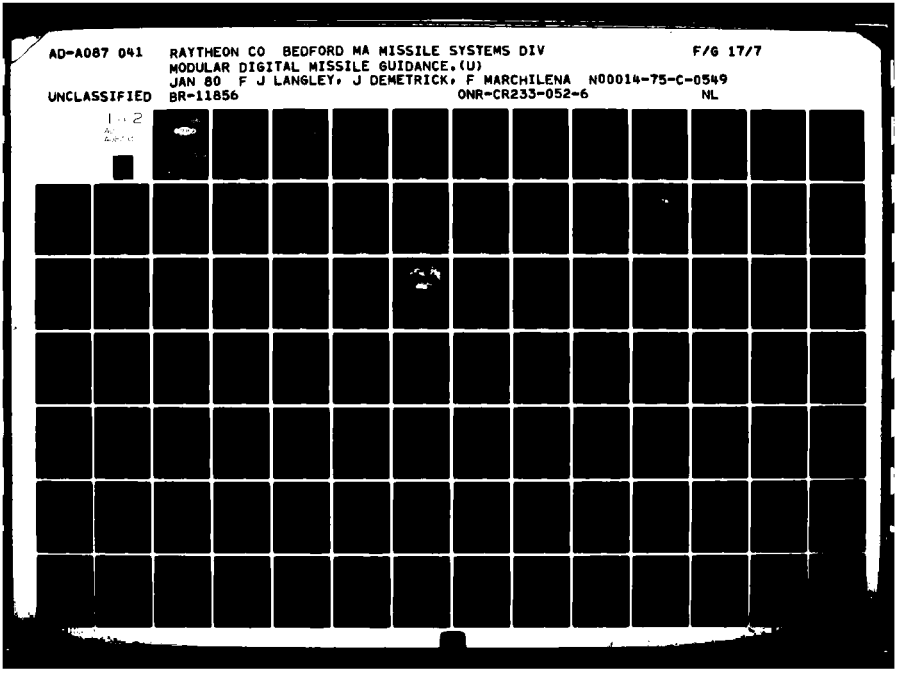
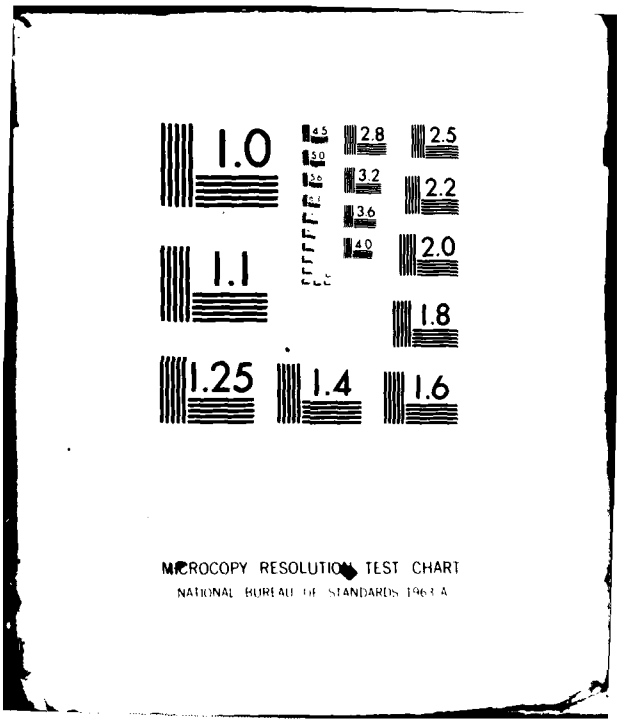
AD-A087 041 RAYTHEON CO BEDFORD MA MISSILE SYSTEMS DIV F/G 17/7
MODULAR DIGITAL MISSILE GUIDANCE.(U)
JAN 80 F J LANGLEY, J DEMETRICK, F MARCHILENA N00014-75-C-0549
UNCLASSIFIED BR-11856 ONR-CR233-052-6 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

REPORT ONR-CR233-052-6F

(1) LEVEL III

AD A087041

MODULAR DIGITAL MISSILE GUIDANCE

Phase VI Final Report

FRANK J. LANGLEY

JOHN DEMETRICK

FRANK S. MARCHILENA

Raytheon Company
Missile Systems Division
Bedford, MA 01730

Contract N00014-75-C-0549
ONR Task 233-052

30 JANUARY 1980

Technical Report for Period 1 Jan. 79 - 31 Oct. 79
Approved for Public Release; Distribution Unlimited

PREPARED FOR THE

OFFICE OF NAVAL RESEARCH • 800 N. QUINCY ST. • ARLINGTON • VA • 22217

NOTICES

Change of Address

Organizations receiving reports on the initial distribution list should confirm correct address. This list is located at the end of the report. Any change of address or distribution should be conveyed to the Office of Naval Research, Code 200, Washington, D.C. 22217.

Disposition

When this report is no longer needed, it may be transmitted to other authorized organizations. Do not return it to the originator or the monitoring office.

Disclaimer

The findings in this report are not to be construed as an official Department of Defense or Military Department position unless so designated by other official documents.

Reproduction

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(18) 1. REPORT NUMBER: ONR-CR233-052-6 — AD-A087 041

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): Modular Digital Missile Guidance

5. TYPE OF REPORT & PERIOD COVERED: Final Report 1/1/79 to 10/31/79

(14) 6. PERFORMING ORG. REPORT NUMBER: BR-11856

(10) 7. AUTHOR(s): Frank J. Langley, John Demetrick, Frank Marchilena

(15) 8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0549

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Raytheon Company, Missile Systems Division, Bedford, MA. 01730

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (12) 63

11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Code 200, Arlington, Virginia 22217

(11) 12. REPORT DATE: 30 Jan 80

13. NUMBER OF PAGES: 154

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(9) Final Rept. 1 Jan. — 31 Oct '79

(6)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microcomputer
Multi-processor
Modular Software
Missile Guidance and Control
Modular Systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Higher throughput general-purpose processors have traditionally been the domain of Schottky-bipolar or CMOS-SOS integrated circuit technology with attendant multi-component packaging logistics and support software problems. In addition, the modular operational software has proven difficult to achieve in practice.

To combat these deficiencies, a super-federated —>

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 24762

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

high-speed, multi-processor architecture has been explored using standard industry 16-bit microprocessors. The resulting design evolved in this study employs a time-phased ring approach to high throughput with single processor programming simplicity. Further, modular software has been achieved by assigning one microcomputer to each major algorithm.

PREFACE

This technical report covers the work performed under Contract No. N00014-75-C-0549 from 1 January through 31 October 1979.

The purpose of this contract together with the work performed in the previous phases, was to provide the means of achieving improved performance, modularity and flexibility in the design of next generation microcomputer-based missile guidance and control systems.

LCDR. W. Savage, Office of Naval Research Arlington, VA, was the Navy LCDR Scientific Officer.

Mr. F.J. Langley was the Principal Investigator for Raytheon, Mr. J. Demetrick was the Hardware Design Engineer and Mr. F.S. Marchilena the Software Design Engineer.

Publication of this report does not constitute Navy approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Cont.)

LIST OF ILLUSTRATIONS (Cont.)

LIST OF ILLUSTRATIONS (Cont.)

LIST OF TABLES

1. INTRODUCTION

This report presents the results of the final phase of the Navy Modular Digital Missile Guidance study which culminates in the design of a high speed "super-federated" microcomputer system. The latter obviates the need to resort to multi component, bit-slice computer architectures to meet the higher throughput processing requirements of missile guidance and control and, more importantly, super-federation supports modular software by assigning one microcomputer circuit to each major algorithm.

1.1 Background

In the previous study phases (References R-1 through R-18), programmable digital techniques were shown to offer improved performance and greater flexibility than the traditional hardwired analog implementations of seeker head control, signal processing, estimation, guidance, autopilot, warhead fuzing, telemetry and test functions.

To achieve modularity and growth in hardware and software, a top-down system study approach was adopted, by first dividing the entire range of air-to-air missiles into three distinguishable generic classes, including upper and lower performance boundaries within each class (Figure 1-1). The major functions and data rates amenable to digital processing were then defined, determining their constituent software modules and sizing these in terms of computer throughput and memory requirements, (References R-1, R-2, R-3 and R-6).

Such a modular breakdown of on-board missile guidance and control functions, together with their associated interfaces, provided the option of configuring and evaluating either single or multiple federated/distributed computer system implementations according to the design constraints of a given missile.

Simulation analyses were also performed to confirm computer requirements and relate algorithm complexity to performance improvements for the guidance, estimation and autopilot/control functions, (Reference R-3).

With the computer design requirements determined from these studies, a set of microcomputer "macromodules" was defined to support the entire range of air-to-air missile functions, in either single or multiple/federated microcomputer system configurations (References R-3 through R-7). The modules were simulated individually and collec-

I	II	III
LOW-COST SHORT-RANGE	HIGH-PERFORMANCE MEDIUM-RANGE	MULTI-MISSION/MODE LONG-RANGE
SINGLE-MODE	SINGLE/DUAL-MODE	MULTI-MODE
FIXED-GAIN AUTOPILOT	BANDSWITCHED AUTOPILOT	ADAPTIVE AUTOPILOT
EITHER INFRA-RED (IR) OR CW-DOPPLER RADAR SEEKER	EITHER 1 OR 2 OF: IR, PULSE-DOPPLER (PD) RADAR OR ANTI-RADIATION (AR) SEEKERS	EITHER 2 OR 3 OF: IR, PD RADAR, AR SEEKERS AND MID-COURSE GUIDANCE

Figure 1-1 - Generic Missile Classes

tively, as whole microcomputer configurations (References R-8 and R-12) in order to validate their effectiveness in missile guidance and control applications to the point where realistic product function specifications could be prepared, (Reference R-14). To broaden the effectiveness of these specifications with respect to current technology developments, the compatibility of the Navy AN/UYK-30 microprocessor for general-purpose processing; charge-coupled device (CCD) technology for signal processing, (Reference R-17); and fiber-optic data link device technology for the serial digital system bus were explored (Reference R-14) through NOSC support.

A similar study was then performed under NSWC sponsorship to address the ship-to-air and ship-to-ship missile requirements, which led to the fabrication of selected microcomputer macromodules and their application to a Class I missile guidance and control system, (Reference R-19).

To accommodate improvements in microcomputer circuit technology without major redesign and provide flexible memory-mapping of input-output and main memory storage space, a programmable Microbus Interface Module (MIM) was designed and incorporated in each macromodule, (References R-16 and R-18).

While standard industry, single chip microprocessors could be used for certain missile functions, their throughput was insufficient for high-performance (Class III) applications such as target seeker and autopilot processing. To avoid the design and fabrication of a high-speed, general purpose processor in Schottky-bipolar or CMOS/SOS circuit technology, with its attendant multicomponent logistics and support software problems, a "super-federated" multimicroprocessor architecture was proposed for investigation and design during this phase of the program (References R-20 and R-21). The goals of this study are described in the following Subsection.

1.2 Objectives and Scope

The objectives and scope of the Phase VI study under the modification of Contract N00014-75-C-0549 are as follows.

The contractor shall continue the Digital Missile Guidance study by performing fundamental hardware and software analysis to determine the feasibility of utilizing one common microcomputer type throughout the modular digital missile concept. In this Phase VI, the following tasks shall be performed:

1) **Task 1 - Microstructural Analysis** - Analyze the high throughput requirement functional groups defined in the previous phases to determine the practicality of decomposing these into microstructures that can utilize one common, single chip microcomputer as the basic computing cell. Investigate the feasibility of software compatibility throughout the functional groups emphasizing replacing software program linkages with hardware interfaces.

2) **Task 2 - Microstructure Simulation and Evaluation** - Perform a digital simulation of the microstructures defined in Task 1 to prove the intermicrocomputer timing and interface and verify that the throughput of each microcomputer group meets the requirements for the complex, Class III missile defined in previous work.

1.3 Publications and Presentations

Throughout the Modular Digital Missile Guidance Program the results of each phase have been widely published and presented to various Government Agencies and Industry. The work has proven timely not only in the field of missile guidance and control but in the design of any microcomputer-based system.

Twenty-two papers and reports have been published. The papers were presented at various conferences sponsored by: IEEE Computer Society, AFIPS/NCC, NASA/JPL, AIAA, DDR&E/IDA, SAE, SPIE, DPMA, NATO/AGARD.

Requests for these papers were received from several overseas countries, viz: Swedish National Defense Research Institute; Center for Applied Research in Electronics, India; Institute of Nuclear Research, Poland; Central Research Laboratory, Mitsubishi Electric Corp., Japan; Rijks University, Holland; The Weizmann Institute of Science, Israel; Institute of Radio and Electronics, Czechoslovakia; Centre National De La Recherche Scientifique, France.

Presentations and briefings were made to several branches of the Navy, Air Force, Army, NASA and allied groups viz: NAVAIR, NOSC, NWC, NPGS, NSWC, SSPO, NAAFI, AFATL, MICOM, ABMDA, NASA/JPL, MIT/Draper Lab.

2. SUMMARY AND CONCLUSIONS

Two serious problems exist in military computer-based systems:

1) The seemingly exorbitant cost of operational software and
2) The absence of a standard microprocessor-on-a-chip for all computing applications.

The first problem has been linked to the size, complexity, and testability of computer programs as well as the level of the programming language and more importantly, the fundamental structure and modularity of the program assigned to a computer, (Reference R-3).

The second problem is purely a technology advancement issue, and the recent advent of commercial 16-bit microprocessor chips with speeds of the order of 500,000 instructions per second (References R-22 and R-23), presents an opportunity to solve both of the above imperfections in the state-of-the-art.

Hence, the main thrust of this phase in the Modular Digital Missile Guidance Program has been to exploit the low cost of the 16-bit microprocessor and microcomputer to solve the high cost of software and computer commonality problems.

The basic concept uses what has been termed "super-federated" computer system design techniques, (References R-20 and R-21) where each major function/algorithm is assigned to a separate microcomputer. A software change is then identified with a specific integrated circuit which, in turn, is part of an overall modular structure. As such, the super-federated design approach reduces program size and complexity, supports software modularity, and uses one microprocessor type. The latter, however, must be configured in a suitable multiprocessor architecture which achieves high throughput in cases where the speed of a single microprocessor chip is inadequate.

This study addresses two major design goals, software modularity and high throughput, while avoiding the pitfalls, (both hardware and software) of earlier multiprocessor designs. The results of the study, which are manifested in the super-federated microcomputer design drawings given at the end of this report, can be summarized in Subsection 2.1.

2.1 Conclusions

Super-federation of individual functions among separate microcomputers in a practical multiprocessor architecture provides the following advantages and/or solutions to computer system software and hardware design problems:

1) **Software Modularity** - By assigning one major algorithm/program module per microcomputer, a software change can be identified with and confined to a replaceable integrated circuit.
2) **Software Control Structure** - The interface between major program modules is a fixed hardware interface which facilitates software changes and bounds the domain of each module. Further, the control hierarchy is implicit in the multiprocessor hardware structure.
3) **Programming Simplicity** - Single computer programming simplicity has been achieved within a common memory map. Each microprocessor is programmed as an entity with the base page of the memory space dedicated to the operational program.
4) **High Throughput** - The high throughput functions in some missiles, (guidance and autopilot, for example), can be performed with several medium performance microcomputers of the same type by exploiting parallelism and overlap in the execution of their constituent program modules.
5) **Standard Mircrocomputer** - Super-federation allows the use of one microcomputer type throughout the missile guidance and control system, singly in such cases as warhead fuzing or in a multiprocessor configuration for the more complex high-speed functions.
6) **Common Support Software** - The use of one microcomputer type for both high and low throughput requirements obviates the need to design and build a high speed processor with a unique instruction set and its attendant support software development cost and risk.

3. DIGITAL MISSILE GUIDANCE AND CONTROL

Despite the many functional advantages of digital versus analog systems, the simple substitution of a small general purpose computer in place of the former analog circuits does not in itself solve all the problems encountered in the life cycle of a missile. The hardcore problems of advancing technology, changing threat situations, systems integration and logistics, together with the ever increasing cost of software, can result in an excessive premium paid for digital missiles.

While throughput could be satisfied with a single, high performance, miniclass computer and a dedicated, special-purpose target sensor signal processor, form-factor and electrical interface problems arise due to the many analog and digital discrete signals being converted and processed at a central point as opposed to being handled at the source. In addition, the design, assembly and checkout of major missile sections/functions, (e.g., seeker, warhead, flight control, telemetry), as completely operational modules are not possible with a single computer design approach.

From a software viewpoint, programming complexity increases with program size and the time multiplexing of individual missile functions to meet the sampling and computational delay requirements of the various control loops. This resulting modification or updating of any given function within the total program is then fraught with virtually unknown and complex software interface problems, a situation which worsens as the level of coding diminishes. In other words, the interface problems cited for analog systems can reappear in digital missiles at the computer input-output interface and in a more devious manner within the invisible internal software. Although software modularity supports the system flexibility requirement for changing threat/mission situations, it has nevertheless proven difficult to achieve and maintain through a development cycle.

3.1 Motivations for Federated and Super-Federated Systems

The motivations for designing and building federated systems stem from the shortcomings of single computer systems cited in the previous paragraphs and the availability of low-cost, large-scale integrated (LSI) circuit microcomputers and associated I/O support circuits.

3.1.1 Hardware

Federated microcomputer systems simplify subsystem design, manufacture, interface, test and the inevitable modifications and updates. Figure 3-1 serves to illustrate the major differences between single and federated computer design approaches. In the case of the single computer system, a relatively large high performance minitype computer is subject to the varying form factor constraints of a missile. To move the computer to a different location invariably entails the repackaging of hardware to fit the space available. A multiwire analog and digital interface problem also results from the concentration of data processing and conversion in one place.

In contrast, the federated microcomputer system performs the data conversion and processing tasks at source, within each major subsystem, and allows a standard serial digital multiplex interface to be used between subsystems and the launcher. This partitioning is discussed at greater length in subsequent sections of this report.

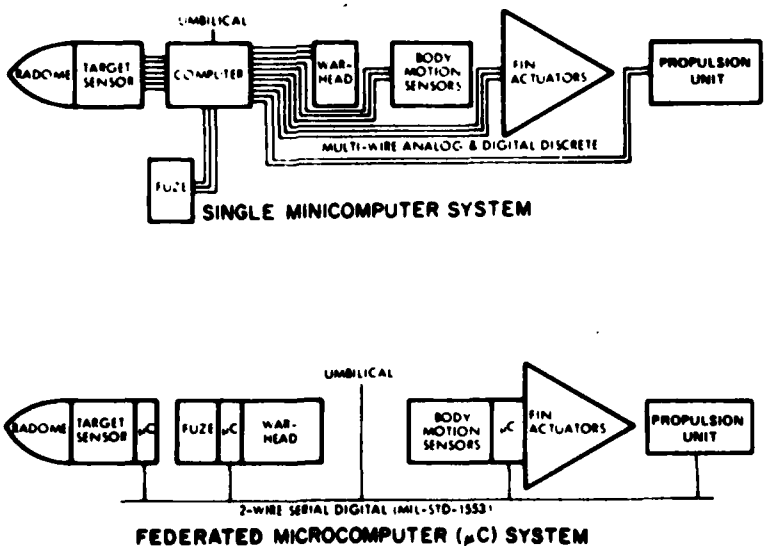


Figure 3-1 - Single versus Federated Missile Guidance and Control Systems

3.1.2 Software

The merits of modular structured software, although well appreciated in this day and age, are somewhat idealistic and difficult to achieve and maintain. Figure 3-2 illustrates a rational, modular, hierarchical control structure for a single computer missile guidance and control system. All calls are made downward from the executive to subordinate mode supervisors and supporting functional program modules. However, under the normal pressure of tight development schedules the finished software is subject to shortcuts which invariably violate the original clean modular lines of the control structure. The outcome of a degradation in software modularity is realized more in the later phases of the development process when changes and substitutions are required (Figure 3-3). Since software costs are pegged to ever-increasing labor rates, the impact of the deficiencies in the design structure, together with other significant factors outlined below, are not apparent until the total cost of the finished product is paid.

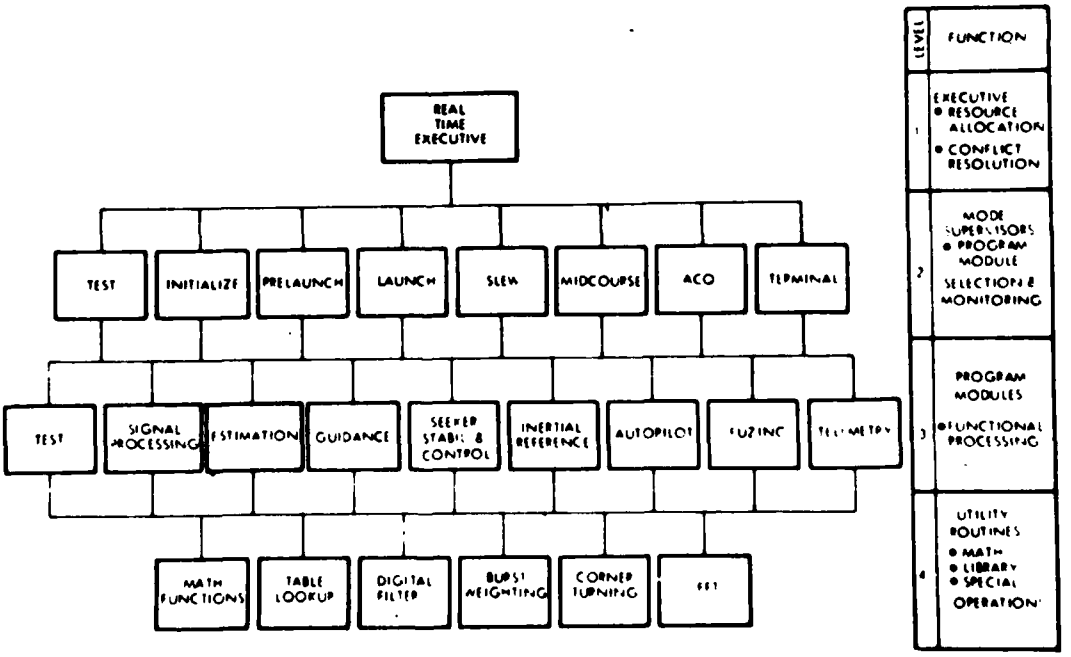


Figure 3-2 - Modular Hierarchical Software Control Structure for Single Computer Missile Guidance and Control System

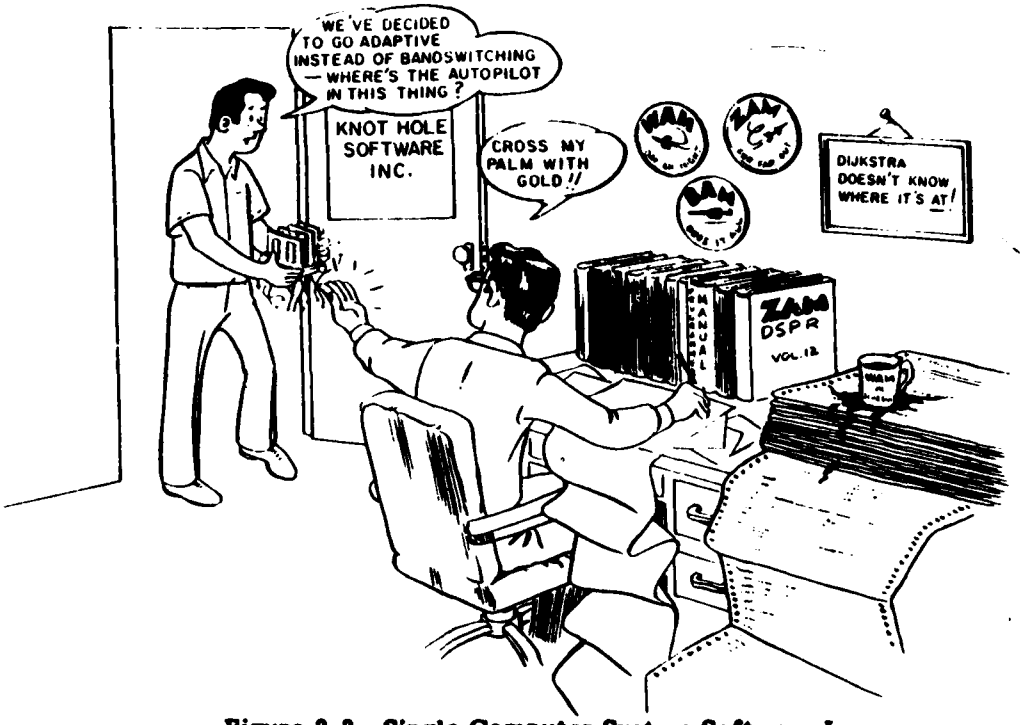


Figure 3-3 - Single Computer System Software Is Inaccessible to Subsystem Designers

Cost Per Instruction =

$$\frac{\text{Design Cost+Coding Cost+Verification Cost+Maintenance Cost}}{\text{No. Lines of Code}}$$

Determining factors:

1) Predominantly labor costs, dependent upon:

   a) Firmness of Requirements
   b) Proportion New versus Proven Algorithms
   c) Size
   d) Complexity

2) Number lines of code, dependent upon:

a) Number Functions Assigned to Software
b) Level of Programming Language

Experienced software costs over the past few years indicate an average cost of $40 to $60 per instruction for a fully commissioned system in terms of new, real-time, operational programs, and between $8 and $30 per instruction for more standard routines. Whereas the cost of semiconductor memory is estimated to be in the order of millicents per bit, a 50-word subroutine typically costs $3000 as a finished product. Microcomputer hardware, on the other hand, enjoys a volume market with modules selling in the tens of dollars. This situation emphasizes the need to be able to reuse or recycle program modules and to curb the tendency of designers to "do it in software" when in doubt about the requirements of a specific system function.

