 10^{-6}$ failures per hour); memory ($\lambda = 30 \times 10^{-6}$); I/O ($\lambda = 15 \times 10^{-6}$), and executive controller ($\lambda = 20 \times 10^{-6}$). Passive failure rates are taken to be 80% of the active failure rates. During Phase 1, all modules are required to operate in NMR mode while in Phase 2, a simplex computer (one active module of each class) is required and in Phase 3, a high experiment computation requirement necessitates 2 parallel processors, 4 memory modules, and 2 I/O units. In phases 2 and 3, at least two executive modules must survive. Using the model, we can determine the number of modules of each class which must be provided to achieve a system reliability of 0.99 over the entire mission.

The sequence of trials is shown in Table 2. In Trial 1, three modules of each class were taken as a point of departure. It is seen that the system reliability is governed primarily by the memory module, but that at least one more processor and executive module is required. After Trial 2 it is seen that 4 processor and executive modules are sufficient, but that one additional memory and I/O module are needed. Thus a successful configuration is reached in Trial 3. Finally, two additional trials were run to determine the effect of variations in dormant failure rates. In Trial 4, the active and dormant failure rates (λ and μ) were taken to be equal, while in Trial 5, $\mu = 0.1\lambda$. In this case, the effect is distinct but not overwhelming. The entire process of conducting this particular case required 13 minutes at a time-sharing terminal. The model has also been written in batch processing form for use when numerous sample cases are to be run.

Conclusion

The contributions of the model described here are that it unifies the treatment of modular computers of varying configurations and that it is a flexible design tool which provides the designer with the facility to explore the effects on total system reliability of variations in configuration, mission profile, and component reliability.

It assists in the resolution of such basic design issues as how many replicates of each module must be provided, how complex each module may be (considering resulting failure rates), and what forms of configuration (if any) will allow the reliability objectives of a given mission to be satisfied.

TABLE 2. EXAMPLE OF A SEQUENCE OF TRIALS IN DETERMINING AN ACCEPTABLE CONFIGURATION

Trial	1		2		3		4*		5*	
	No. of Modules	Class Reliability								
Processor $\lambda = 20 \times 10^{-6}$	3	.98872	4	.99836	4	.99836	4	.99803	4	.99906
I-O $\lambda = 15 \times 10^{-6}$	3	.99339	3	.99339	4	.99903	4	.99889	4	.99933
Memory $\lambda = 30 \times 10^{-6}$	3	.96869	5	.91590	7	.99483	7	.99292	7	.99742
Executive $\lambda = 20 \times 10^{-6}$	3	.98631	4	.99803	4	.99803	4	.99802	4	.99828
System	-	.96571	-	.90656	-	.99028	-	.98790	-	.99410

*In Trial 4, the dormant failure rate (μ) equals the active failure rate (λ).

In Trial 5, $\mu = 0.1 \lambda$.

It should also be observed in conclusion that the model has not yet reached its final form. As part of an ongoing design program, it is evolving with the design. Most notably, coverage and switching unreliability have not yet been incorporated. These are the next steps in the analysis.

References

1. J. L. Bricker, A Unified Method for Analyzing Mission Profile Reliability for Standby and Multiple Modular Redundancy Computing Systems which Allows for Degraded Performance, Hughes Aircraft, Report FR 72-11-450, April, 1972.
2. W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems", ACM 1969 Annual Conference, P 295-309. Also, in greater detail as IBM Report #RC-2378.
3. F. P. Mathur, "On Reliability Modeling and Analysis of Ultrareliable Fault-Tolerant Digital Systems", IEEE Transactions on Computers, November, 1971, P 1376-1382.
4. L. J. Koczela and G. J. Burnett, "Advanced Space Missions and Computer Systems", IEEE Transactions on Aerospace and Electronic Systems, pp 456-467, May, 1968.
5. E. E. Bean & C. E. Bloomquist, The Effects of Ground Storage, Space Dormancy, Standby Operation, and On/off Cycling on Satellite Electronics, Planning Research Corp, PRC R-1435, May 1970.

ARMMS Voter Switch Placement Study

A study was made during Phase II to determine the optimum placement of ARMMS voter switches – either as additional self-contained modules external to the memories and processors or internal to the memories and processors. The study involved development and execution of a computer program to determine overall ARMMS reliability over the following ranges of parameters:

- | | | |
|----------------------------------|----------|-------------------------------|
| 1. Module failure rates | 1 - 130 | failure/10 ⁶ hours |
| 2. Voter failure rates | 0.3 - 30 | failure/10 ⁶ hours |
| 3. Number of modules | 4 - 32 | |
| 4. Number of voters | 4 - 10 | |
| 5. Mission duration | 1/2 - 5 | years |
| 6. Number of TMR triads required | 1, 2 | |

It was found that the voter placement decision is sensitive to module and voter failure rates as follows:

<u>Failures/10⁶ hours</u>	<u>Result</u>
1 - 1.3	Voter placement has no significant effect
10 - 13	Significant difference in favor of external voter
100 - 130	Long term reliability unattainable

Processor failure rates should be less than 1.0 per 10⁶ hours and hence voter placement will not be significant in their case. Memory modules maskable failure rates fall in the 10 - 13 per 10⁶ hour range with non-maskable failure rates running one-half this number. Hence memory modules are in the region of sensitivity.

Table III summarizes the probability of one TRM triad surviving at several points in the mission for variable numbers of modules and voter configurations where M = No. of modules. The configurations with external voters have slightly higher reliabilities than those with internal voters but it was concluded that the reliability differences were not significant enough to dictate the decision.

Factors favorable to external voters are 1) a small increase in reliability 2) a net reduction in hardware for large numbers of memory modules and 3) increased modularity. The factors favorable to internal voters are 1) lower system pin counts, 2) elimination of the external voter module class, 3) reduction in the number of buses, 4) increased bus speed, 5) reduced executive software complexity, and 6) reduced system power. The tangible factors favoring internal voters are considered to be more important than the small reliability loss involved - particularly since number of buses and pins were not reflected in these reliability calculations, the specific requirement for the marginal added reliability may not exist and moreover the difference could be removed at the system level through the use of additional memory or processor modules. Therefore voters located internally to ARMMS modules at their inputs are recommended.

TABLE 3. PROBABILITY OF 1 SURVIVING TMR TRIAD

Case	M	t=4380	t=8760	t=43800
Internal Voter, Module Failure Rate: 11.5×10^{-6}	4	.99954	.99673	.82571
	7	>.99999	.99996	.96098
	8	>.99999	>.99999	.99205
Internal Voter, Module Failure Rate: 13×10^{-6}	4	.99935	.99541	.77928
	6	>.99999	.99992	.94095
	8	>.99999	>.99999	.98558
External Voter; Module Failure Rate: 10×10^{-6} Voter Failure Rate: 1.5×10^{-6} 4 Voters;	4	.99970	.99779	.86752
	6	>.99999	.99998	.97460
	8	>.99999	>.99999	.99439
External Voter; Module Failure Rate: 10×10^{-6} Voter Failure Rate: 3×10^{-6} 4 Voters;	4	.99970	.99771	.86312
	6	>.99999	.99991	.96966
	8	>.99999	.99993	.98935

BLANK PAGE FOLLOWS

5-9/1