3.1.3 Throughput

Studies have shown that the throughput requirements for single computer systems can reach the two million operations per second (two MOPS) mark (Figure 3-4). While a machine could be designed and built to meet the speed requirement, the tendency has been to add more functions, during the initial system development cycle and later, throughout the life span of the missile. Since there is a finite limit to the speed of the original computer, the increased load must either be accommodated by redesigning the machine or outrigging satellite processors to absorb the overflow which, in turn, tends toward a distributed system of haphazard design.

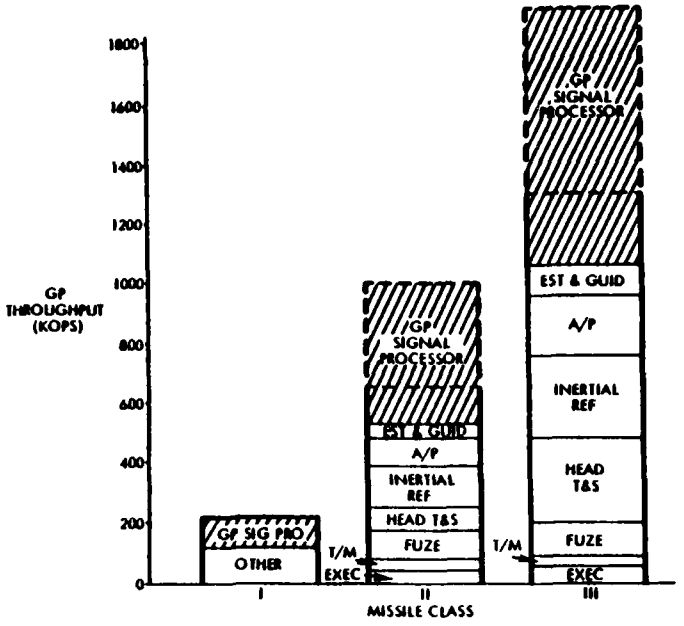


Figure 3-4 - Single Computer System General-Purpose Throughput Requirements

3.2 Defining/Identifying System Structures

Before embarking upon the design of any federated system, it is important to consider the whole system as opposed to applying federated techniques on a piecemeal basis. The reason for this is to identify the characteristic structure of the system in terms of its constituent functions, data flow and data rates. Figure 3-5 shows the major functions of a typical missile system and their relationship to one another.

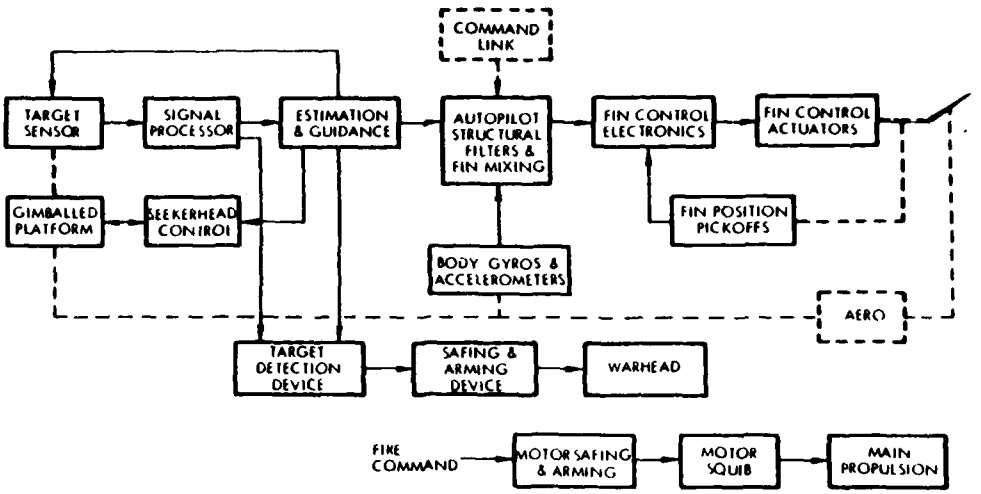


Figure 3-5 - Typical Missile Guidance and Control System - Functional Block Diagram

In the system shown, the target sensor is mounted on a gimballed platform stabilized against missile body motion by platform-mounted rate gyros and torquers in conjunction with the seeker head control electronics. Target sensor, e.g., radar, infrared (IR) electro-optical (EO), outputs are processed by the signal processor which provides target range and angle data (angle only for IR and EO sensors), for subsequent filtering and processing into boresight error and 'g' commands using appropriate estimation and guidance law algorithms. The latter "steering" data controls the seeker platform and autopilot for target intercept. The autopilot also stabilizes the airframe against body motion and bending effects using body gyros and accelerometers as data sources and outputting fin deflection commands to fin control actuators. Detonation of the warhead is determined by the detection of the target by the warhead's target detection device augmented with end game geometry data from the primary target sensor signal processor and estimator. The form of motor control can vary from a simple squibbing signal from the weapon control system, via the umbilical, to sophisticated fuel control based on temperature, pressure and aerodynamic data in the case of ramjet propulsion units.

The degree of interaction between the functional components of the system, their physical relationship, system modularity requirements, and the magnitude of the processing task in each case, influences the structure of the practical distributed microcomputer system.

3.2.1 System Timing Considerations

The basic or characteristic structure of a system, as far as its implementation with distributed microcomputers is concerned, is determined by the system timing constraints and the autonomy of functions. Figure 3-6 shows the system of the previous figure with switches interposed between the major functional blocks and the associated sampling or update rates indicated to satisfy the Nyquist criteria. Three major control loops are visible: seeker head, autopilot and steering command. The first two of the latter require relatively high sampling rates (125-500 Hz), to meet the bandwidths involved, whereas the steering command update rate is quite low (10-20 Hz). Also, the two high speed loops are virtually autonomous with their respective sensors and torquers/actuators.

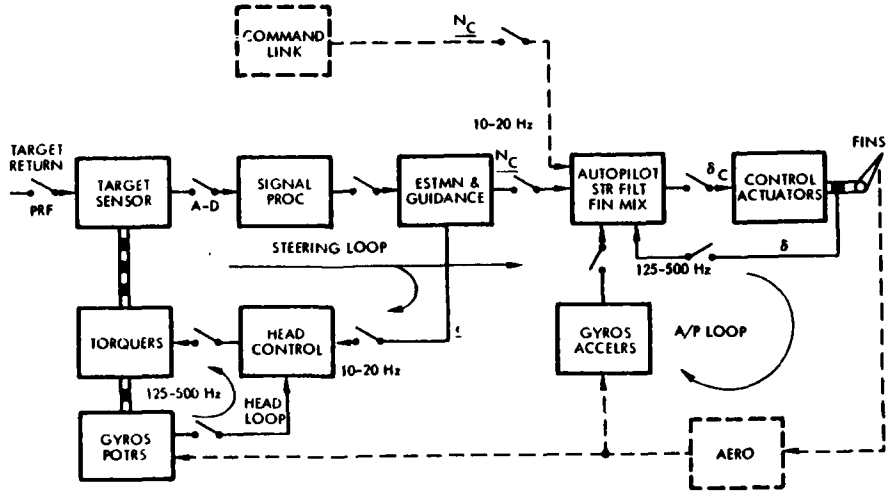


Figure 3-6 - Digital System Timing Considerations

3.2.2 System Parallelism

Figure 3-6 views the system as a set of functional blocks, but if the system is redrawn to reflect the planar control channels of pitch and yaw, for the seeker gimballed platform, pitch, roll and yaw for the autopilot, branching out into four fin control channels, then parallelism becomes evident (Figure 3-7). The latter system characteristic offers potential for using several low throughput microcomputers as opposed to a few high throughput machines.

3.2.3 Macro-Structure System

Based upon the system as it appears in Figure 3-6, the obvious macrostructure which exploits subsystem autonomy and low intersubsystem data rates is as shown in Figure 3-8. One microcomputer is assigned to each major subsystem and a common input-output (I/O) interface interconnects subsystems via a system bus at the low 10 Hz update rate. In terms of control hierarchy, the target seeker microcomputer controls the system bus since all other subsystems are subordinate "users" of the seeker data (Figure 3-9). This form of distributed microcomputer system is a true federated system, since each microcomputer operates virtually autonomously. Further, it meets the subsystem modularity design goal whether subsystems are colocated physically or not. However, there is one major drawback to this level of partitioning, as shown in Figure 3-10.

Throughput requirements for the individual microcomputers vary widely from up to one MOPS to as low as 50 KOPS. As a result, the high throughput requirements of the seeker, flight control and head control functions indicate a bit-slice Schottky-bipolar or complimentary metal-oxide semiconductor, silicon-on-sapphire (CMOS-SOS) device technology machine, and the remaining low throughput functions as a single-chip microcomputer. The cost of designing and building the bit-slice μCs and necessary support software is something to be avoided if possible; hence the need arises to explore alternative implementations using one type of microcomputer-on-a-chip throughout the system.

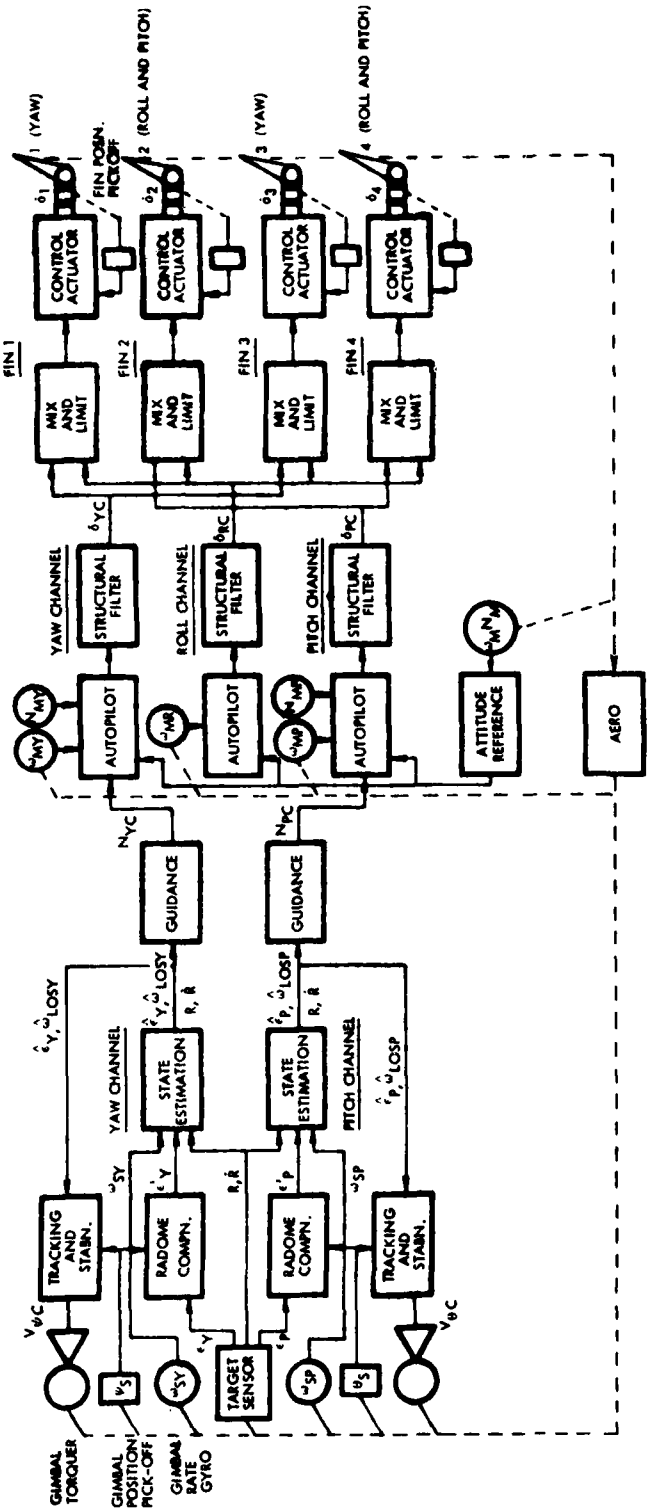


Figure 3-7 - System Partitioning by Control Channel

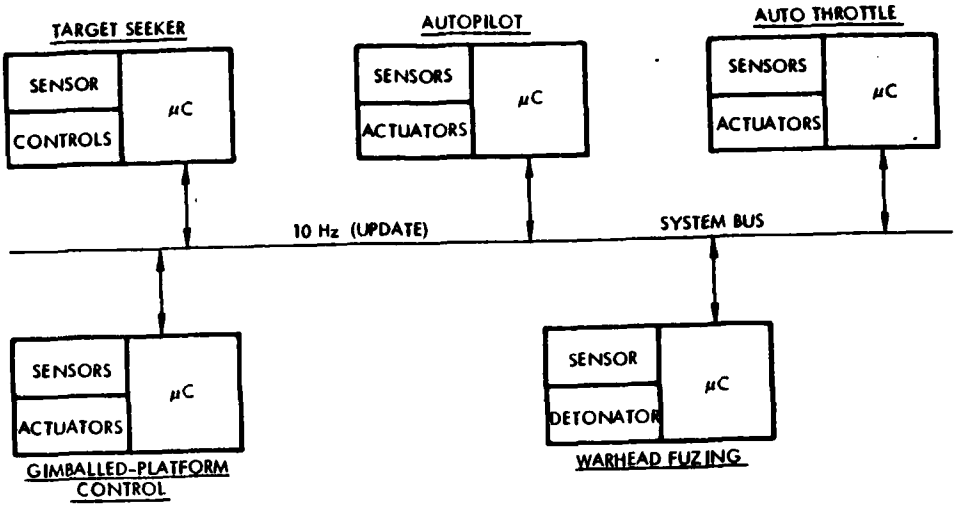


Figure 3-8 - Macro-Structure Partitioning

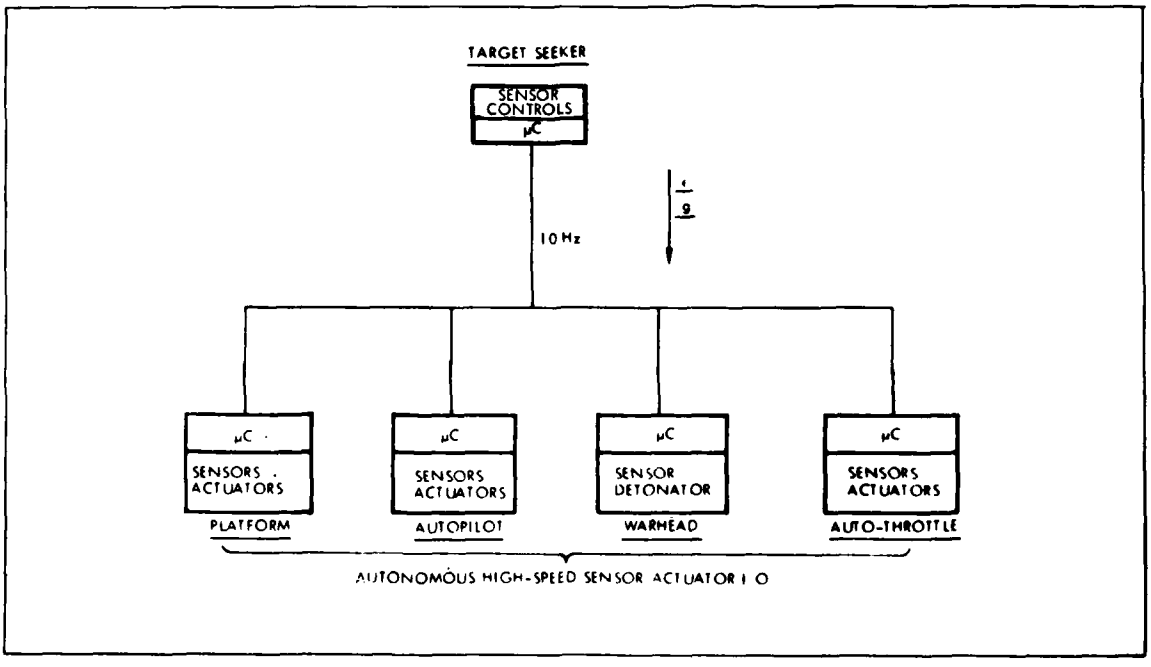


Figure 3-9 - Macro-Structure Control Hierarchy

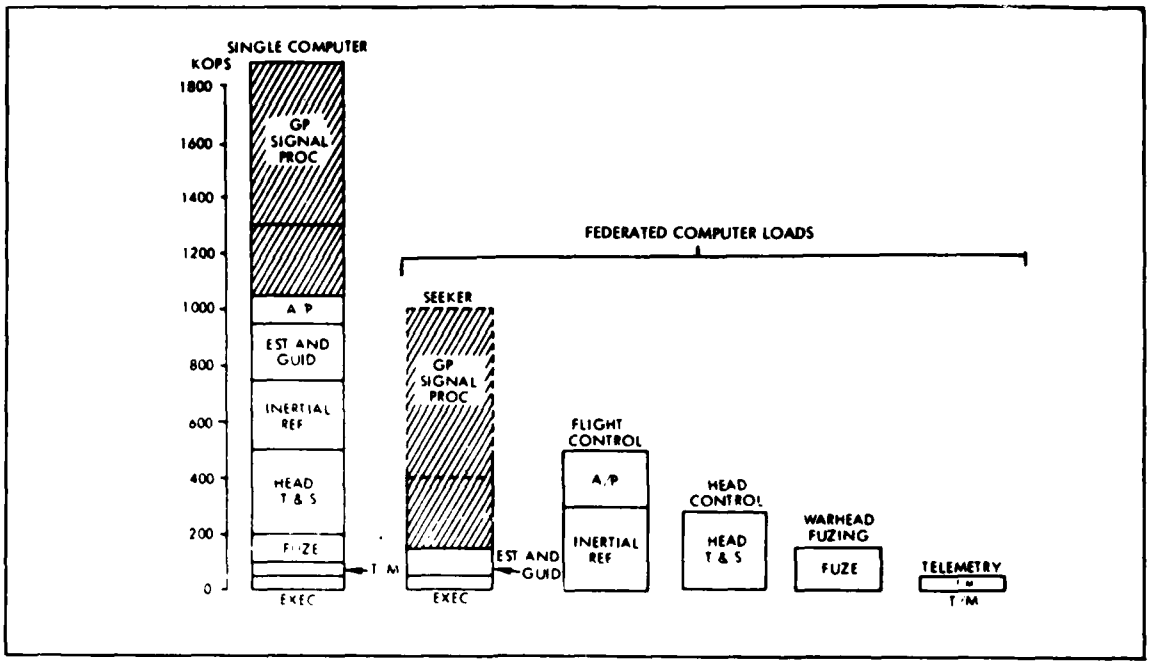


Figure 3-10 - Microcomputer Throughput Requirements for Macro-Structured System

3.2.4 Super-Federated Systems

The high throughput macro functions identified in the previous paragraphs have the potential of being broken down into "microstructures" exploiting the intrinsic parallelism and overlap timing characteristics of the system. Figure 3-11 illustrates the use of separate single-chip microcomputers for each subfunction within the major functions of target seeker and autopilot.

In the case of the target seeker processing group a "heel-to-toe" computing sequence is evident since each microcomputer is waiting for the output of a preceding subfunction. However, certain preliminary operations can proceed while waiting for real-time update information, e.g., state estimation. Further, since the spectrum analysis subfunction is a fixed entity i.e., either a 64, 128 or 256-point fast Fourier transform (FFT) process, then this should be executed in a high-speed special purpose processor to allow more time in the overall budget of 20 or so milliseconds for the slower general-purpose μCs to execute their respective tasks. In other words, software is used where flexibility is required.

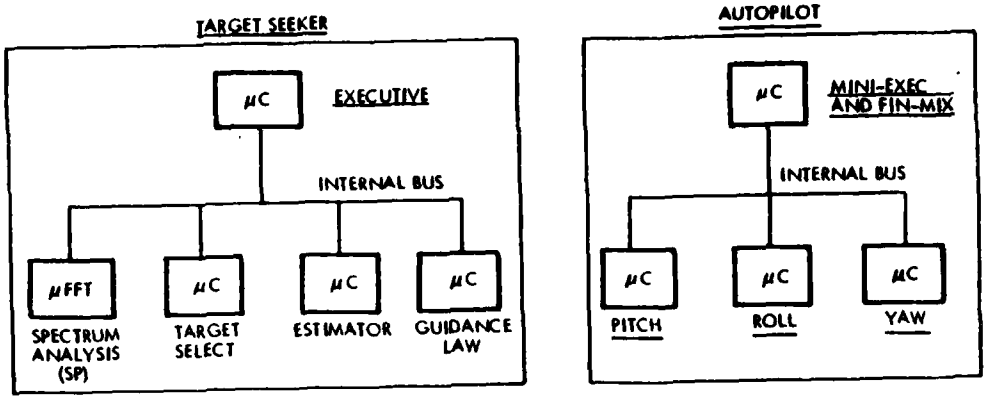


Figure 3-11 - Super-Federated Subsystem Processing, Target Seeker and Autopilot

The autopilot case is quite different. As was noted earlier (Figure 3-7), three-axis control can be exploited through parallel processing, thereby allowing several relatively low speed μCs to be used to perform a high-speed composite function.

Software modularity is enhanced in each of the above cases, since the functional program modules shown in Figure 3-2 are now visible as separate single-chip microcomputers. Taken to an extreme, a 1:1 correlation between the program modules of Figure 3-2 and μCs would ensure software modularity and provide a fixed hardware interface between software routines. Subroutine calls would then be handled by hardware linkages between μCs. The situation depicted in Figure 3-3 could conceivably be transformed into the more desirable state of affairs shown in Figure 3-12, where a subfunction change is performed by the simple replacement of a single-chip microcomputer with the correctly programmed alternative.

Figure 3-12 - Software Change by Hardware Substitution in Super-Federated Systems

3.3 Microcomputer Modularity

To cover the range of missile throughput requirements, a set of microcomputer macromodules was defined (Tabel 3-1). Memory-mapped I/O is used to eliminate Direct Memory Access (DMA) to "main" memory and the associated control circuits. Figure 3-13 shows the grouping together of modules to form a federated missile guidance and control system.

The crux of modularity at the microcomputer level was the definition of a standard microbus , (Reference R-14) oriented toward standard industry semiconductor memory circuit interfaces, i.e., read-write/random-access memories (RAMs) for data storage and read-only memories (ROMs) for programs.

TABLE 3-1

MICROCOMPUTER MACROMODULES

Module	Description	VLSI Circuit Technology	Application
MICROPROCESSORS			
µCPU-1	Medium-Speed (0.5Mips) Microprocessor/ Central Processing Unit, 16-Bit Word, Fixed-Point, General-Register.	N-MOS, CPU-on a Chip,	o Telemetry o Fuzing o Head Control o Autopilot
µCPU-2	High-Speed (2Mips) Microprocessor/ Central Processing Unit, 16-Bit Word, Fixed-Point, General-Register,	Multi-µCPU-1 Processor.	o Autopilot (Adaptive) o Signal Processing o Estimation o Guidance
HIGH-SPEED ARITHMETIC AND MEMORIES			
HMPY	High Speed Multiplier, Memory-Mapped 200 nsec, Max. 16x16-bit Multiply	CMOS-SOS Single Chip or Hybrid	o Throughput Enhancement for µCPU E.G. Class I Sig. Proc.
FSA	High-Speed Frequency Spectrum Analyzer Memory-Mapped, 150 µsec Max. for 64-pts, 8 + J8. Preprogrammable for 128, 256 or 512 pts.	CMOS-SOS FFT or CCD/CST	o Throughput Enhancement for µCPUs E.G. Class II & III Sig. Proc.
RAM-1	Random-Access, Read/Write Memory, Medium Speed, 128-2Kx16-Bits 300 nsec Max. Access Time	N-MOS Single Chip or Hybrid	Data o Telemetry o Fuzing o Head Control o Autopilot
P/ROM-1	Programmable (Mask/Electrically) Read-Only Memory, Medium Speed, 1K-16Kx16-Bits, 300 nsec Max. Access Time	N-MOS Single Chip or Hybrid	Programs
RAM-2	Random-Access, Read/Write Memory, High-Speed 256-1Kx16-bits 100 nsec Max. Access Time	CMOS-SOS Single Chip or Hybrid	Data o Sig. Proc. o Estimation o Guidance o Head Control o Autopilot o Fuzing
P/ROM-2	Programmable (Mask/Electrically) Read-Only Memory, High-Speed, 1K-4Kx16-bits 100 nsec Max. Access Time	CMOS/SOS Single Chip or Hybrid	Programs
INPUT-OUTPUT			
PDIO	Parallel Digital Input-Output Channel. Memory-Mapped. Parallel Word and Discrete Transfers	CMOS-SOS/Bipolar Single Chip or Hybrid	o Telemetry
ADAC	Analog to Digital/Digital to Analog Input-Output Channel. Memory-Mapped. A-D: 8 Chs., Mux. 8/12-Bit, A-D 3/8 µsec Max/Ch. D-A: 4 Chs., 12-bit D-A, 5 µsec Max/Ch.	CMOS-SOS Single Chip or Hybrid	o Head Control o Autopilot o Telemetry o Radar Receiver
SDIO	Serial Digital Input-Output Channel. Memory-Mapped. Word & Bit Serial Data/Command Transfers, 1Mbit/sec Max. MIL-STD-1553A/B	CMOS-SOS Single Chip or Hybrid	o Avionics o Inter Micro-computer

S/H - sample-and-hold
Mux - multiplexer
Demux - demultiplexer

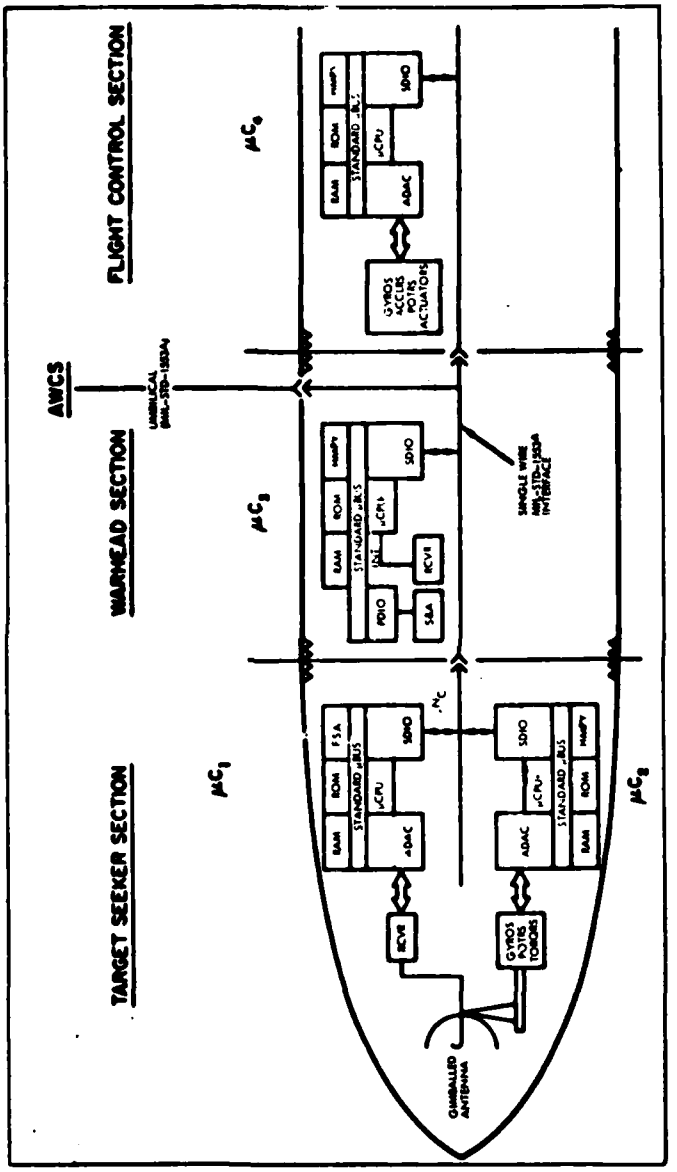


Figure 3-13 - Modular Federated Microcomputer Missile Guidance and Control System

3.3.1 Programmable Microbus Interface Module

In the microcomputer industry no two microbus interface schemes are the same, e.g., S-100 Bus, Intel MULTIBUS, National Microbus, etc., and similarly, the electrical interfaces of available support modules varies, e.g., analog-to-digital (A-D) and digital-to-analog (D-A) converters, memory modules, and serial digital interface modules.

Through the definition of an independent microbus interface a programmable microbus interface module (MIM),(References R-16 and R-18) was designed. This module employs high-speed field programmable logic arrays (FPLAs) and programmable read-only memories (PROMs), to interface standard-industry microcomputer components, i.e., microprocessors, RAMs, ROMs, multiplexer A-D converters, D-A converters and serial digital I/O modules with the microbus. Further, each individual component can be replaced with a more desirable product from a different manufacturer, at any time during the life cycle of the system, by reprogramming the MIM to accommodate the interface peculiarities of the new product.

3.3.2 Frequency Spectrum Analyzer Module

Missile radar target seeker signal processing requirements are low compared to avionic and ground-based air defense systems (Reference R-3) (Figure 3-14). Nevertheless, the frequency spectrum analyzer (FSA) module of Table 3-1 using bit-slice microprocessor circuits, requires approximately 150 LSI/MSI/SSI circuits, dissipates approximately 50 W, using Schottky-bipolar circuit technology, and executes a 64-point complex FFT in approximately 300 μsec, meeting only Class I and II missile performance requirements (References R-6, R-14 and R-17). Such a processor dwarfs the single-chip microcomputer (Figure 3-15). In contrast, a charge-coupled device (CCD) processor using the chirp-Z transform (CZT) and transversal filters, executes an equivalent 64-point analysis in approximately 13 μsec, with a power dissipation of less than 5 W, meeting all three missile class requirements (References R-17, R-24 and R-25). While dark current is a limiting factor in the dynamic range of analog CZT processors at the upper end of the MIL temperature range, recent improvements in prototype surface channel CCDs at Raytheon and elsewhere (Reference R-26) indicate a temporary situation in this performance deficiency. Further, based on recent NASA/TI work, a 2-chip CCD CZT processor appears feasible in the near future.

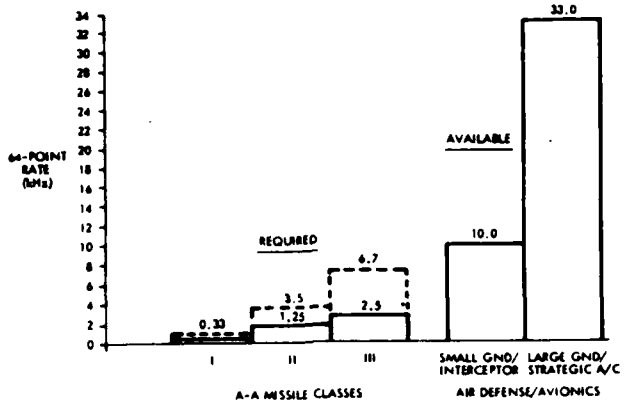


Figure 3-14 - Radar Signal Processing Throughput Requirements

3.3.3 Serial-Digital Input-Output (SDIO) Module

The SDIO module provides a MIL-STD-1553B- compatible serial digital multiplex bus interface between microcomputers in the missile and the external weapon control system (Reference R-27). Using conventional transformer coupling to the transmission line requires relatively large, high-current, line driver, receiver and transformer components which, in turn, are inconsistent with today's single-chip microcomputers and the small size, weight and power limitations of a missile. Fiber-optic coupling between subsystem microcomputers, using simple LED/PIN diode/T^2L interface components (Reference R-28) and single-chip Manchester II/NRZ code converters (Reference R-29) reduces the serial I/O interface hardware to more realistic proportions (Reference R-14). However, the single party-line bus is not currently amenable to fiber-optic technology, since T-couplers introduce a 3 dB loss at each drop point. A simple alternative is the ring system of Figure 3-15, using a round-robin protocol (Reference R-30). A

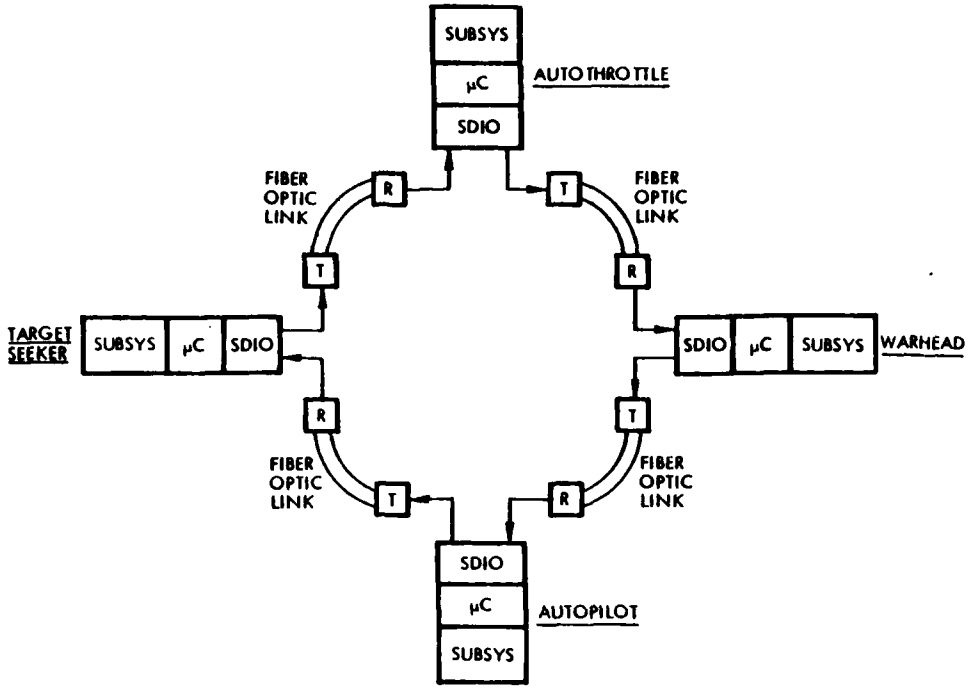


Figure 3-15 - Fiber-Optic Ring Communications Between Missile Subsystems

more complex multiline approach is the star configuration which would be suitable for a simple, single-mode, short-range missile where the seeker becomes the focalpoint. Eight-port couplers have been built under Air Force contracts (Reference R-31).

3.4 Navy Demonstration System

The culmination of the above work has been the fabrication of a basic federated microcomputer guidance and control system under a NSWC contract (Reference R-19). This microcomputer system constitutes the "hardware-in-the-loop" element of a real-time missile simulation to evaluate the performance of the federated approach under the constraints of a MIL-STD-1553B I/O protocol (Figure 3-16).

Breadboard versions of the μC macromodules have been designed and built using standard industry μC components integrated with microbus interface modules (MIMs), Figure (3-17). The simulation is based upon a modular digital missile guidance simulation

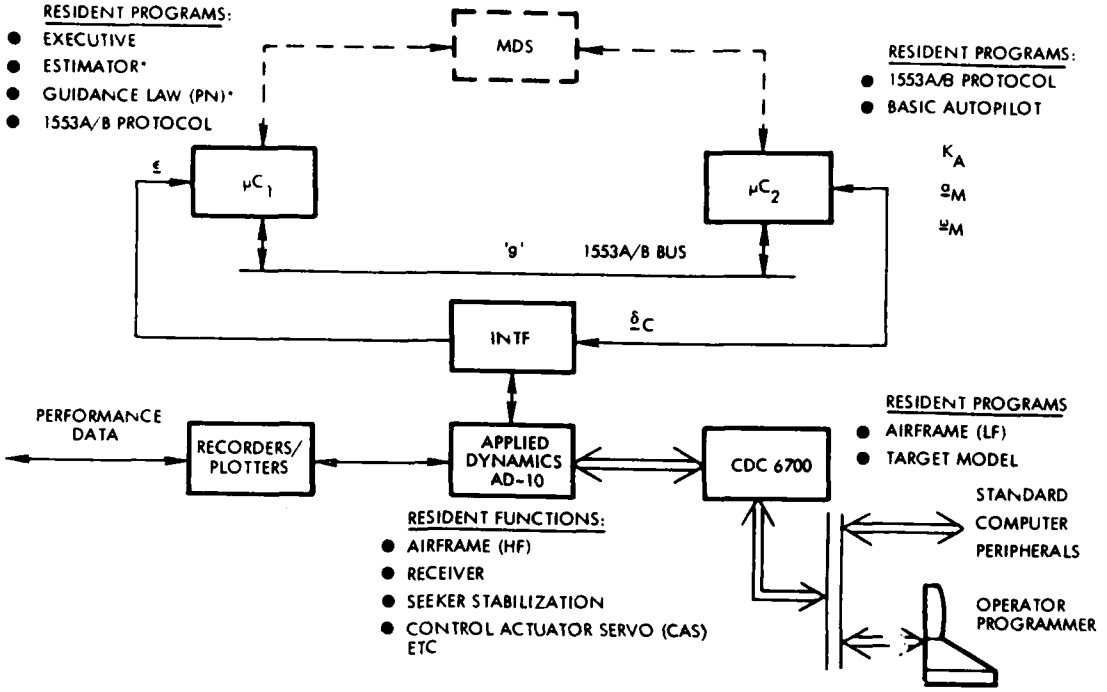


Figure 3-16 Navy/Raytheon Hardware-in-the-Loop Federated μC System for Missile Performance Simulation

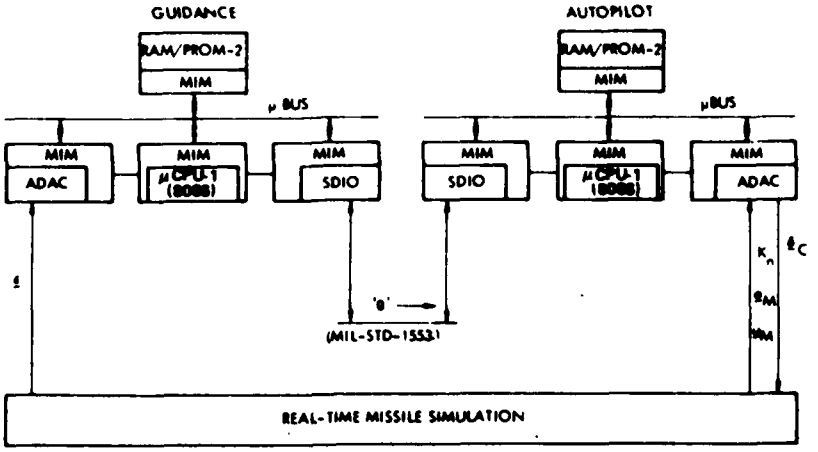


Figure 3-17 - Federated Microcomputer Macromodules Using MIM Interface

system developed for NSWC under a separate contract, (Reference R-32). System growth is achieved by adding additional microcomputers to the system bus and transferring/recoding simulation program modules to be executed by the appropriate microcomputer(s).

Figure 3-18 shows modular growth from the simple low-performance guidance and control system of Figure 3-17 to a high-performance super-federated system using several microprocessors of the same type and maintaining the original data memory and I/O modules. Each microprocessor executes only one algorithm using a dedicated program memory chip. This arrangement ensures software modularity and programming simplicity while minimizing μbus traffic.

3.5 Summary

Federated microcomputer systems provide the flexibility to design, develop, modify and update missile guidance and control systems on an individual subsystem basis, thereby enhancing system modularity. Standard industry microcomputer components which meet military environmental specifications can be integrated into a set of microcomputer macromodules using a standard programmable interface module and microbus. To achieve and maintain modularity in software, the potential exists to assign each major program module to a separate single-chip microcomputer, placing a fixed hardware interface between major function algorithms. Furthermore, by exploiting parallelism and/or the time overlapping of function execution, the use of several standard-industry, single-chip microcomputers in a "super-federated" configuration eliminates the need to resort to one-of-a-kind, high-speed, bit-slice processors for high performance missiles, with their attendant hardware and software logistics support problems. In terms of signal processing, improved charge-coupled device technology, in the form of chirp-Z transform processors using transversal filters, offers a solution to the high chip/parts count of current fast Fourier transform processors. A two-chip CZT processor would match the level of large-scale circuit integration presently available in microcomputer technology.

In cases where federated microcomputer systems are distributed physically throughout a missile, relatively low performance, fiber-optic, serial-digital communications between microcomputer-based subsystems using a round-robin protocol eliminates the high power, transformer-coupled interface of traditional electrical bus systems.

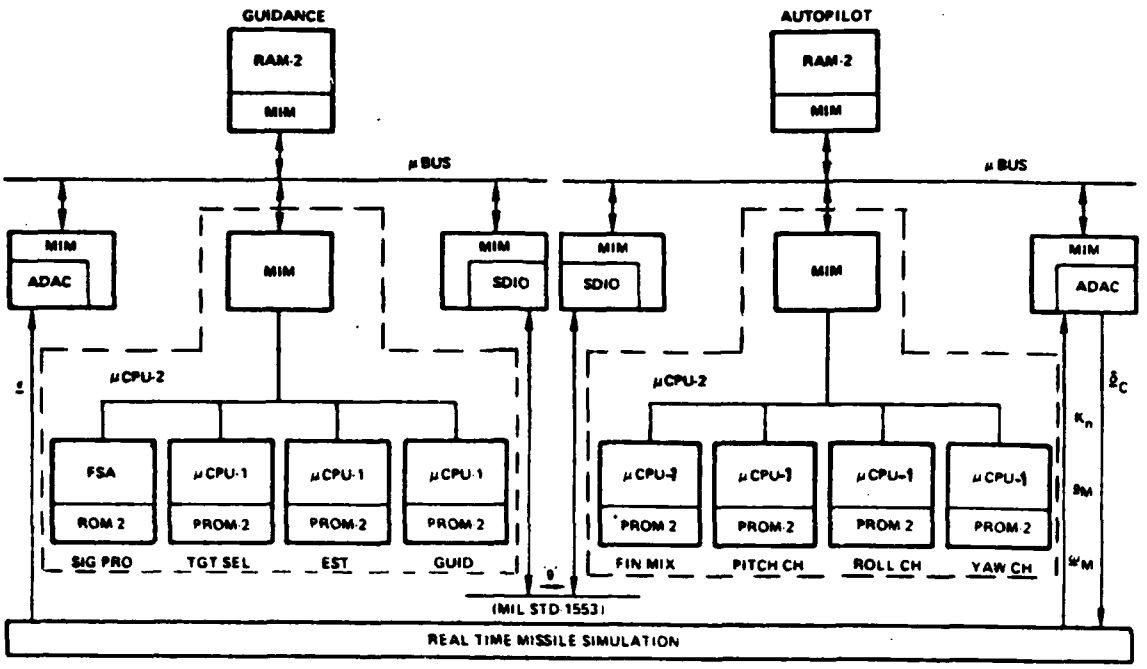


Figure 3-18 - Super-Federated Microcomputer System for Higher Performance Missile Guidance and Control

4. CLASSIC MULTIPROCESSOR ARCHITECTURES

Before embarking on the super-federated microcomputer system architectural design, a review of earlier multiprocessor architectures was performed to determine their respective merits and failings both from a hardware and software viewpoint.

Although large physically, due to the state-of-the-art in hardware at the time of construction, these earlier architectures become classical in terms of the various approaches adopted to solve such problems as throughput, availability and growth for both random and highly repetitive computing tasks.

Further, the deficiencies experienced in these architectures (particularly software), are as important today as earlier, except of course for the shortcomings resulting from the number of discrete components and their associated failure rates.

4.1 Multiprocessor and Computer Systems

To overcome the deficiencies of single, uniprocessor computer systems for high-performance, high-availability applications, viz: limited throughput; failure upon a single fault; restricted growth in size and performance; various architectures incorporating either several of the basic elements of a computer, i.e. CPUs, memories, and IOUs, (multiprocessor systems), or several whole computers (multicomputer systems) have been designed and built. These systems are characterized by their ability to perform the simultaneous or parallel execution of similar and/or different tasks at several times the speed of a single sequential machine.

Multiprocessor systems are essentially expanded and more complex versions of the basic Von Neumann uniprocessor, (Reference R-33), usually performing a centralized role in a given system. However, a significant drawback to certain types of past multiprocessor systems has been the executive processing load associated with the efficient utilization of processors in a multi-task operating environment, (Reference R-34). Since this overhead remains sequential, system throughput is not linearly proportional to the number of processors employed.

Multicomputer systems, on the other hand, are composed of several relatively simple and familiar computers interconnected via their I/O units (IOUs). Multicomputer

systems normally function as decentralized, distributed/federated systems with each computer dedicated to a specific set of interrelated tasks and external devices (EDs).

Communication within a multiprocessor system involves megaword per second information transfer rates whereas within a multicomputer system, i.e., between IOUs, functional partitioning is designed to achieve transfer rates in the kiloword per second range.

The various characteristic forms of multiprocessor and multicomputer systems built to date involve either single or multiple data busses, or a cross-point switching matrix for communication between major computer elements or computers respectively. In the following examples reviewed it can be seen that it is both the type of communication employed and the degree of customization of the architecture to the type of processing task to be executed which characterizes the system, whether multiprocessor or multicomputer.

4,1.1 Single Time-Shared/Party-Line Bus

The most simple form of multiprocessor and multicomputer system employs a single, time-shared/party-line communications bus. These architectures achieve the highest throughput only when accesses to the bus can be scheduled to avoid user conflicts. Of the two types of computer system, the multiprocessor (Figure 4-1) is more throughput limited by the single bus than its multicomputer counterpart due to the lack of autonomy of the individual processors and their dependence upon access to a common/shared "main" memory. The multicomputer system (Figure 4-2) however is far more amenable to the single-bus for inter-IOU communication due to the low transfer rates in a properly partitioned system. In the example shown, bus accesses can either be controlled by a master-slave hierarchy or on a round-robin basis to eliminate conflicts. Furthermore, the low inter-IOU transfer rates in a well designed multicomputer system enables a serial digital multiplex bus to be employed, affording higher reliability through simple duplication. Current technology in serial data transmission also offers virtually error-free performance at 1 MHz bit rates and using a ring bus with a round-robin I/O protocol, a low cost fiberoptic data link becomes practicable. Figure 4-3 shows a simple missile guidance and control system using one microcomputer (μc) per subsystem, serial digital input-output (SDIO) channel and fiber-optic/T^2L transmitter (T) and receiver (R) interface circuits.

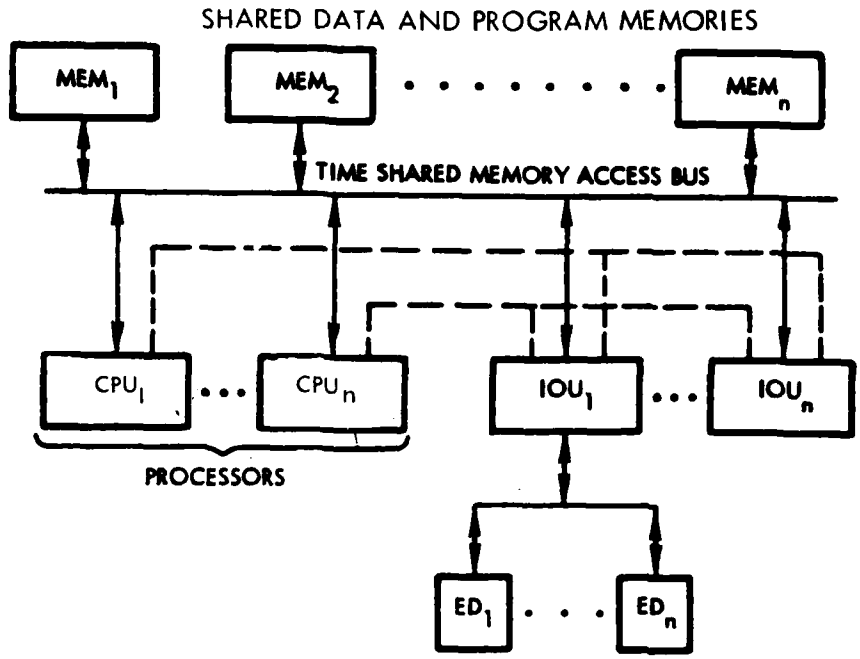


Figure 4-1 - Multiprocessor System - Single Time-Shared Memory Access Bus

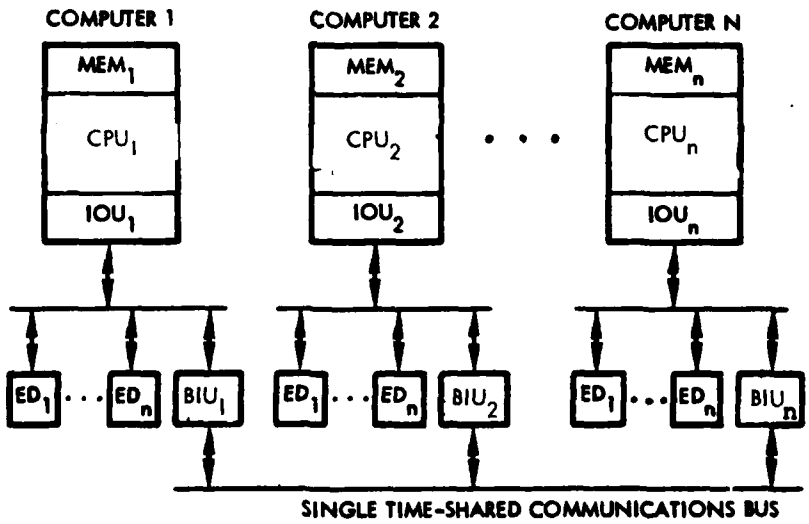


Figure 4-2 - Multicomputer System - Single Time-Shared Communications Bus

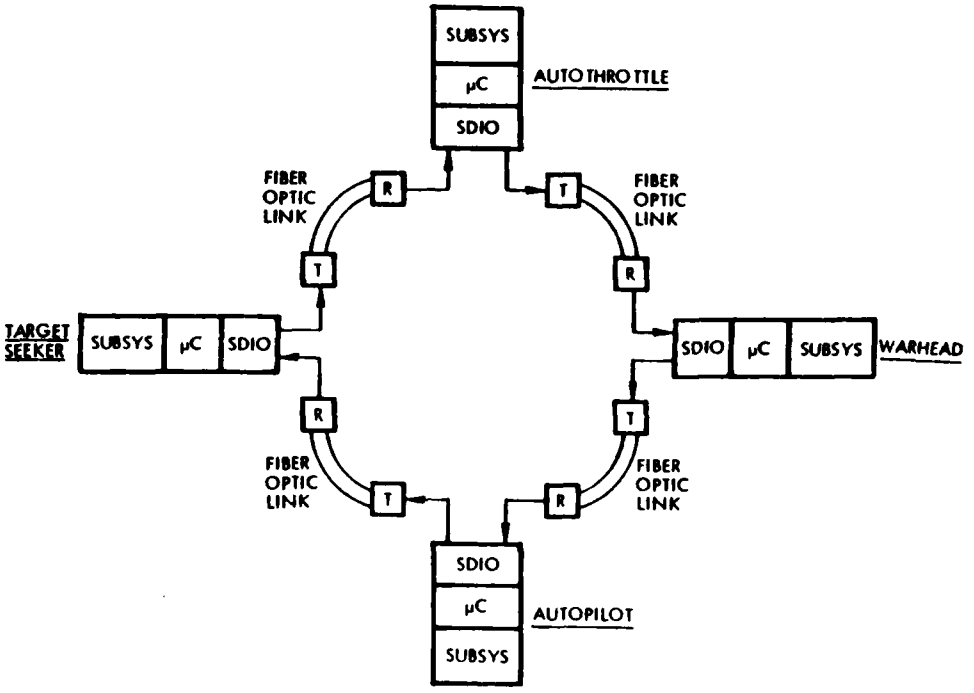


Figure 4-3 - Multicomputer System - Fiber-Optic Coupled Round-Robin Serial-Digital I/O

4.1.2 Multiple Bus

This form of communication within a computer system is more commonly encountered in multiprocessor systems to overcome the speed limitations of the single, party-line bus, thus trading off simplicity for increased speed, size, weight, cost and complexity. Figure 4-4 shows a typical multiple bus multiprocessor which has been with us for well over a decade. Each memory user (processors and IOUs) has a separate bus to access any memory bank. Conflicts in accessing the same memory are resolved in each memory bank by a multiplexer (MUX) with priority logic. Optimum speed is achieved when processors can use separate, dedicated memory banks for their respective instruction and operand accesses coupled with infrequent accesses to shared data bases and similarly infrequent DMA I/O transfers. In earlier systems using destructive readout (DRO) core memories with relatively long data transfer cycles, it was possible to access the stored data before the completion of the full memory cycle, thereby enabling the partial overlapping of instruction and operand fetch cycles when the latter were stored in separate memory units. Such fine tuning techniques are neither possible nor worthwhile

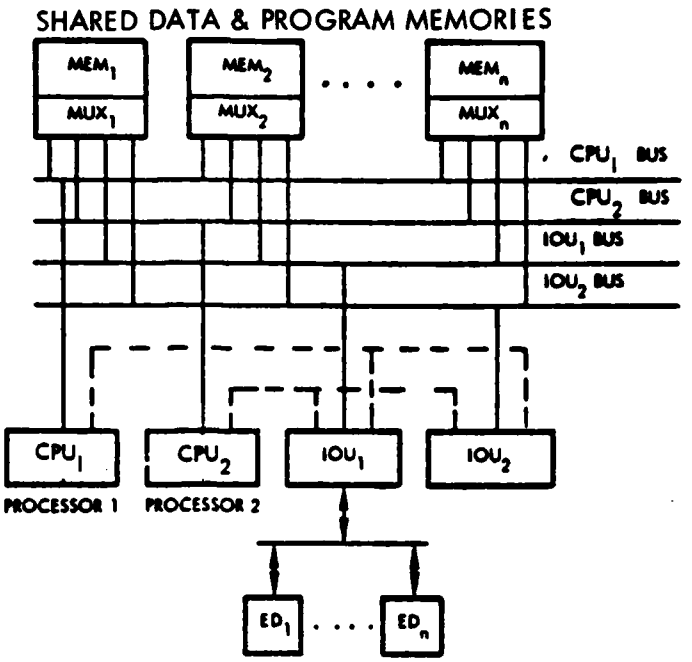


Figure 4-4 - Multiprocessor System - Multiple Memory Access Bus

with today's high-speed semiconductor memories. Under these most favorable conditions, virtually a multicomputer operating mode, throughput for a dual microprocessor system approaches twice that of a uniprocessor.

A multicomputer system employing independent I/O busses is shown in Figure 4-5. Such a system requires multiplexed, direct-memory-access (DMA) IOUs.

4.1.3 Cross-Point Switch

This form of communication/coupling between computer elements or computers was first described by H.A. Keit in 1960 and formed the essence of what was termed the "Polymorphic" concept (Reference R-35 and R-36) i.e., a system having "many shapes". One of the major objectives was to decentralize system control by making a passive switch the central element instead of a single processor. Figure 4-6 shows a multiprocessor configuration employing the polymorphic principle which was used in a high availability tactical air defense system (Reference R-37). Figure 4-7 is an example of a multicomputer version. With reliable solid-state switches these systems

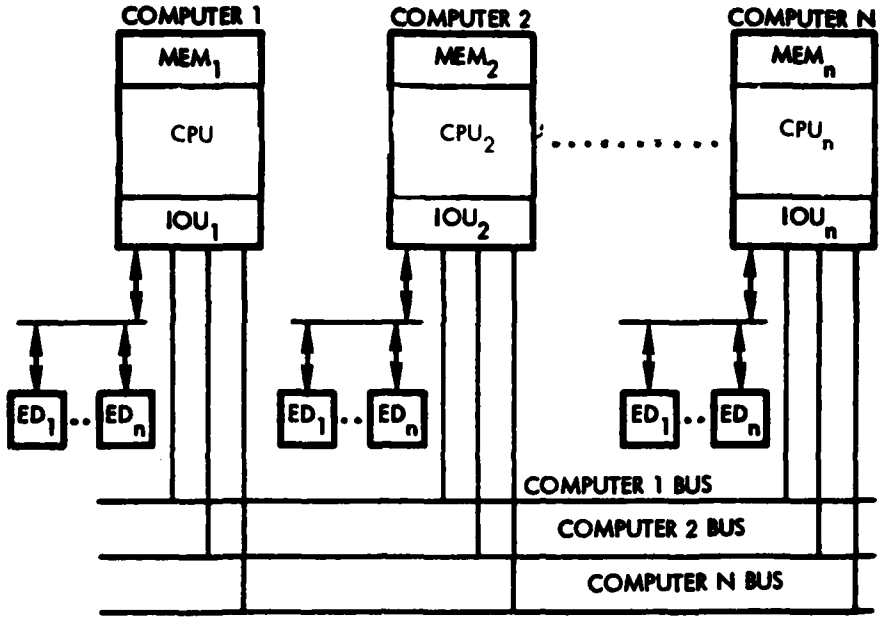


Figure 4-5 - Multicomputer System - Multiple I/O Communications Bus

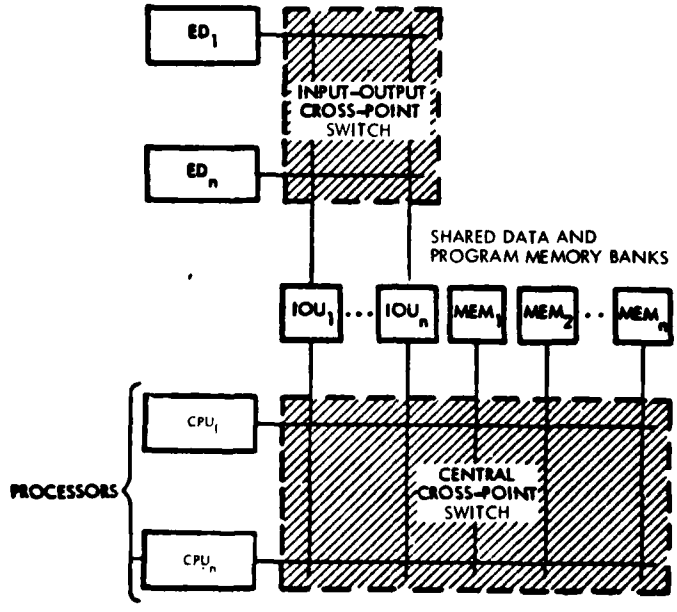


Figure 4-6 - High-Availability Multiprocessor System - Cross-Point Switch Communications

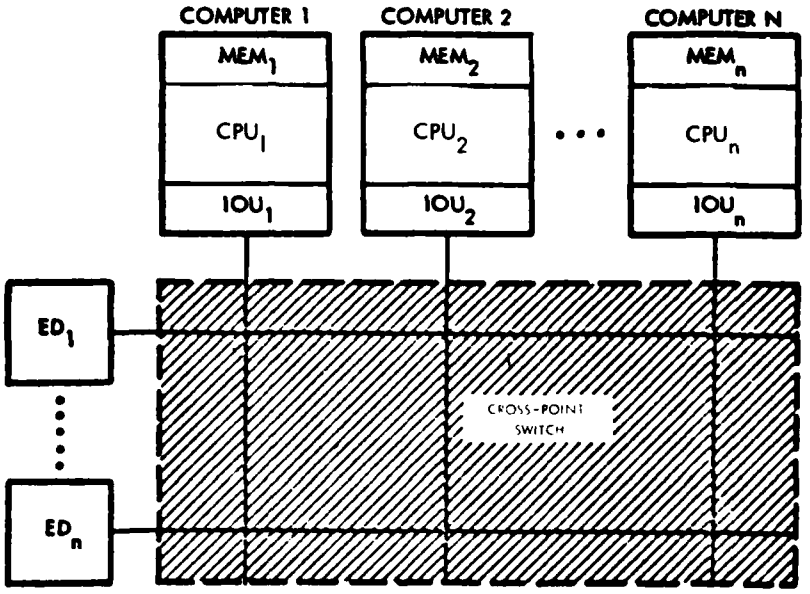


Figure 4-7 - Multicomputer System - Cross-Point Switch I/O Communications

provide high availability, flexibility and growth capabilities, and although of necessity confined to ground systems in the past, due to the size and weight of available hardware, a polymorphic multi-micro computer system with serial communications and LSI switching becomes a viable candidate for tactical avionic and missile systems (Figure 4-8).

4.1.4 Array Processor Systems

Array processors achieve high throughput for a limited range of tasks, thereby trading general-purpose features for speed. On one of the first forms of array processor the register arithmetic and logic unit (RALU) of the uniprocessor was effectively replaced by a matrix or array of processing units, each interconnected to its neighbor, and the whole designed to achieve high throughput for mesh oriented problems using a common instruction stream.

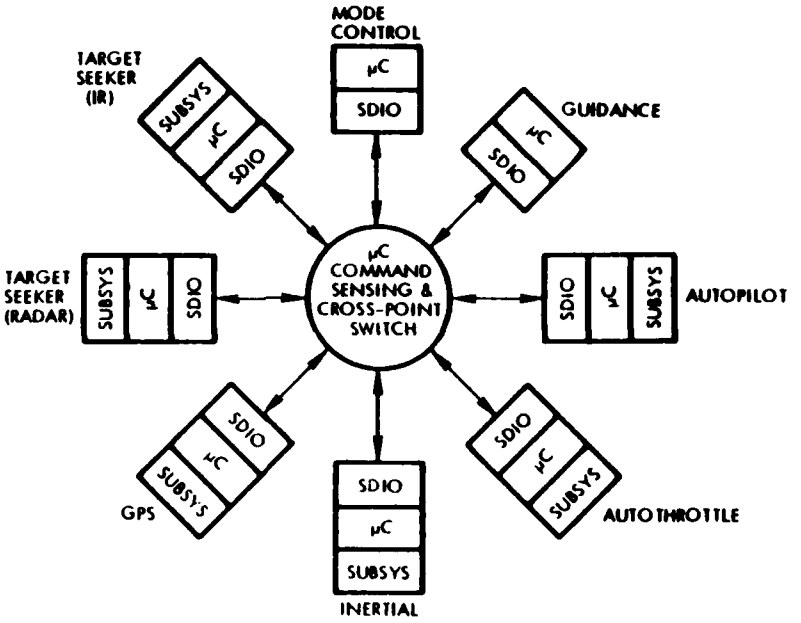
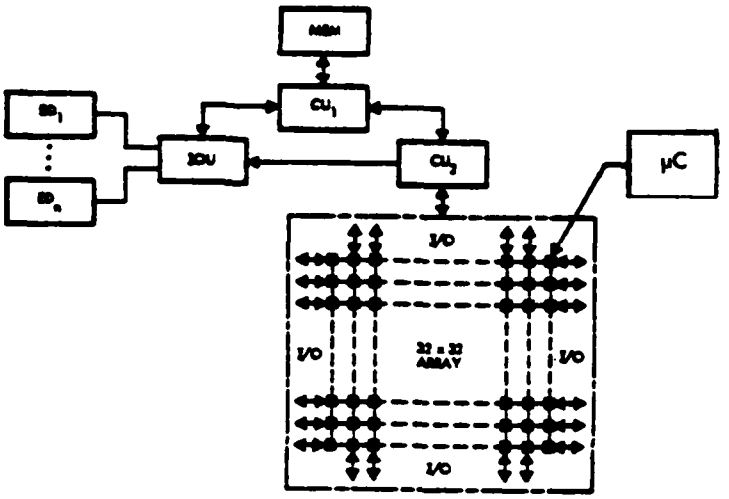


Figure 4-8 - Tactical Multimicrocomputer System - Serial Cross-Point Switch I/O Communications

Speed improvement of array processors over the uniprocessor can be a factor equal to the number of processing units in the array provided that they are continuously active. The availability of such systems for tactical applications is degraded by system complexity and the single common source of instructions. The latter deficiency however could be overcome with autonomous processing units.

4.1.4.1 Single Integrated Array

The SOLOMON II (Reference R-38) provides a good example of an array processor system using a single integrated array (Figure 4-9). The processing units in the array are virtually small microcomputers, each incorporating a 128 x 32-bit memory.



Legend CU - Control Unit

Figure 4-9 - Array Processor System - Single Integrated Array

4.1.4.2 Multiple External Array

The early ILLIAC IV (Reference R-39) shown in Figure 4-10 employed four arrays and four control units, thus significantly improving systems availability and flexibility by eliminating the dependence upon a single control unit (CU) and instruction stream. The master executive is resident in an external host computer which impacts upon the overall system reliability. The ILLIAC IV array processing units were again effectively microcomputers, each with a high-speed 2K x 64-bit memory and parallel arithmetic and logical unit. Figure 4-11 illustrates a possible 4 x 4 microcomputer array.

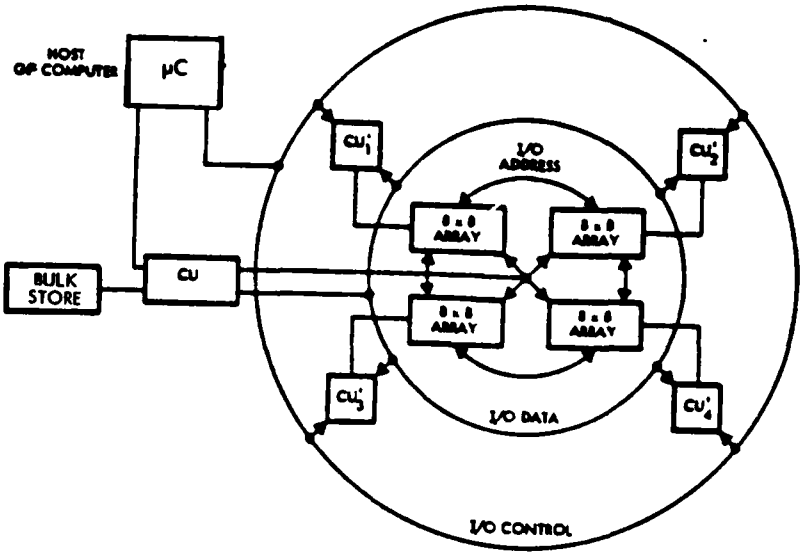


Figure 4-10 - Array Processor System - Multiple External Array

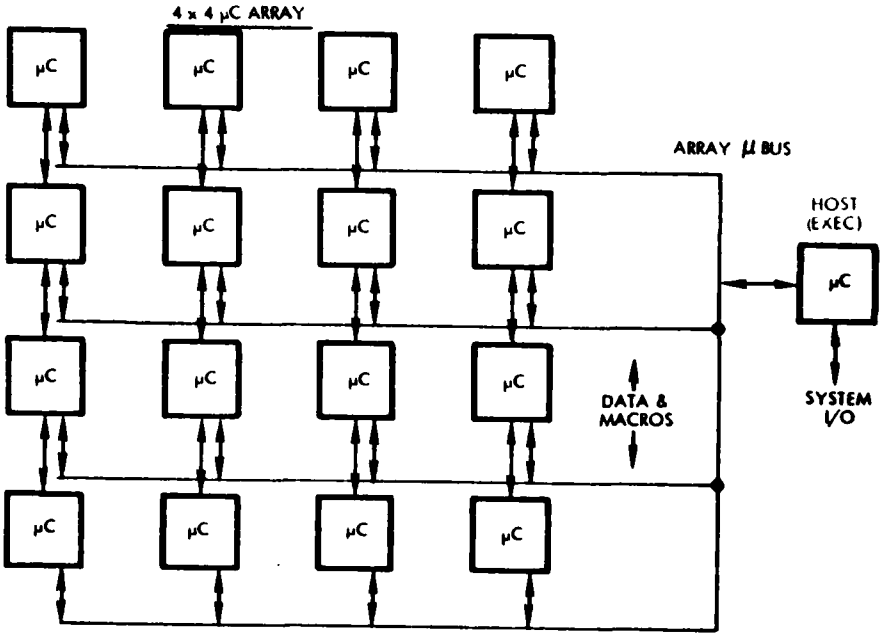


Figure 4-11 - Array Microcomputer System - Single External 4 x 4 Array

4.1.5 Associative Processor Systems

In common with the array processor, the associative processor (Reference R-40) achieves high throughput by executing a specific class of functions which, instead of suiting an array of processing units, are suited to the use of a content addressable memory. Tasks in this category typically involve the correlation of many data points with a common reference point, e.g., target tracking. Figure 4-12 illustrates an associative processor system, where the single RALU in a uniprocessor is effectively replaced by a stack of processing units, each incorporating a serial ALU, "memory", some form of autonomous control (CU) and input-output circuits (IOU) - or in other words a microcomputer. The "memory" in each processing unit is normally considered as one long word (128 or 256 bits) divided into fields, each containing a specific parameter pertinent to the single item stored, e.g., range, azimuth and elevation of a target.

Availability of such a system is again impaired by a common instruction stream, which feeds the associative memory, and the overall uniprocessor architecture. Speed is unsurpassed for correlation type tasks in a multitarget environment, and is unaffected by the number of targets within the storage capacity.

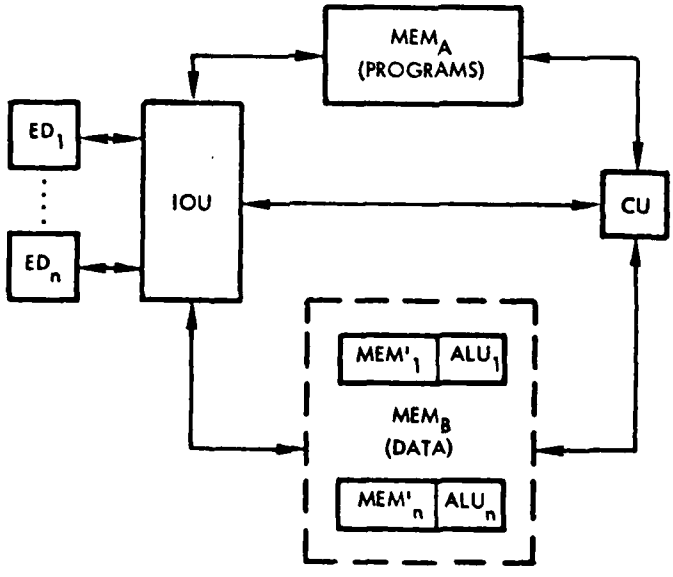


Figure 4-12 - Associative Processor System

4.1.6 "Hybrid" Processor Systems

In an effort to achieve high speed for all or many different classes of problems within a central computer system, systems have been configured to handle all the tasks encountered in tactical air defense and avionic systems. In the two configurations reviewed, a distinction is made between tasks amenable to highly parallel processing, and the remaining irregular tasks which are better suited to the traditional sequential method employed in GP uniprocessors.

4.1.6.1 Dual-Bus External Ensemble

One configuration of a high-speed multi-function processor is shown in Figure 4-13. This form of processing system (Reference R-41) employs an "ensemble" rather than an array of processing units as an adjunct to a GP computer which performs the irregular sequential-oriented tasks and furnishes instructions to the ensemble for tasks suited to parallel processing. A common global control unit interfaces with the GP computer and the radar subsystem and furnishes operands and microinstructions to all

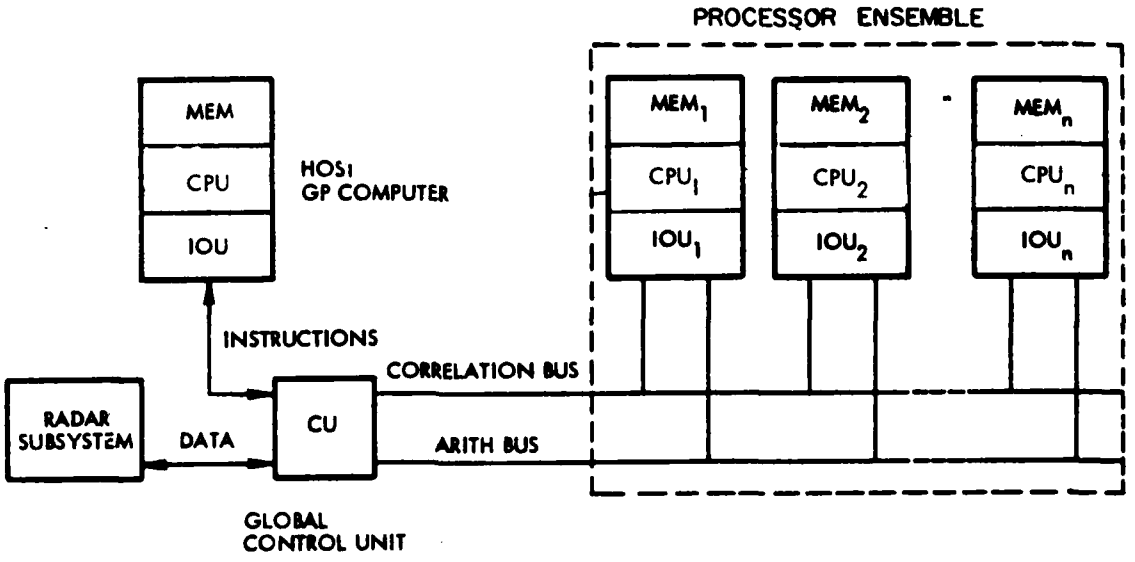


Figure 4-13 - "Hybrid" Processor Systems - Dual Bus External Ensemble, (PEPE)

processing units via a dual-bus system, one dedicated to correlation tasks, the other arithmetic. Each processing unit is again the equivalent of a microcomputer with a 512 x 32-bit word memory. Virtually any number of targets can be processed simply by adding more processing units and without any apparent loss in throughput. Availability, however, is again jeopardized by the dependence on a single GP computer and global control unit.

4.1.6.2 Multiple-Bus Integrated Ensemble

An integrated approach to the ensemble of processors is exemplified in the system shown in Figure 4-14. In this system (Reference R-42), the host computer is eliminated and the single RALU in the uniprocessor configuration is again effectively replaced by an ensemble of sequential uniprocessors and one special-purpose processor which incorporates a FFT processor, associative processor and pseudo-associative memory. A separate bus is provided for access to the bulk storage, main store and master executive control (shared bus), and the IOU. These communications busses are further augmented by an interprocessor bus and direct links between the special-purpose processor and the bulk store matrix providing EDs with direct access to the sequential uniprocessors. Each uniprocessor contains a 2-4K word high-speed memory to store and process large routines. These routines or program modules are transferred from the main store as "burst" transfers under the direction of the master executive control (MEC), the objective being to reduce activity and conflicts on the bus system compared to conventional multiprocessor systems. The ensemble is in many respects a multibus, multicomputer system of Figure 4-5 without the permanent dedication of computers to specific sets of tasks, except in the case of the MEC which is virtually a host computer. Availability of this system would depend on the duplication of the MEC and other critical programs, for immunity against memory failures, together with a duplicate special-purpose processor. The complexity of the parallel multi-bus communications system represents a significant deterrent to achieving true high availability.

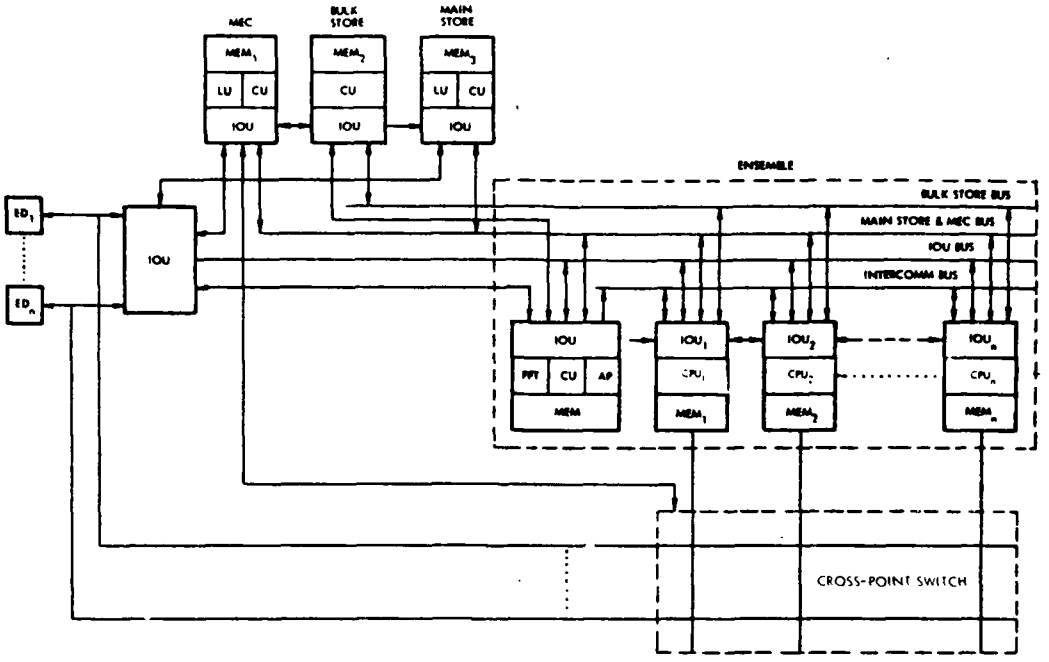


Figure 4-14 - "Hybrid" Processor Systems
Multiple-Bus Integrated Ensemble, (Original Navy AADC)

4.1.7 FFT Processor Architectures

The Cooley-Tukey fast Fourier transform (FFT) algorithm, (Reference R-43), has been widely used in high-speed real-time signal processing since its introduction in 1965. Although the FFT algorithm provided a dramatic reduction in the number of arithmetic operations required to perform frequency spectrum analysis, it nevertheless severely burdened the throughput capability of the conventional general-purpose uniprocessor. As a result of the latter deficiency, various architectures were identified (Reference 44) and developed providing gigahertz (GHz) computational rates through the use of pipelined arithmetic elements within the arithmetic unit(s) (AU) of the processor. Figure 4-15 illustrates the growth from a single sequential uniprocessor to a pipeline of AUs, (one for every major iteration in the FFT); a parallel iterative organization, and a full array. As in the previous architectures reviewed, it is conceivable to replace each MEM/AU combination with a single-chip microcomputer, thereby deriving similar factorial throughput improvements, based upon the performance of the μC.

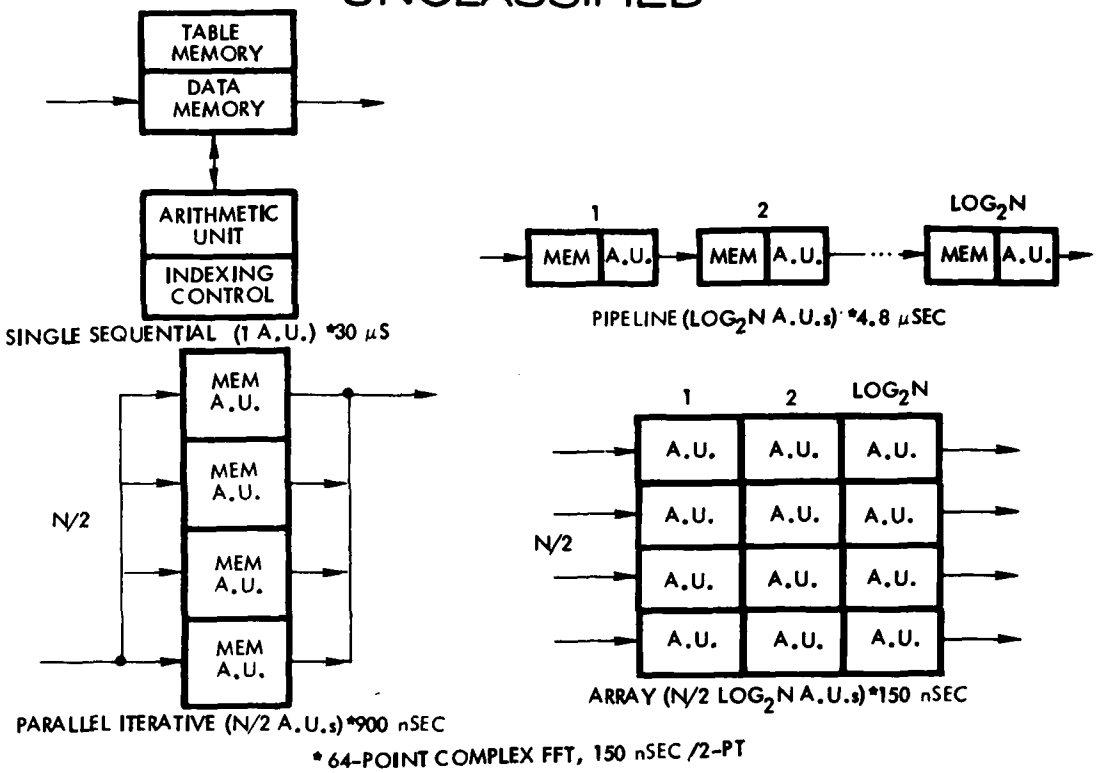


Figure 4-15 - FFT Processor Architectures

4.1.8 Conclusions

It becomes apparent that the classic multiprocessor/computer architectures reviewed are just as easily implemented, (if not more easily in a practical sense), with today's single-chip microprocessors/computers as with the smaller scale integrated circuits. Of the several types discussed, the simplicity of the single time-shared bus multiprocessor configuration suggests its viability as a candidate for the kernel element of a modular high-performance architecture. The chief deficiency of the single bus architecture lies in the shared memory, both programs and data, for each microprocessor and the resulting high incidence of conflicting memory accesses. Further, the modular software goal is defeated through shared memory. The latter deficiencies were therefore more carefully scrutinized in an effort to adapt the basic architecture to satisfy both high-throughput and modular software.

5. SUPER-FEDERATED MICROCOMPUTER SYSTEM (SFMCS)

As was stated in the preceding section, the single-bus multiprocessor architecture has only the single merit of simplicity in hardware design, but if solutions could be found to its throughput and software modularity deficiencies it could well prove to be an ideal kernal processing element for modular digital missile systems.

5.1 Modifying/Optimizing the Single-Bus Multiprocessor Architecture

Since one of the original design goals was to perform a software change through an integrated-circuit (IC), hardware change, i.e., identifying each major functional algorithm with an IC, the first obvious modification to the conventional single bus multiprocessor architecture was separation of program memory from data. Figure 5-1 illustrates the change.

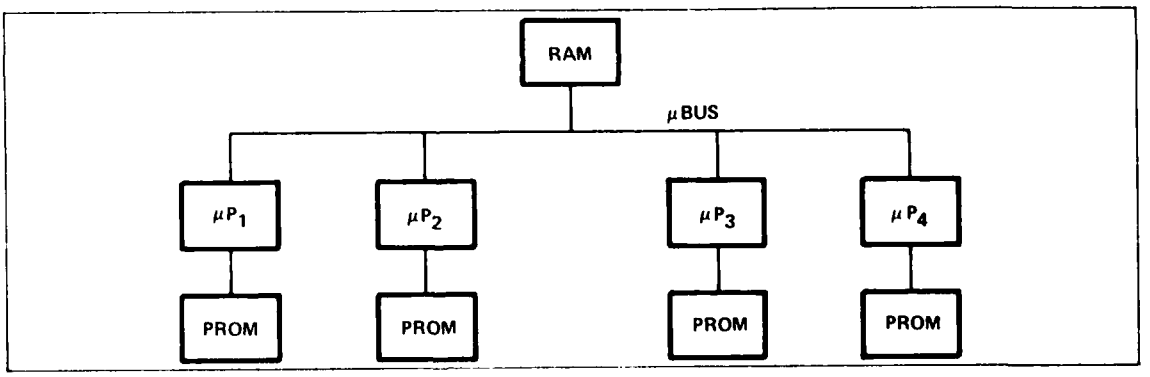


Figure 5-1 - Single Time-Shared Bus Multi-Microprocessor - Separate Program Memories

Separating programs from data has two major advantages:

1) Reduced microbus access conflicts and hence higher throughput.
2) Each major program module/algorithm is identified with a PROM IC.

The remaining shared data memory becomes a "mail box" for the transfer of computed results from one major algorithm to another, and stores the partial results of each of the separate programs when the number of operands exceeds the register storage capacity of each microprocessor.

5.1.1 Memory Mapping and Single Computer Programmability

The dedication of program memories to specific program modules, which in turn are assigned to individual microprocessors, enables each processor to be programmed as an entity rather than as part of a complex multiprocessor system. The only proviso is the establishment of a common memory address boundary for the beginning of the shared data base. This base address must exceed the highest program address used by any one of the group of microprocessors in the system, and programming can then proceed as for a single processor. Figure 5-2 illustrates this memory mapping approach for four processors.

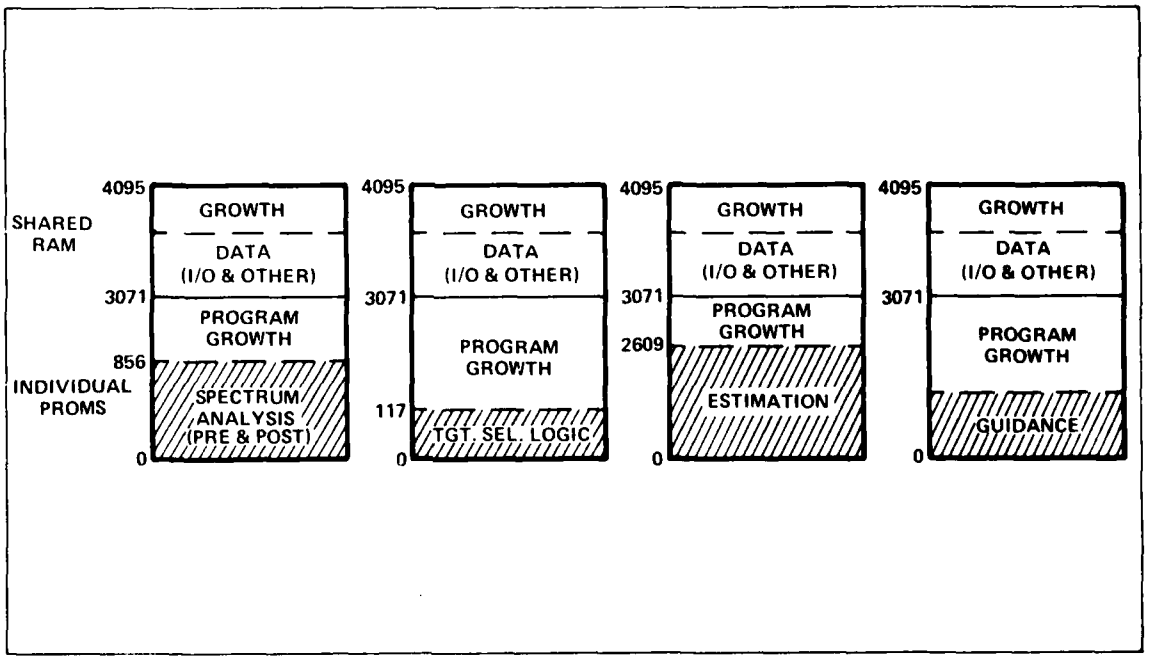


Figure 5-2 - Super-Federated Multiprocessor-Memory Mapping Class III Seeker Processing

Program growth can be accommodated for each function up to the predefined data boundary. Data can be similarly submapped for memory-mapped I/O channels as well as partial and final results. Further, if several similar super-federated groups are controlled by a host processor, a third upper level of memory space can be made available to each microprocessor for direct communication with the host processor's storage space (Figure 5-3).

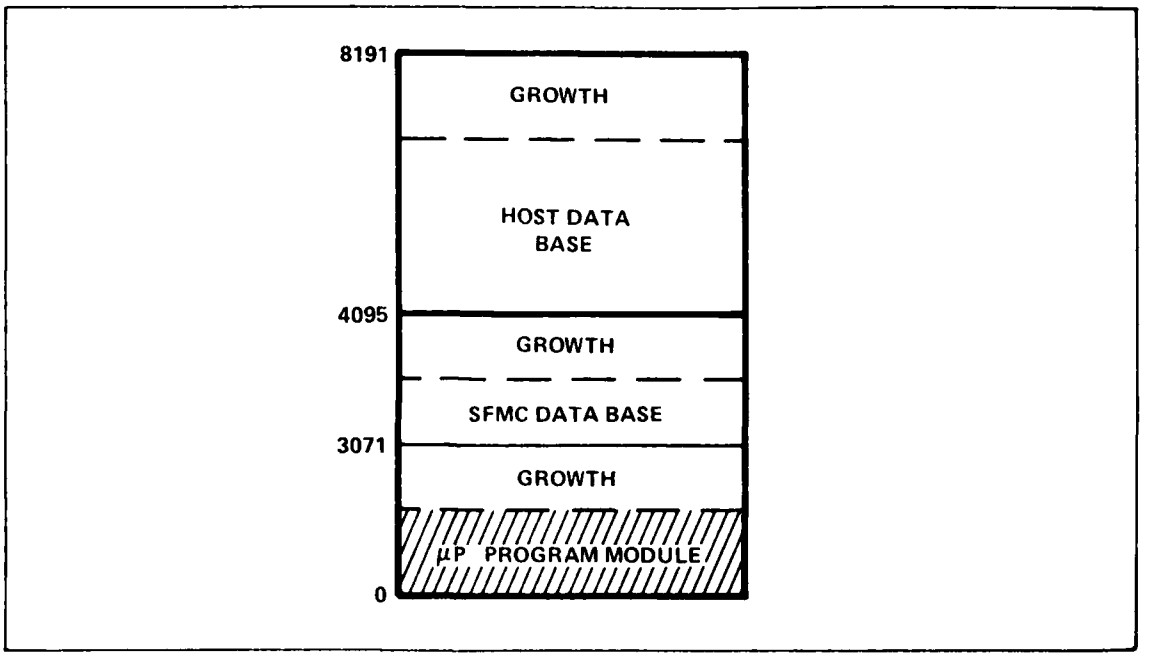


Figure 5-3 - Super Federated Multiprocessor-Extended Memory-Mapping For Host Processor

5.1.2 High Throughput Refinements

Returning to the basic single bus super-federated multiprocessor of Figure 5-1, the frequency of data memory access conflicts becomes a function of the degree of synchronism of the memory fetch cycles of each microprocessor. If all four processors were executing identical programs and were driven by the same clock waveform, then all memory fetch cycles would coincide. This situation could occur if, for example, those

processors executed the three autopilot channels, i.e., pitch, roll and yaw, respectively. At the other extreme, the dissimilarity of individual processor programs would minimize fetch cycle conflicts, as would the time skewing of clock waveforms by one memory access interval to each microprocessor. The latter phasing of clock pulses would then convert the simplistic single bus multiprocessor to a time-phased ring bus architecture Reference R-45. Such clock time phasing would eliminate memory access conflicts when identical programs are being executed by each microprocessor and could be expected to significantly reduce conflicts among dissimilar instruction sequences. The latter could then be resolved by a rotating priority scheme executed by a bus controller. Figure 5-4 illustrates the above timing for four processors, although it would be valid for more using additional clock phases.

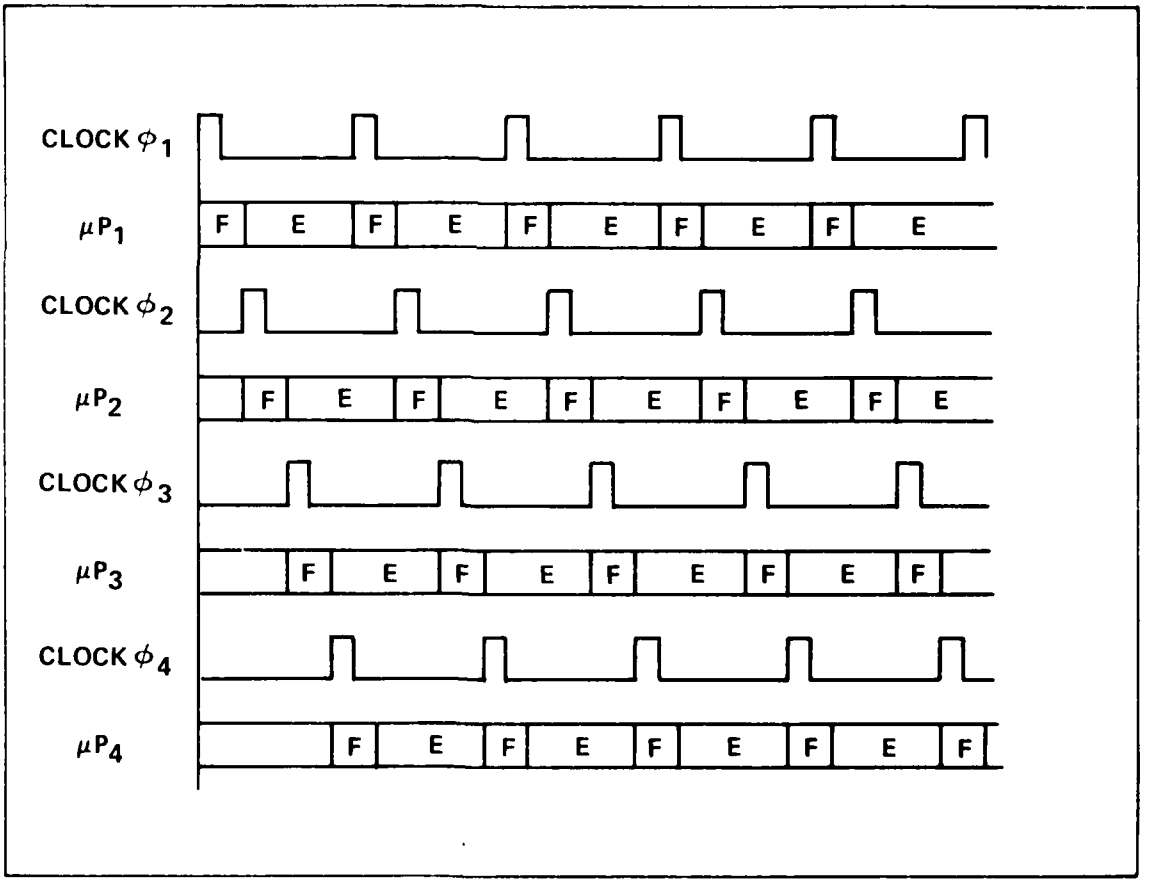


Figure 5-4 - Time-Phased Ring Bus Memory/Instruction Fetch (F) and Execute (E) Sequences Four Microprocessors Identical Instruction Streams

5.1.3 Time-Phased Ring, Practical Case

To be effective in a realistic sense the foregoing timing refinements must be applicable to commercially available microprocessors. The Intel 8086 and Zilog Z-8000 16-bit microprocessors were selected as representative of the state-of-the-art in single-chip microprocessor technology.

5.1.3.1 Intel 8086 Waveforms

Figure 5-5 shows the compatibility of the Intel 8086 timing waveforms with time-phased clocking of each microprocessor clock input. Further details of the latter timing relationships are given in Appendix A.

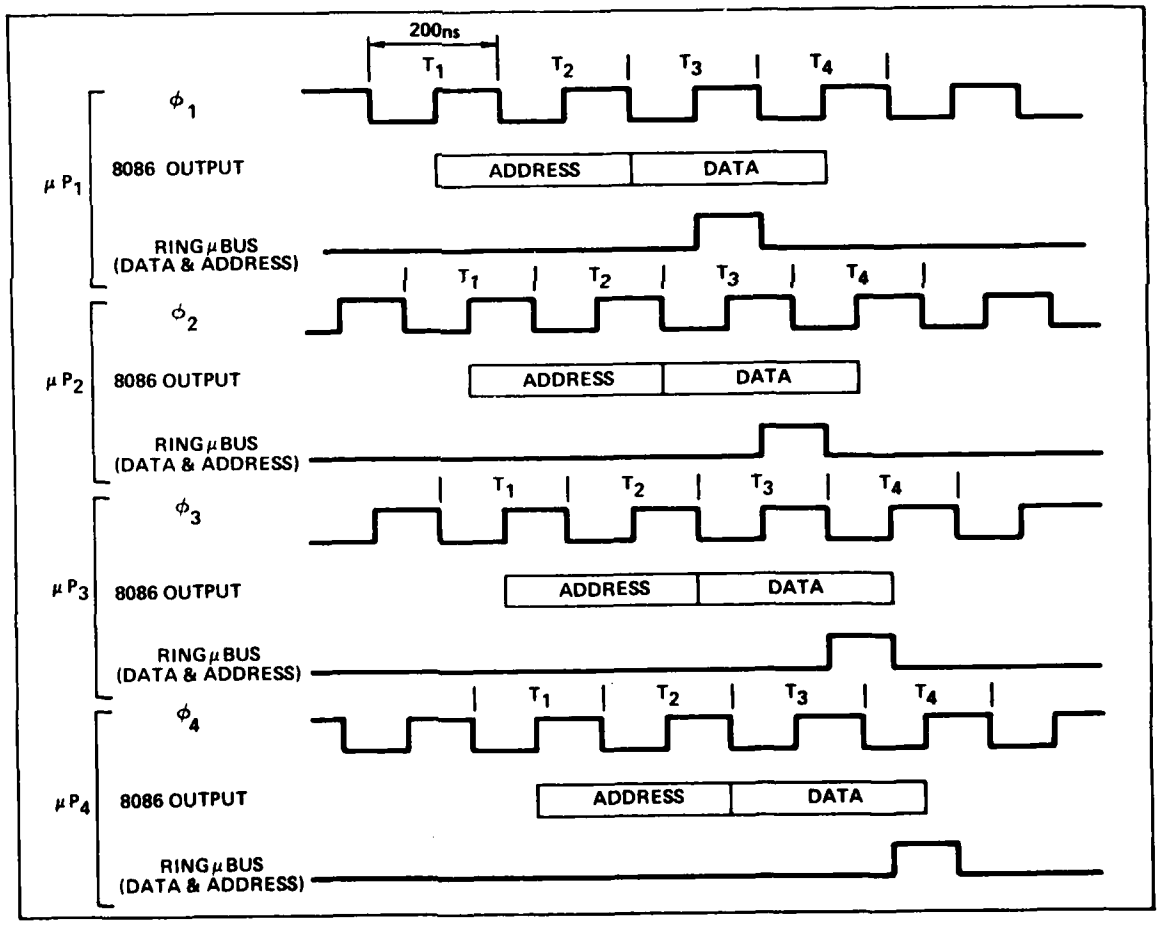


Figure 5-5 - Time-Phased Ring Bus Intel 8086 Timing Compatibility

5.1.3.2 Zilog Z-8000 Waveforms

The Zilog Z-8000 microprocessor requires a minimum of three clock cycles to output the memory address and transfer a data word to/from memory. Figure 5-6 shows the staggering of ring micro/bus accesses through the use of a four-phase clock (Appendix A).

5.2 Expanded System Architecture

Using the basic four-microprocessor module (quad) described in the previous paragraphs, an expanded system was configured using four quads and a host processor (Figure 5-7). A functional block diagram of this system is given in Appendix C.

The memory map for this system is as shown in Figure 5-3. Single computer programming simplicity is maintained and memory access conflicts are resolved by FPLA-based arbitration units, transparent to the programmer. Input-output activity is handled by memory-mapped I/O modules (ADAC and SDIO) whose data is directly accessible by any microprocessor.

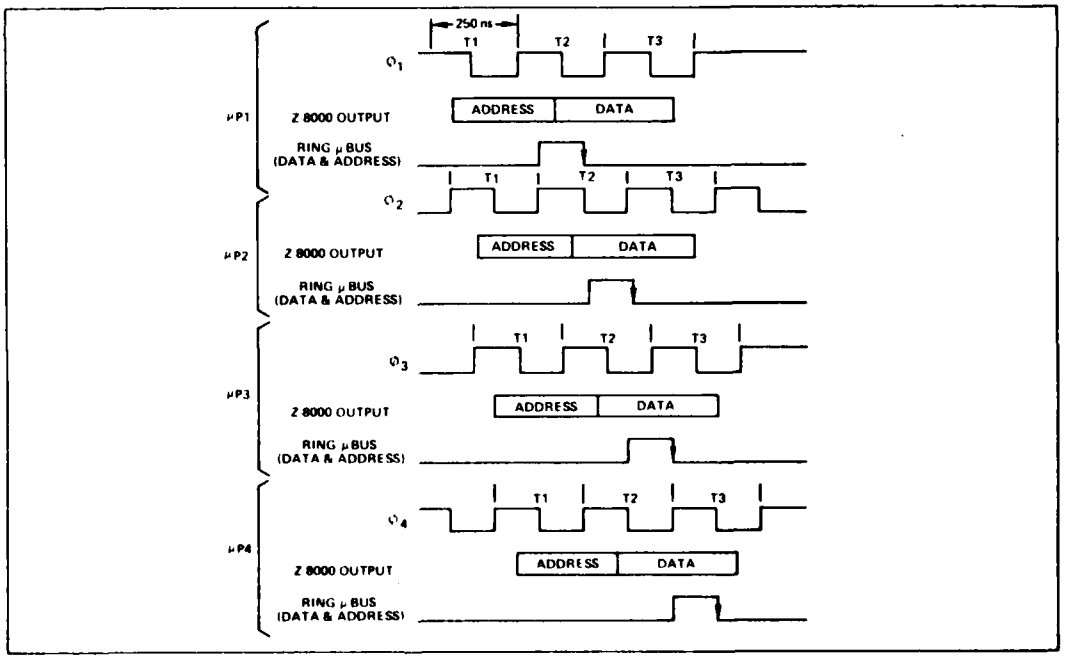


Figure 5-6 - Time-Phased Ring Bus - Zilog Z-8000 Timing Compatibility

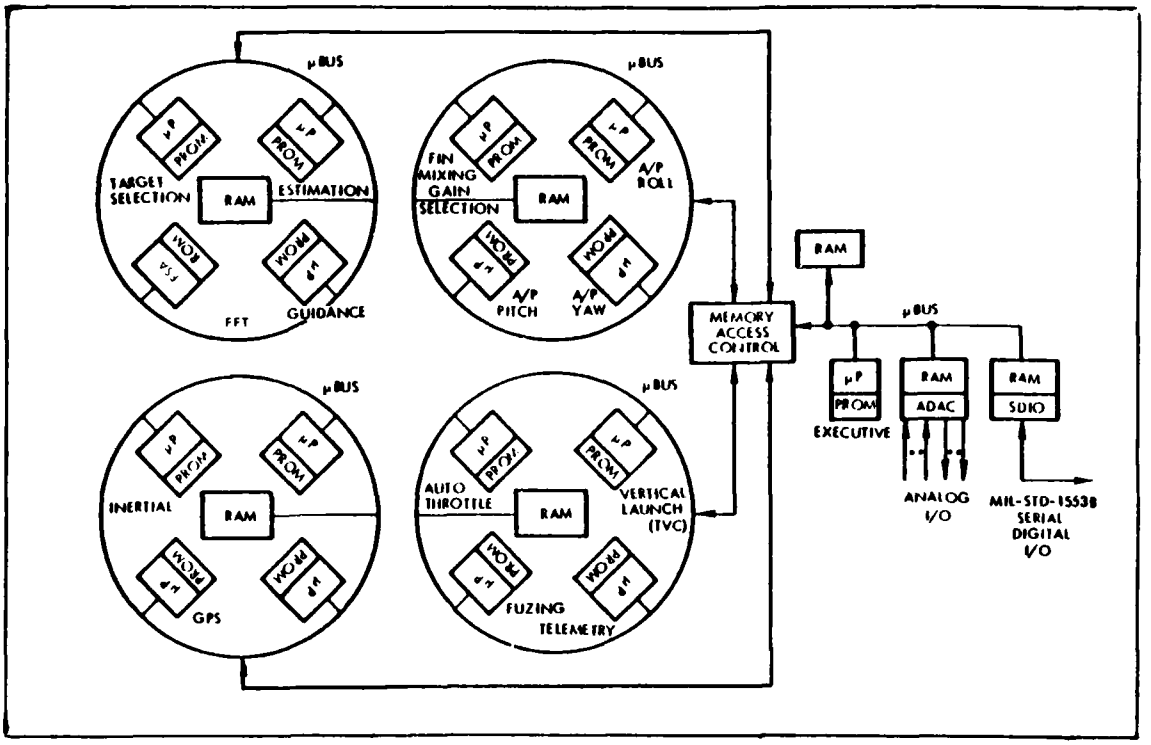


Figure 5-7 - Expanded Super-Federated Microcomputer System For Physically Centralized Applications

This architecture, using 16-bit parallel data paths between quads and host, provides high throughput, i.e. up to 10 MIPS, depending on the instruction mix, microprocessor type and the amenity of the task to be distributed among the processors. In the configuration shown for missile guidance and control each quad contains separable subfunctions of the major function. The host random access memory (RAM) provides a common mail box store for all four quads, e.g., for the transfer of "g" commands from the seeker quad to the autopilot quad at the low 10-20 Hz rate. Such a system provides super-federation of hardware and software in applications where all the processing hardware must be located in one place.

5.3 Physically Distributed Systems

Using the basic quad building block as a replacement for the bit-slice, Schottky-bipolar/CMOS-SOS processor, i.e., μ CPU-2 in the macromodular family, a physically distributed system would be as shown in Figure 5-8.

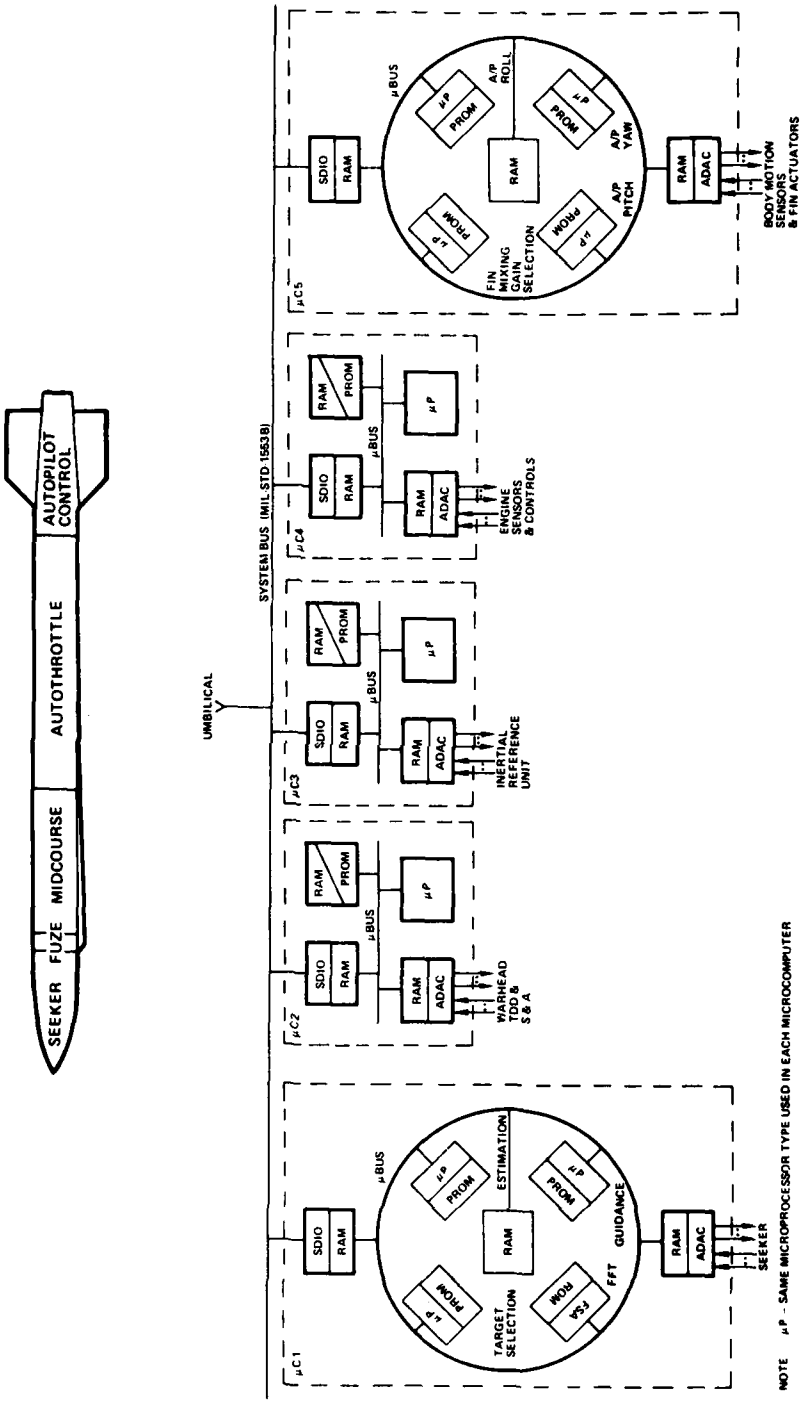


Figure 5-8 - Physically Distributed Super-Federated Microcomputer System

Similarly, growth from a low-performance federated guidance and control system such as that built for NSWC (Figure 5-9) to a high-performance Class III system could be achieved by the simple substitution of a SFMCS quad in place of the single microprocessor (Figure 5-10).

The chief difference between the expanded system of Figure 5-7 and that of Figure 5-10 is the connection of I/O modules to the quad microbus, as opposed to the host processor microbus, and the memory mapping of I/O RAMs as part of the ring RAM data base.

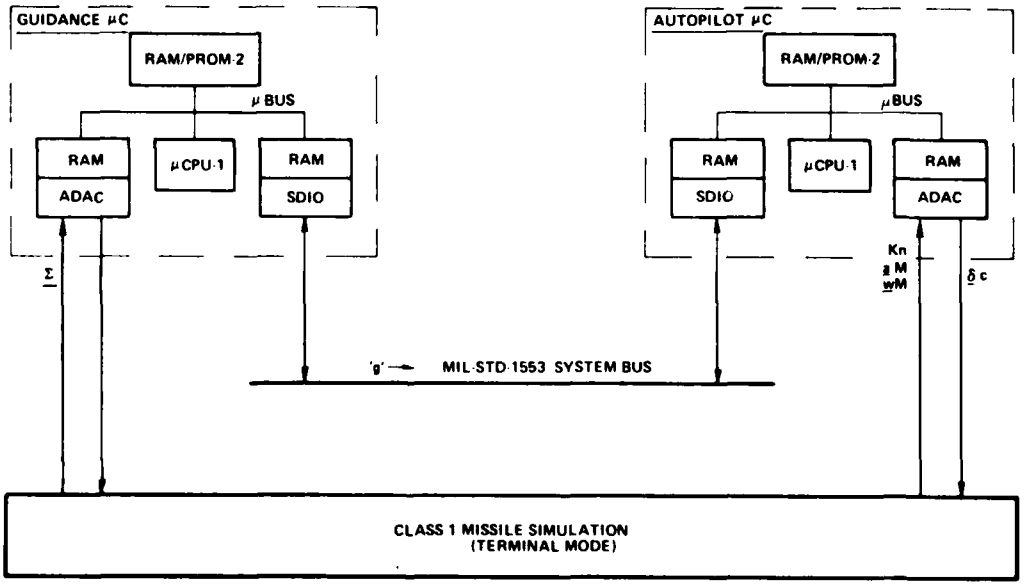


Figure 5-9 - Low-Performance Federated Microcomputer Missile Guidance and Control System (NSWC System)

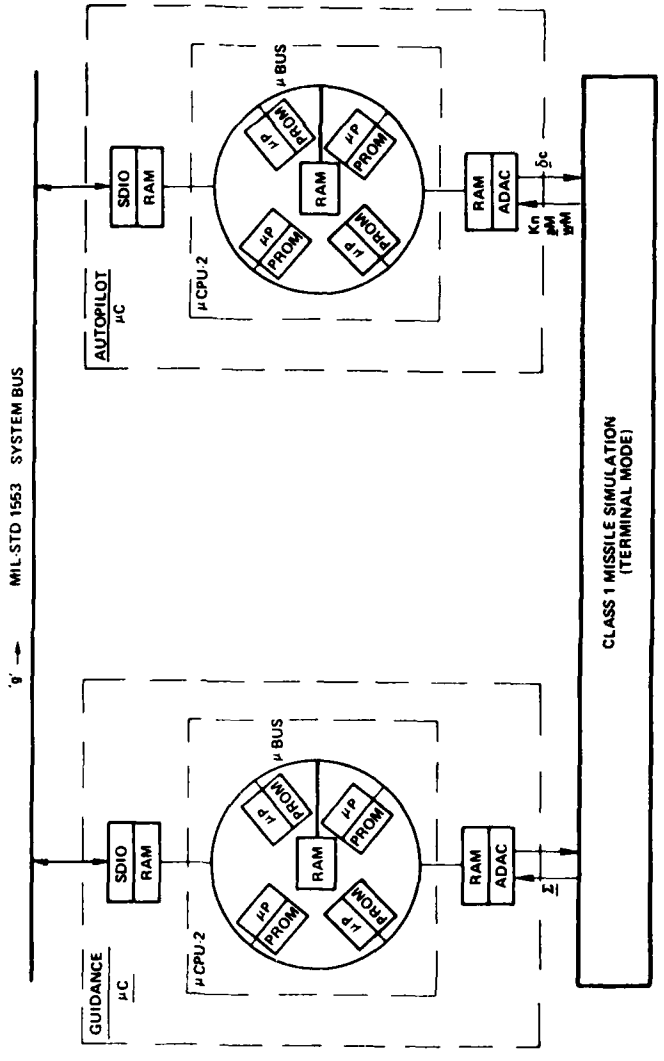


Figure 5-10 - High-Performance Super-Federated Microcomputer System For Higher Performance Missile Guidance and Control (NSWC System)

6. SFMCS SOFTWARE

As stated earlier in this report, the primary advantages of super-federated computer systems for on-board missile guidance and control are throughput and physical compatibility with the modular design requirements of a missile, both hardware and software. From a software architecture point of view, there are certain technical trade-offs which must be analyzed to achieve the primary design requirements viz:

1) High throughput
2) System extensibility
3) Minimum software development risk
4) Associative software/hardware modularity

The throughput capabilities of a super-federated system will be affected by the following factors:

- Distribution of application programs throughout system memory
- Distribution of application programs among the processors within the federated system
- Distribution of data throughout system memory
- Selection of control software (network executive)

Support of system extensibility is directly related to the amount of software modularity which can be supported by the super-federated system architecture and the adaptability of the control software to the changing software requirements.

Reduction of software development risk in a super-federated system is related to:

- Use of high order languages which support concurrent processing
- Independence of application program design from system architecture
- Strict adherence to software modularity guidelines.

The close interrelationship of software and hardware modularity is viewed as a key ingredient in the efficient management of software development and maintenance throughout the systems life cycle.

Hence, it is apparent that the successful use of a super-federated system is influenced by the distribution of programs and data throughout the system, modularity, selection of control software, and the use of high order languages to support concurrent processing.

6.1 Distribution of Programs and Data

The distribution of programs among the various processors and system memory will directly impact system throughput. If common memory were used to hold all programs, it is obvious that the SFMCSs throughput, or any tightly coupled distributed system's throughput, would be reduced to the availability of that memory to the various processors. In general, the missile environment does not lend itself to the use of "large amounts of code sharing" by various software functions. For example, the code of an autopilot is distinct from the code of a tracking filter, with the possible exception of a service routine (possibly a matrix manipulation service). For this reason the optimum layout of code throughout system memory is through the use of a local processor memory in which the processor does not compete for use of the memory. The SFMCS's architecture permits maximum throughput by its use of local memories (i.e. dedicated PROMs). The disadvantage of the local memory design is a slight increase in total memory due to the possible duplication of service routines.

We traditionally accept target tracking logic as separate from a guidance law or an autopilot with limited data interfaces. For this reason, we partition the functions of missile control software among the various processors within the system. The SFMCS offers the ability to partition the functions within a quad or within several quads.

But the distribution of a function (set of application programs) within a set of processors is complicated by our limited ability to visualize certain software functions as parallel (i.e., concurrent) processes. Traditionally, a guidance law or a tracking filter is presented as a serial process as illustrated in Figure 6-1. The serial process is typified by the use of a parameter calculated in the previous statement. The challenge is to partition a function so as to maximize throughput. Figure 6-2 illustrates the type of partitioning which must be used to distribute the example in Figure 6-1.

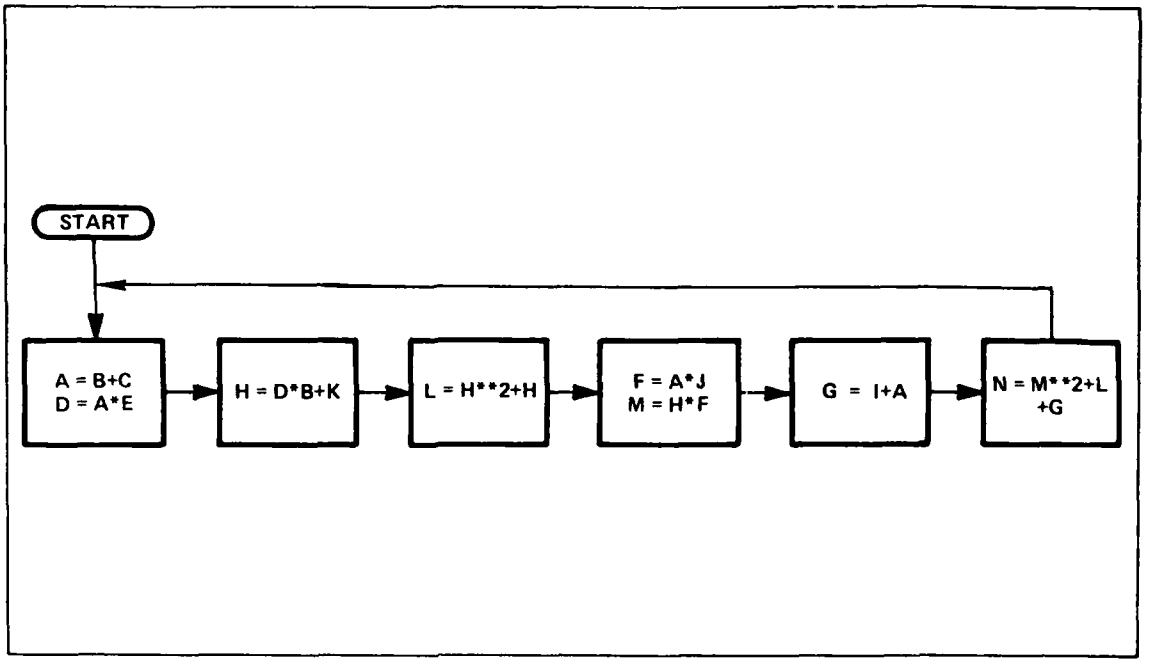


Figure 6-1 - Typical Serial Process

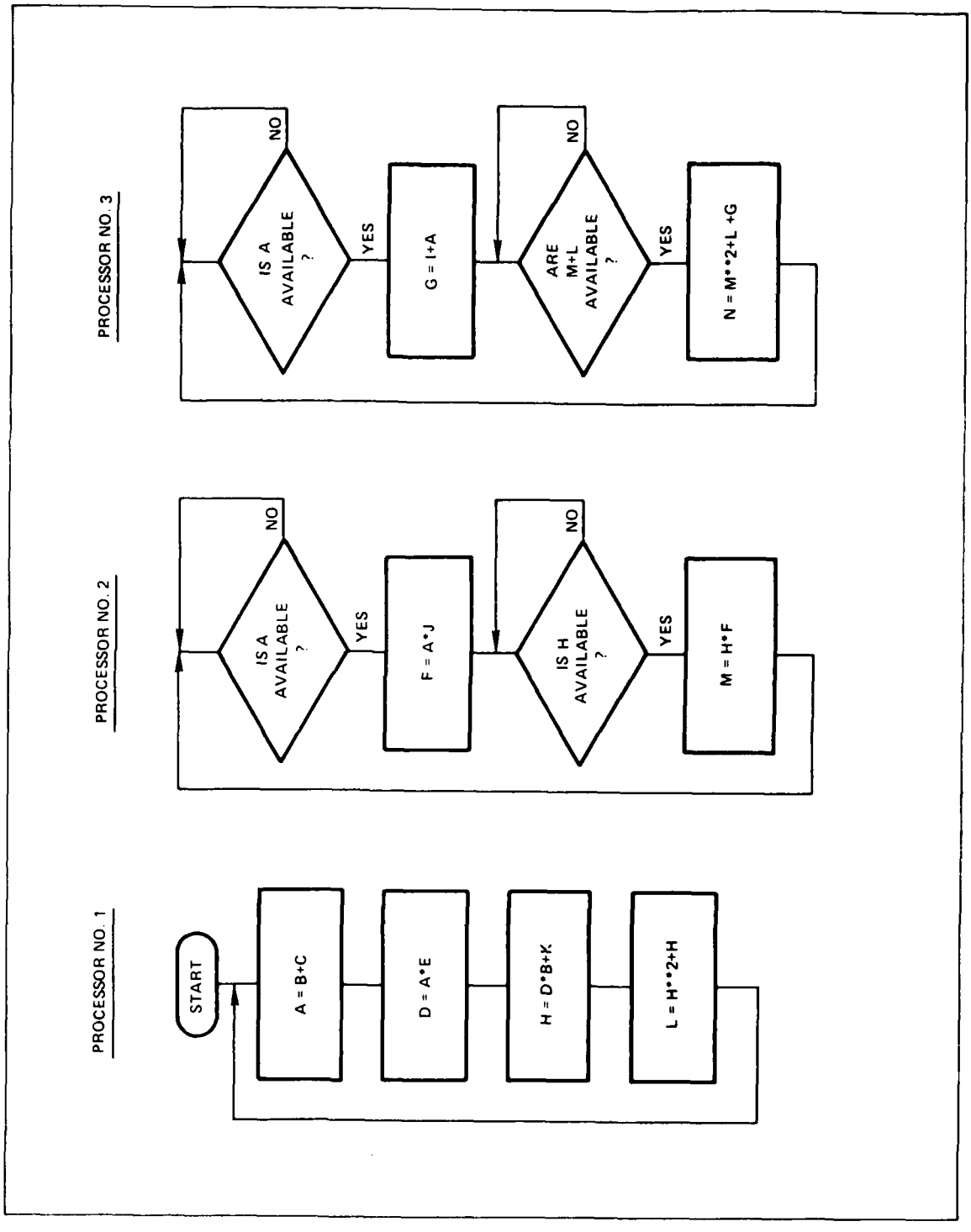


Figure 6-2 - Distribution of Figure 6-1 Statements

The development of software tools to assist in this partitioning activity is a necessity, if the application of the SFMCS or any tightly coupled computer system is to mature to wide application in a real-time environment as throughput-demanding and complex as missile guidance.

The partitioning of data will impact system throughput if the data is localized in such a manner as to cause the processors in the system to wait for access to a particular memory unit. In addition, throughput is adversely affected if a processor must wait for data to be available before it can continue processing (as illustrated in Figure 6-2). (Data consistency is obviously an important consideration in any partitioning scheme.)

The SFMCS permits the distribution of data among the various levels of memory to minimize memory conflict. If we take the example in Figure 6-2, we would distribute the parameters as illustrated in Figure 6-3.

6.2 Modular Software

This section summarizes the intrinsic characteristics of modular software as they tend to impact on a super-federated microcomputer system architecture. (References R-46 and R-47) Desirable modularity features are as follows:

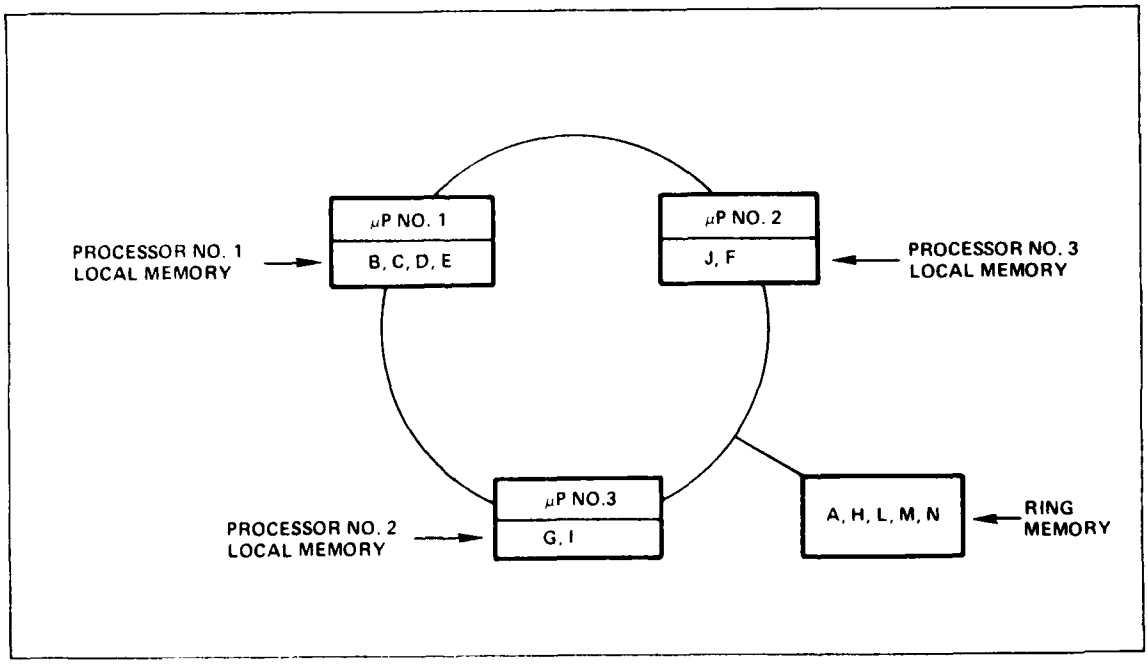


Figure 6-3 - Distribution of Data

- May be executed as an independent set of codes given proper drive input.
- The software module is not required to obtain its input from the system (i.e., do I/O operations), rather the system supplies the module with data.
- The code may be transported from system to system with no changes in either the code or the methodology of linkage method.
- The code should be machine and system independent.
- The execution of the code should be system thread independent.
- The module should be able to be replaced with a stub whereby a command response is a given rather than a calculated one.
- It should be identified as a set of logic with real world boundaries, i.e., a PROM integrated circuit.
- If the module is an I/O driver, its methodology linkage to the system should be independent of the specific I/O device.
- A modular system is one in which units of standard size, design, etc., can be arranged or fitted together in a variety of ways (implementation independence).
- The operating system needs to be distributed and not centralized. Centralization and modularity are diametrically opposite concepts.
- A centralized executive is one which is highly dependent on the implementation.
- A hierarchy of distributed control enhances the modularity concept.
- The concept of local autonomy should be used in partitioning the distributed structure.
- A centralized operating system is based on the concept of a sole source issuing directives to subordinate tasks, posting requests for and dispatching tasks for execution. The program threading is generally controlled by the supervisor. In distributed control tasks run asynchronously and do not need to be explicitly dispatched. The local autonomous control program has sufficient delegated control to determine whether a task should be executed or other subordinate tasks dispatched.

For example, consider a conventional OS dispatcher function. Generally an external stimuli (interrupt) or a task complete causes the OS to schedule a task to run, the dispatcher is next called to the control point to dispatch the next task according to priority, resource availability, etc. Generally an explicit directive from the OS initiates the task.

In a distributed control task, managers have access to the task queue and determine whether or not the task can be run. There will be some duplication of software in this structure, however this is the penalty for modularity. For tasks that are truly autonomous and thread independent, they can run continuously. An example would be a continuous A/D converter driving a memory mapped I/O System. The system need not command the conversion since it is being performed continuously. If we were to elevate the level of control to an autopilot, for example, then a continuous autopilot calculation would be performed. One key to modular software is therefore the linkage mechanism.

6.2.1 Table Driven Software Modules

The communications between the operating system or real-time executive and the modular software is performed through messages prepared by the OS and deposited in a memory space common to both the OS and the modular software. This message may contain such parameters as the location of the data to be operated on, the task to be performed, explicit data fields identifying where to deposit results, a field to ascertain equipment or program status and a variety of parameters necessary to execute the task at hand. The linkage may not only contain data necessary for the execution of a single point, sequential task, but may also contain a set of instructions the OS is requesting the servicing device to perform. The preparation of the table is not restricted to the operating system but may be loaded by other processors passing data or control to the next processor in the system thread.

This method of linking programs has several "buzz words" associated with it which include: packet-directed procedures, linked control blocks, table driven software, semaphore control, task block and control program generation.

Several methods to bring a software module up to the control point and cause it to go into execution are available. These methods include:

- A direct call from the OS with argument pointers to the table directing. This may be termed synchronous execution.
- A subprogram may be polling a directive table when the OS or other programs deposit an execution directive. Execution commences. This mode of operation may be termed asynchronous.
- Direct hardware interrupt is applicable primarily in multiple CPU configurations, where a single vector interrupt is used to "wake up" an idling program and cause execution to commence.

6.2.2 Composition of Software Modules

The module is composed with two primary nodes: a computational, and a management. The subdivisions of this partitioning include the following elements:

6.2.2.1 Computational Node

1) **Functional Description** - A mathematical or algorithmic description of the processing requirement.
2) **Data Integrity** - It is the responsibility of the calling procedure to insure the data validity before invoking this procedure.
3) **Data Source/Determination** - The location of source data, placement of transitory variables and destination of the resultant data shall be specified in the procedure as a part of the calling linkages.

6.2.2.2 Management Node

The following management (local) functions are identified as:

1) Identification of any subprocedures/subroutines invoked.
2) Identification of internal data for control purposes.

3) Identification of any resources shared by the system.

4) Providing the linkage to the operating system to allocate these resources to this subprocedure.

Modular software or configuration independence requires:

- The need for functional interface definition
- Flexibility for growth
- Changes in configuration do not necessarily mean changes in code (repercussion effects)
- Ability to introduce another subsystem without disturbing the entire system
- Keep specification and top level design independent of implementation when possible.

Configuration dependence requires:

- Interdependence of elements

6.2.2.3 Definition of a Control Block (High Level) Task Switching

For each process, the software system defines a control block which represents that task to the system and through which system and process interaction is performed. It represents a place for the representation of any relationship between the process and processes that have invoked it. It provides a place for the description of events that must be completed before the task is to operate. It provides a place for pointers to other system control blocks which represent both the allocation of memory and devices to the process.

6.2.2.4 Definition of Reentrancy

A reentrant island of code is one in which no changes occur as the result of execution at any time. All parameters are passed to it and to all intermediate values that it develops. All results, etc., are considered to be objects external to the code itself. (NOTE: Common system subprograms and subroutines should be reentrable.)

6.3 SFMCS Control Software

While the individual microprocessors of the SFMCS are standard industry devices supplied with conventional support software, and each processor can be programmed as an entity using a predefined memory map, nevertheless the expanded architecture of Figure 5-using a host processor presents various options for system control.

Possible methods of software control of the Super-Federated Microprocessor System (SFMCS) are similar to those available to multiprocessors and tightly coupled distributed systems. The methods available are:

1) Master/Slave:
2) Floating Executive (Decentralized) Polling
3) Floating Executive with Multiprogramming.

6.3.1 Master/Slave

The master/slave has a master processor in the federated system which controls the processing carried out by the other (slave) processors in the system. Essentially the scheduling and dispatching of tasks within the SFMCS is carried out by the master processor. The master/slave is a hierarchical configuration with two levels of hierarchy in which the host (master) performs all of the task scheduling and dispatching for the satellite (slave) processors.

It should be noted that in general the master/slave relationship is independent of the method of communications among the processors. The communications between processors in the SFMCS can be implemented through memory (the slave polls a memory location for control information from the master) or through a positive control signal (e.g., an interrupt) between processors.

The cascading memory feature (the ability of processors within the SFMCS to directly access various levels of memory) of the SFMCS permits easy implementation of a memory polling scheme. The control software would be located in the system memory (Figure 6-4).

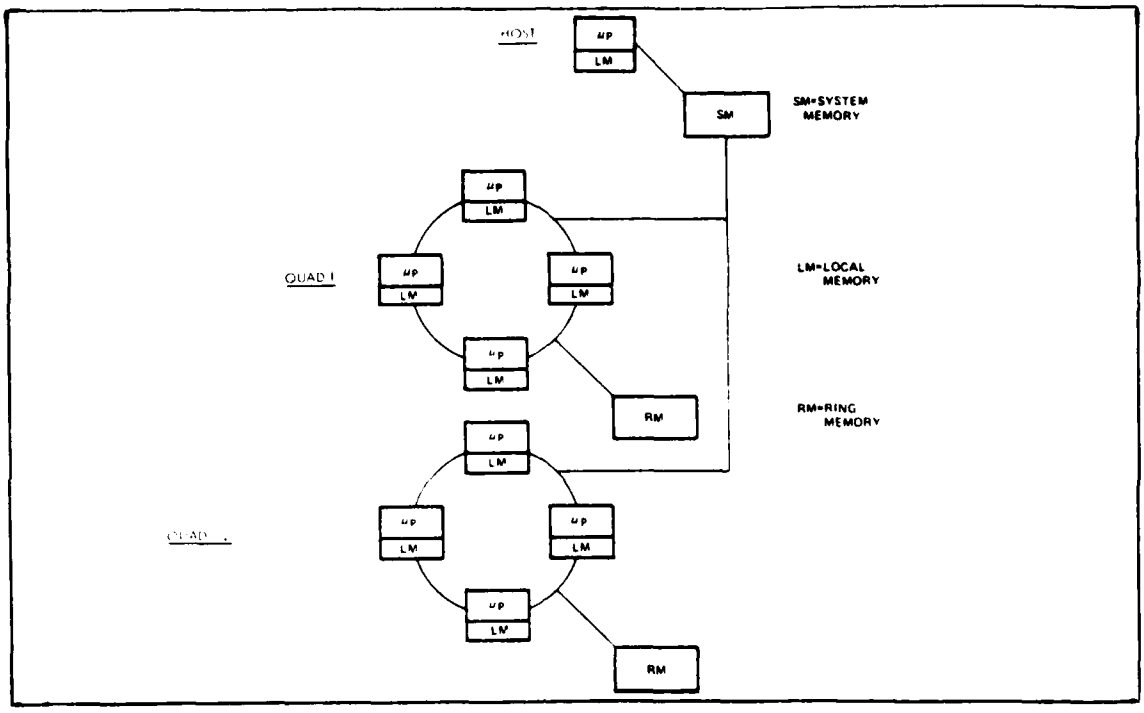


Figure 6-4 - SFMCS, Host and Dual-Quad Configuration

The polling scheme has two main drawbacks:

- Changing of a slave processor task cannot be accomplished after the slave processor has started.
- Use of system memory becomes extremely high when slave processors are polling.

Figure 6-5 is a simplified illustration of the polling control software required in a slave processor.

The use of a control signal between processors would eliminate the drawbacks associated with the polling scheme. The use of a control signal between processors would increase the complexity of both the software and the hardware in the SFMCS. Figure 6-6 is a simplified illustration of a slave processor controlled by a control signal.

The selection of a master/slave control system for the SFMCS implies that the application requires the assignment of multiple task to the individual processors and further implies that the task assignments must be synchronized.

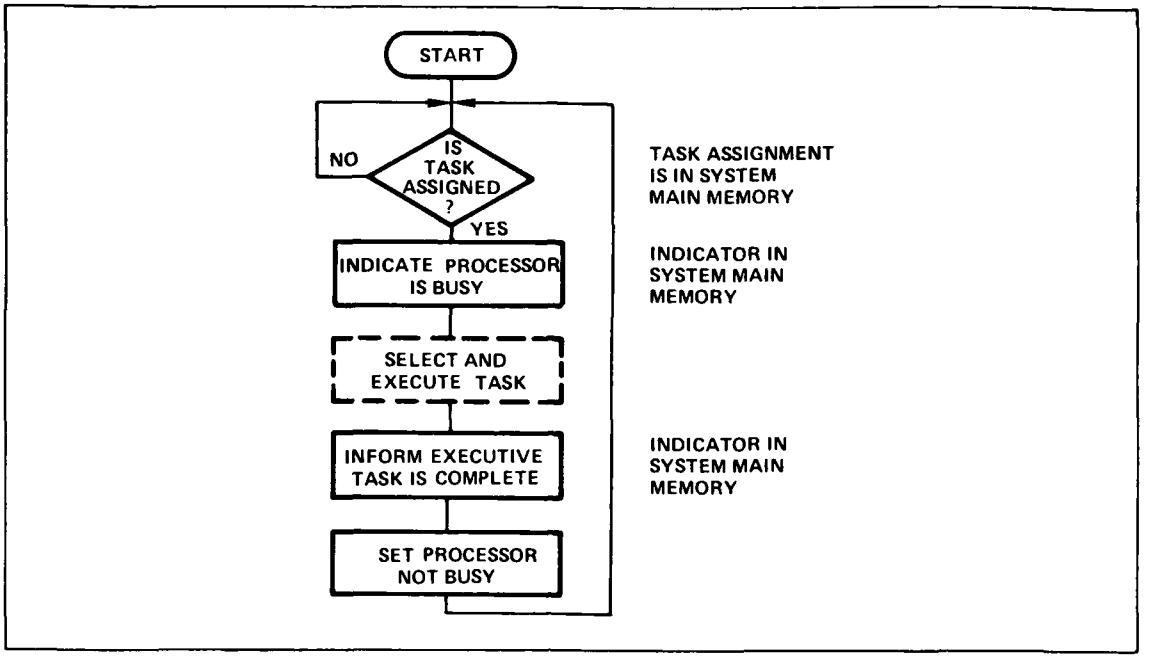


Figure 6-5 - Master/Slave (Polling)

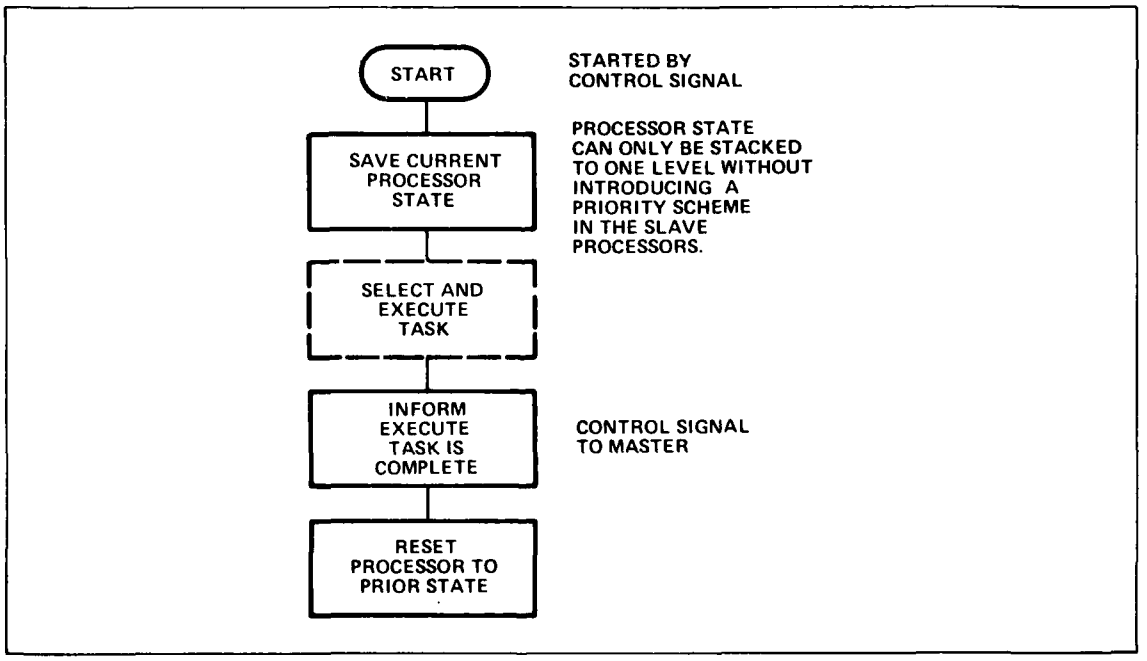


Figure 6-6 - Master/Slave (Control Signal)

6.3.2 Floating Executive (Polling)

This type of control software is essentialllv the same as the polling master/slave without one processor providing synchronization of the tasks.

The polling floating executive is a simple routine if a processor is required to perform one task. If processors are required to perform more than one task, the polling floating executive must have the ability to stack tasks for processors. The ability to stack tasks would increase the control software complexity significantly.

The polling floating executive suffers from the same drawbacks as the polling master/slave software.

Figure 6-7 illustrates a simplified version of the polling floating executive (without stacking).

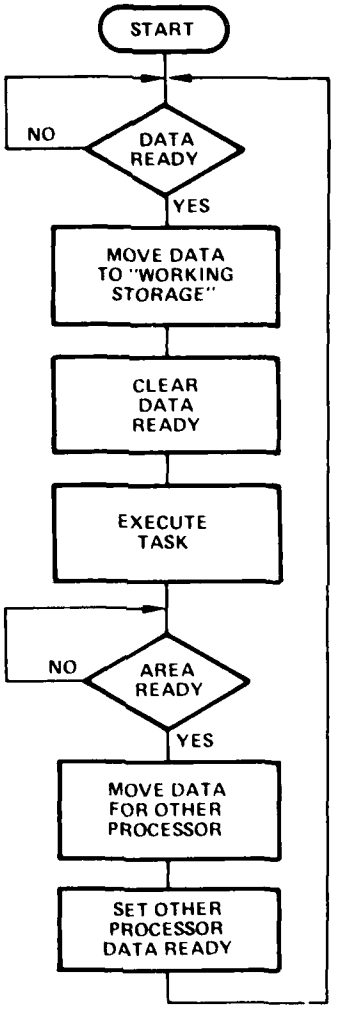


Figure 6-7 - Floating Executive (Polling)

6.3.3 Floating Executive (Multiprogrammed)

If the application of the SFMCS requires that the various processors in the system respond to multiple external stimuli on a priority basis, the multiprogrammed floating executive will provide the quickest response. The major drawbacks of the multiprogrammed floating executive are complexity and size of the control software. Figure 6-8 illustrates a multiprogrammed floating executive.

It should be noted that the multiprogrammed floating executive requires a control signal between the processors in the system (more complex hardware).

In order to select the control software for an application in which an SFMCS is to be used, the various methods of software control can be evaluated based on the following criteria:

- Response Time
- Throughput
- Complexity
- Extensibility*
- Size (Memory Requirements)
- Development Cost
- Partitioning Visibility **

Table 6-1 presents a comparison of the various control software methods as applied to the SFMCS.

---

*Extensibility is the ability to modify the functions of the system without requiring changes to the system design.

**Partitioning visibility is the amount of knowledge that the applications programmer must have of where/how the various functions are partitioned in the system.

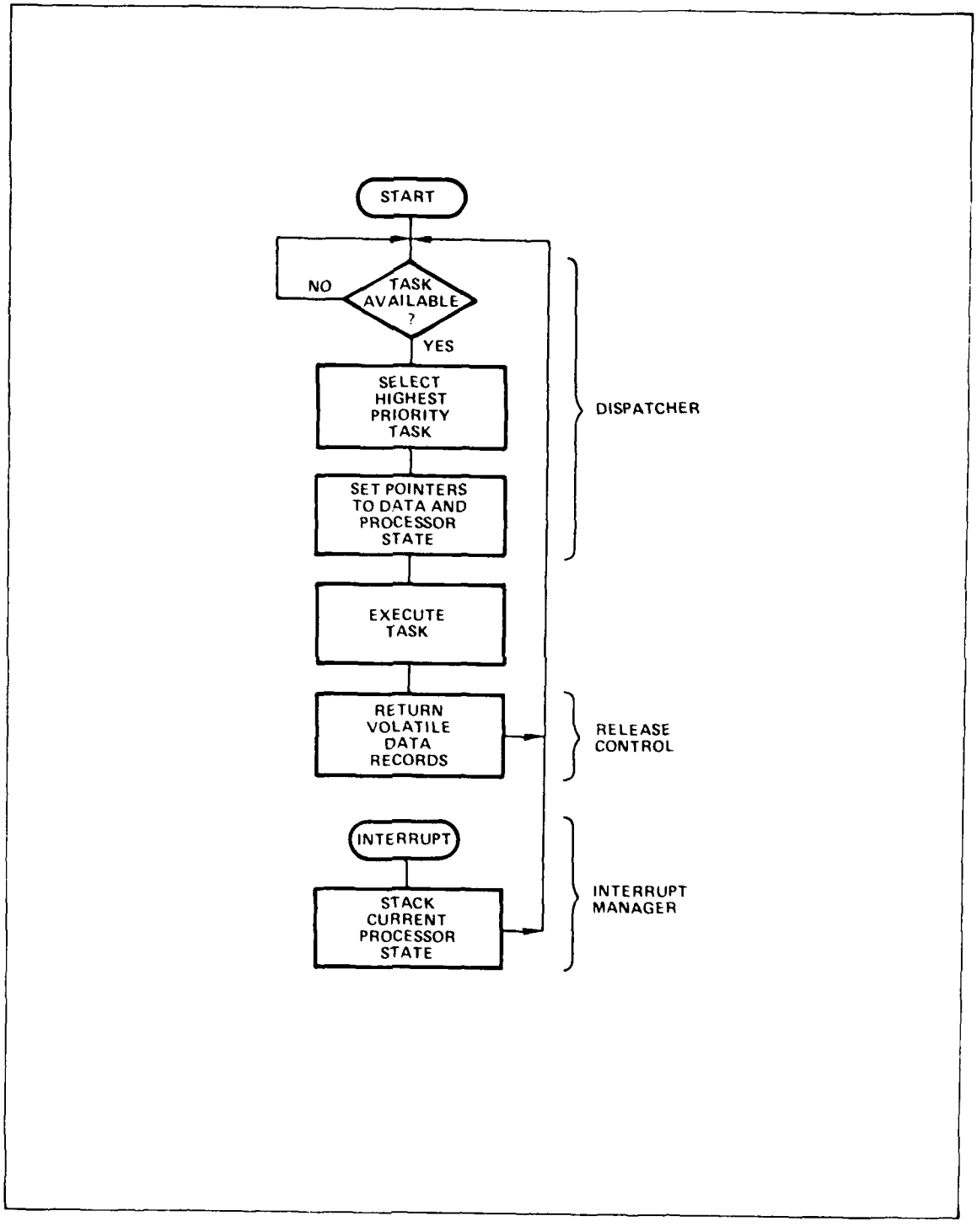


Figure 6-8 - Floating Executive (Multiprogrammed)

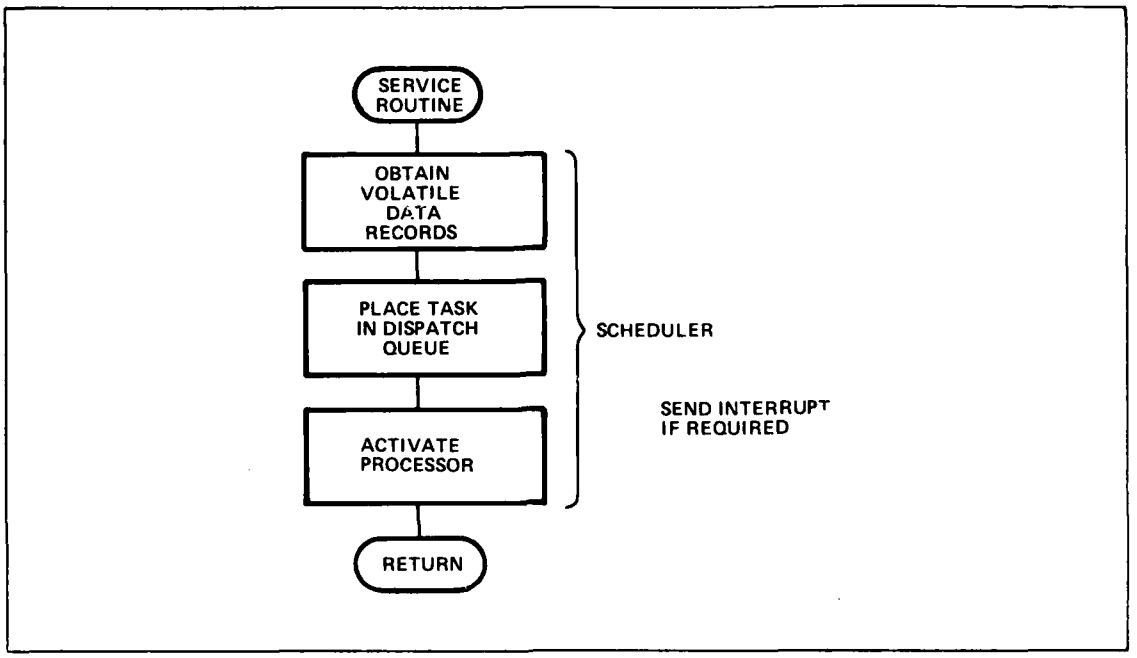


Figure 6-8 - (Cont.)

TABLE 6-1

COMPARISON OF SFMCS SOFTWARE CONTROL METHODS

Criteria Control Method	Response Time	Throughput	Complexity	Extensibility	Size	Development Cost	Partitioning Visibility
Polling Master/Slave	Poor	V. Good	Simple	V. Good	Small	Low	V. Visible
Control Signal Master/Slave	Fair	Good	Moderate	Poor	Medium	Medium	Visible
Polling Floating Exec. (No Stack)	Poor	V. Good	Simple	Fair	Small	Low	V. Visible
Polling Floating Exec. (Stack)	Poor	Good	Moderate	Fair	Medium	Medium	Visible
Multiprogrammed Floating Exec.	Excellent	Good	High	V. Good	Large	High	Practically Invisible

6.4 High Order Language/Advanced Software Tools

It is evident that the distribution of software (both programs and data) within a super-federated system requires careful analysis of the software functions to develop a partitioning scheme which will produce the required throughput and satisfy the associative software/hardware modularity goal. A manual partitioning scheme would, in most cases, result in a software architecture which would not follow modularity guidelines.

Two items appear to be desirable to support development software for a super-federated or tightly coupled system. The first item would be a concurrent high order language which would be efficient enough to support real-time processing and relieve application programmers of the requirement of understanding the details of the hardware configuration. The current Ada development could conceivably lead to solutions to these requirements. The second item required to support software development is a computer aided partitioning system. Such a system would aid the system's programmers to define the hardware architecture, e.g. number and configuration of quads, and evaluate various software partitioning schemes.

7. SFMCS SIMULATION MODELING (Expanded System)

The validation of the expanded SFMCS architecture (Figures 5-7, C2 and C3), by simple simulation techniques was explored and in the course of establishing a model, the significance of each element in the system, in terms of its effect on throughput, was determined.

The major activity in the expanded system occurs in each quad since the host simply provides the means of initializing the system and interfacing it with the analog and serial digital data sources/users. The latter occurs virtually autonomously through memory-mapped I/O channel buffers. Further, traffic between host and quads for missile applications is relatively light both in quantity and frequency. The detailed timing analysis for the quad time-phased ring is given in Appendix A, and this, in many ways, preempts the need and effectiveness of a higher level simulation. However, the structure of the expanded system using several quads and a host computer was characterized before the detailed timing analysis of the quad was performed.

Major elements of the system model are shown in Figure 7-1. These are developed as follows:

1) Microprocessor (CPU) Model
2) Memory Address Translator Model
3) Priority Resolution Model
4) Microbus Model
5) Memory Model

7.1 Microprocessor/CPU Model

Figure 7-2 shows the simulation model developed for the microprocessor/CPU.

Instructions are classified according to type, memory access, local, wait states and extended addressing.

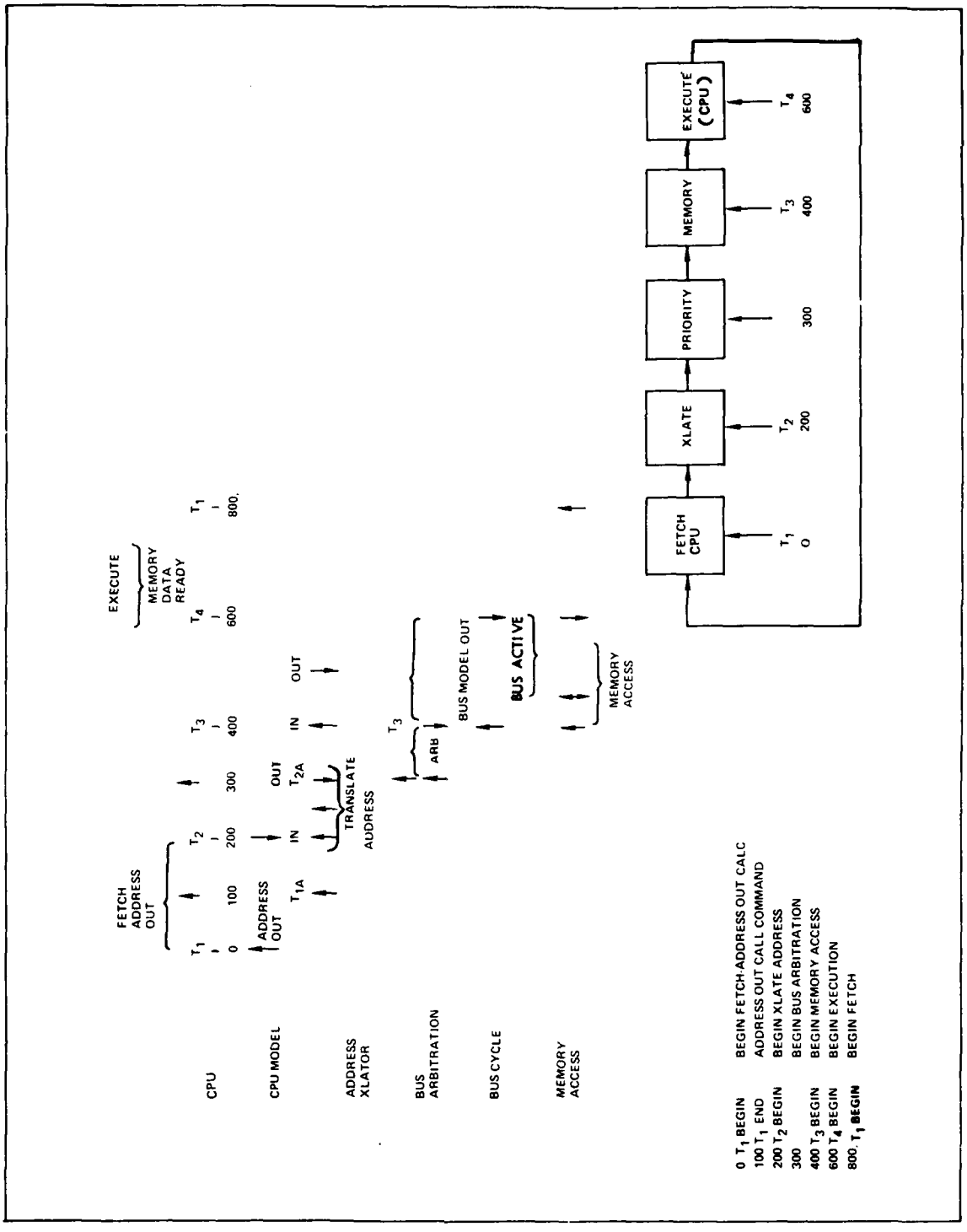


Figure 7-1 - SFMCS Major Elements and Timing

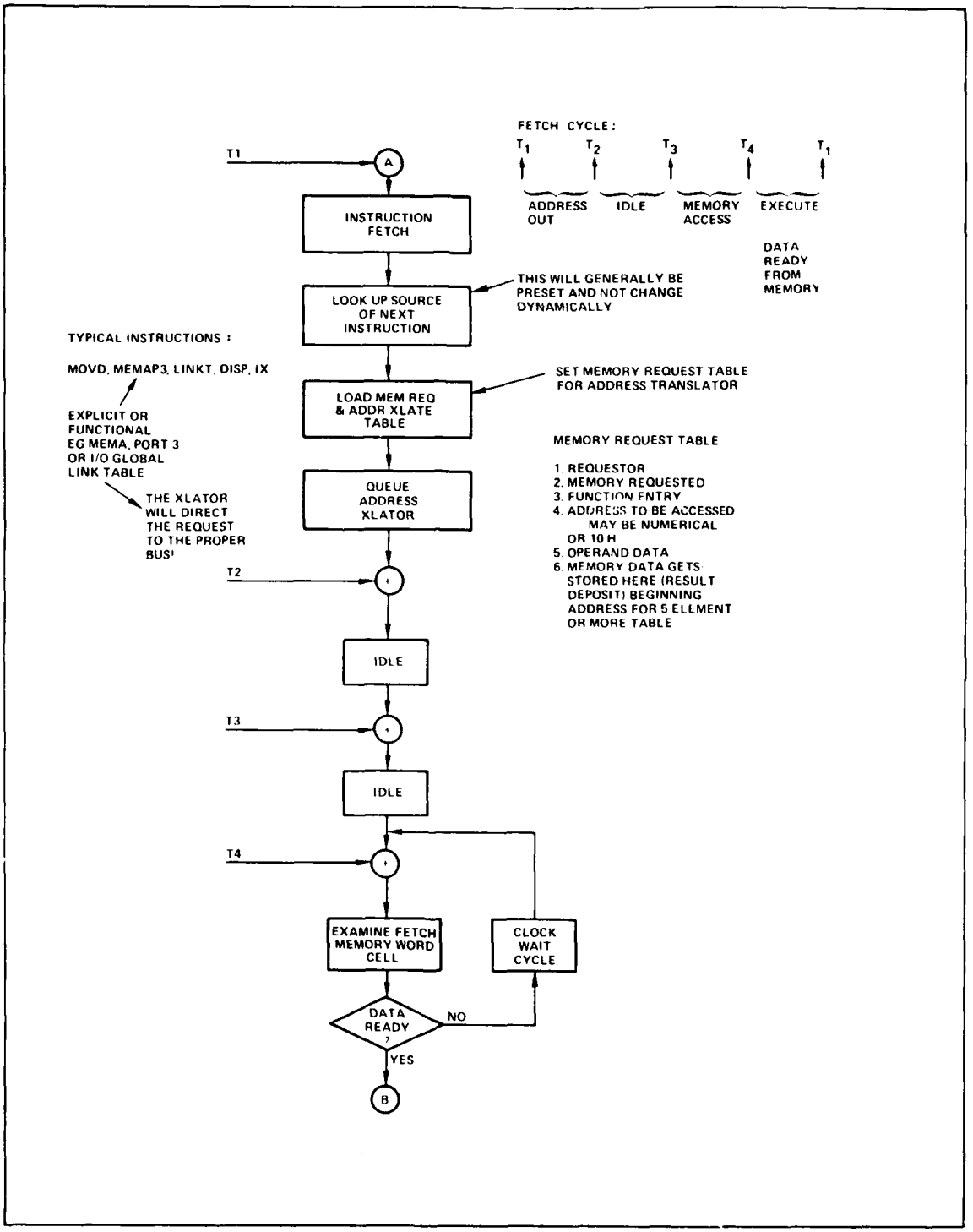


Figure 7-2 - Microprocessor (CPU) Model Flow Diagram (SA1)

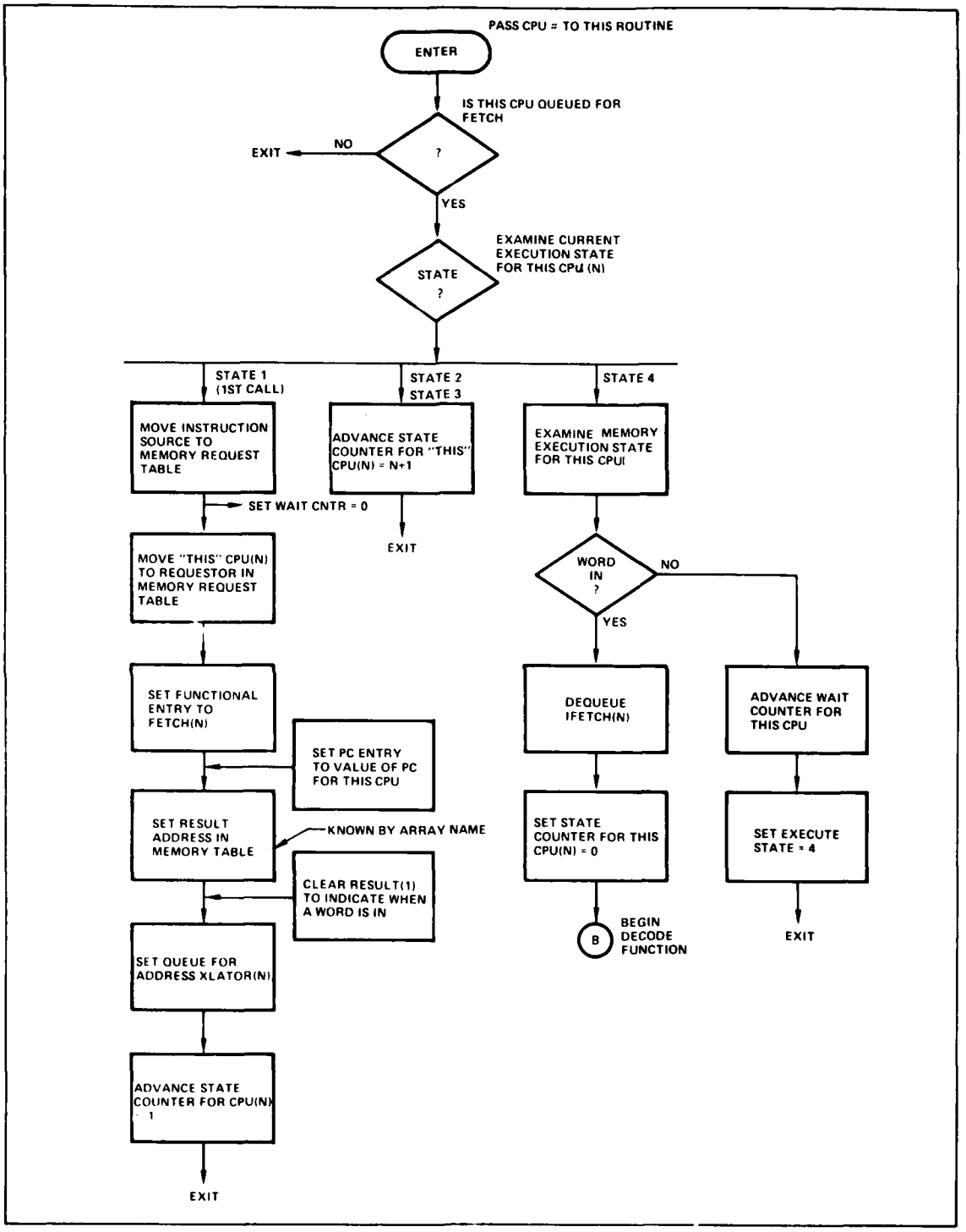


Figure 7-2 - (Cont.)

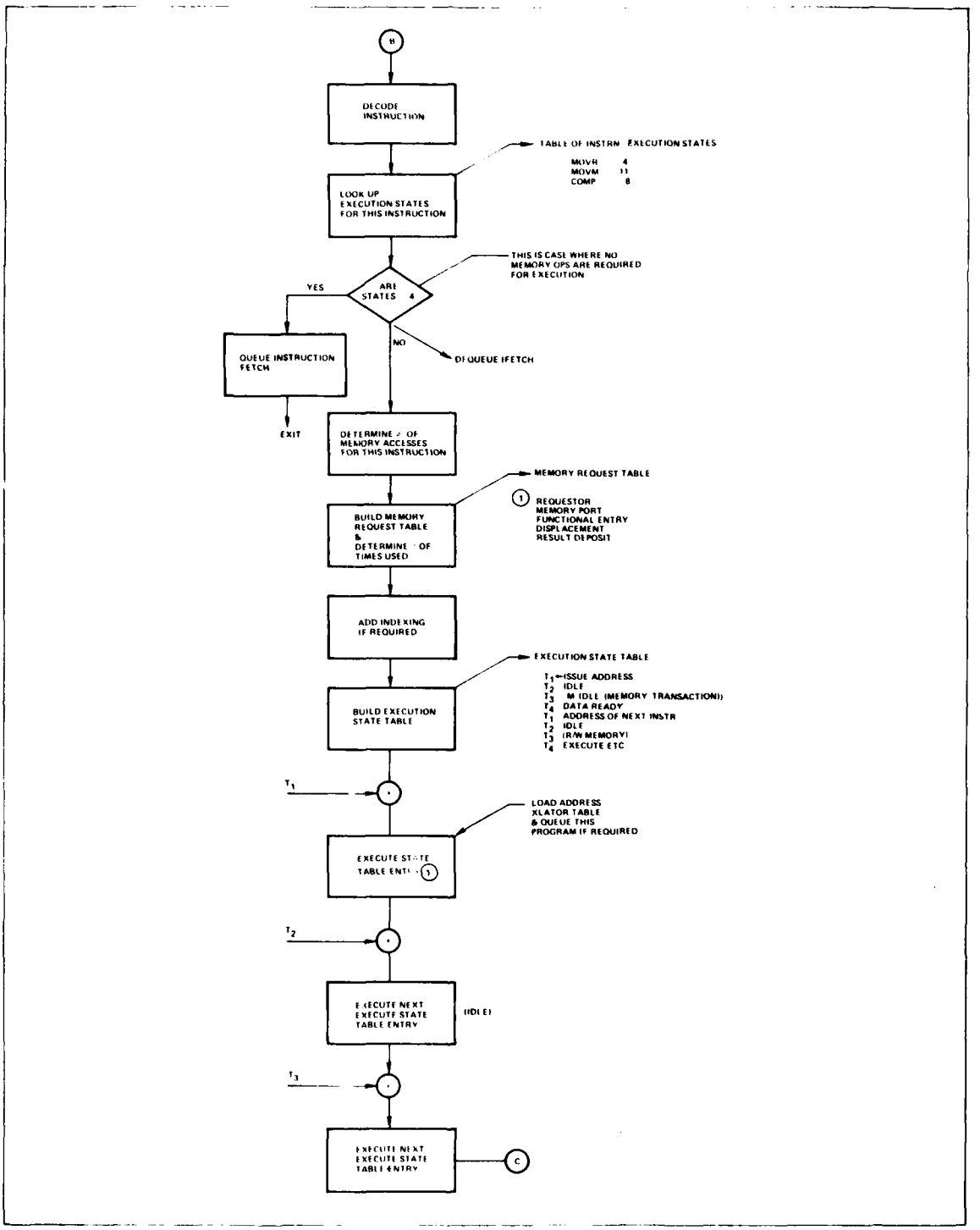


Figure 7-2 - (Cont.)

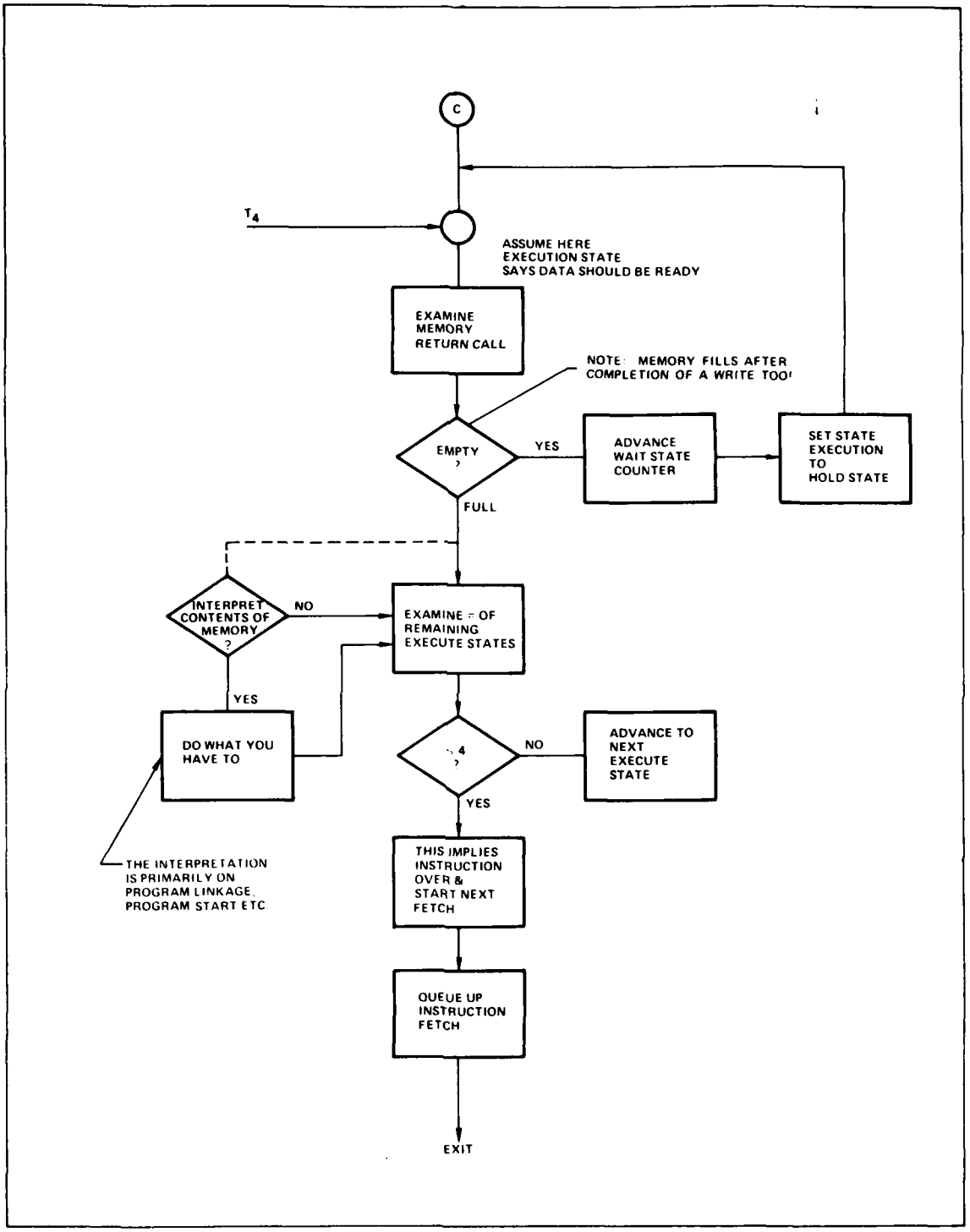


Figure 7-2 - (Cont.)

Move Data

- Memory operation Y, N?
- Data location Mem A, Port no.=?
  B
  C
- Memory fetches/stores/instruction
- Total clock cycles/instruction
- Probability of an indexed instruction
- Number of repeats
- Next source of instruction field

A typical example stated in high order language (HOL) would be:

MD, Mem A, P3, 4, 17, .3, 2, L

Which means Move data to/from memory. Specifically:

From Memory A, Port 3
The number of memory access =4
The dwell time for this instruction is 17
The probability the instruction is indexed is 0.3
Repeat this instruction twice

What happens is as follows:

1) The CPU model decodes an MD operation
2) The timing for the memory access is determined
   17 clock/4 machine cycles/state = 4.X cycle

Since there are now four memory accesses, 1 wait state is generated which schedules execution as follows:

$T_1\ T_2\ T_3\ T_4\ T_1\ T_2\ T_3\ T_4\ T_1\ T_2\ T_3\ T_4\ T_1\ T_2\ T_3\ T_3\ T_4$

MEM OP MEM OP MEM OP MEM OP 1 WAIT due Prog.

If the memory is not ready at T_3 then wait states are injected.

3) A request for access to Mem A P_3 is sent to the address translator. The translator determines which bus the data is on and queues a request for Mem A P_3 to the proper priority resolver.

The priority resolver sends a request for a memory operation to Mem A P_3. When data is ready for CPU #X, a ready flag is set and the CPU model continues by fetching the next instruction from the source specified (Local Memory) and repeats the next instruction (decrements the repeat counter by 1 and continues). Other categories of instruction can be defined in a similar manner.

7.2 Address Translation Model

This model (Figure 7-3) receives requests for memory and determines which bus access model to send the request to.

7.3 Priority Resolution (Bus Access)

This device is in reality a priority resolver (Figure 7-4). Its inputs are requests for bus service and its output is a request to the bus model for a transaction.

The form of the request is:

Device #X requests service of bus #.

The priority scheme is variable. It may be fixed, head to tail, i.e., the next priority is dependent on the previous device grant. It may be fixed cycle, the cycle may rotate with time independent of access, and other methods may be used. The point here is to identify the bus access model as an entity that it may be attached/detached, from/to any bus subsystem.

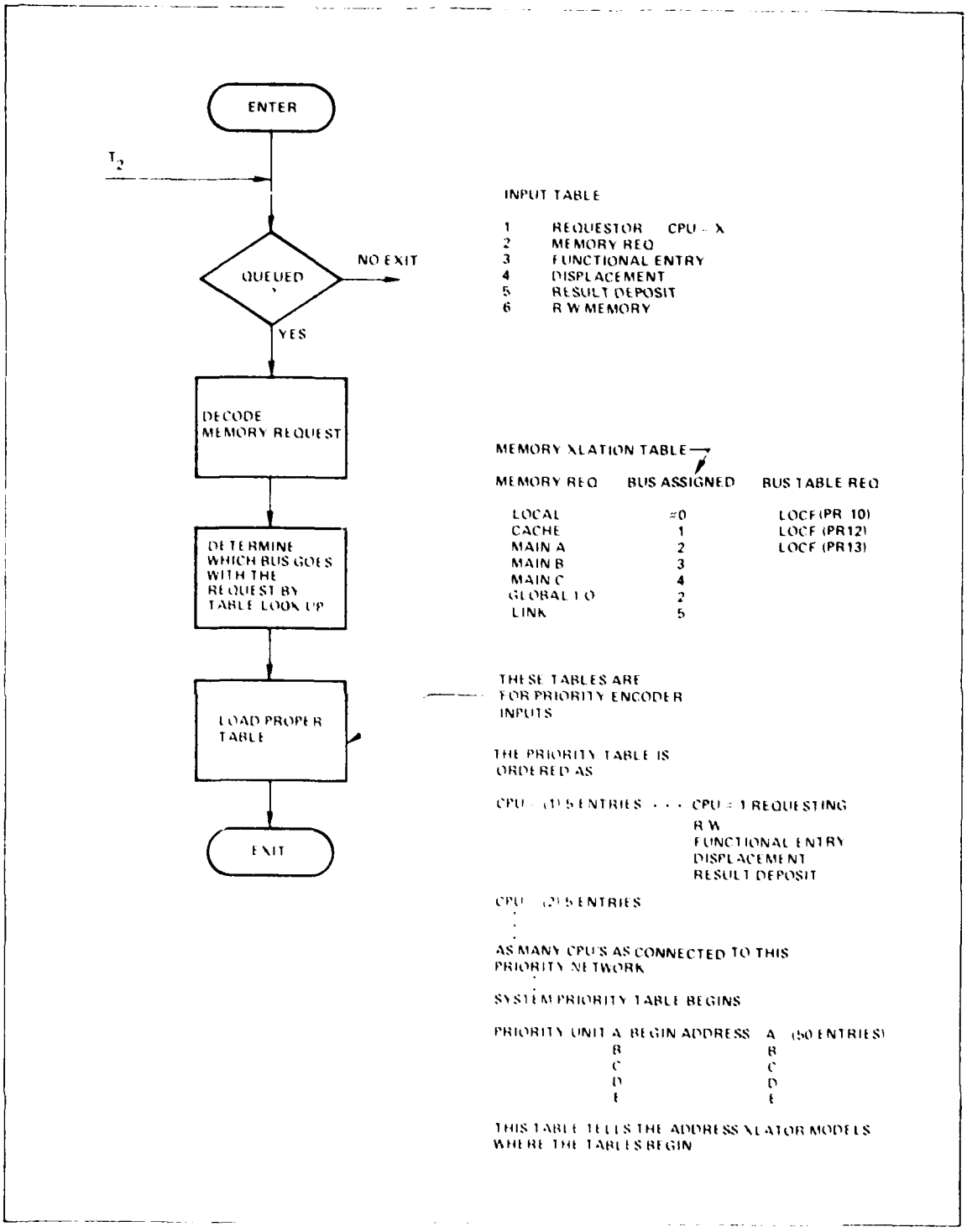


Figure 7-3 - Memory Address Translator Model Flow Diagram

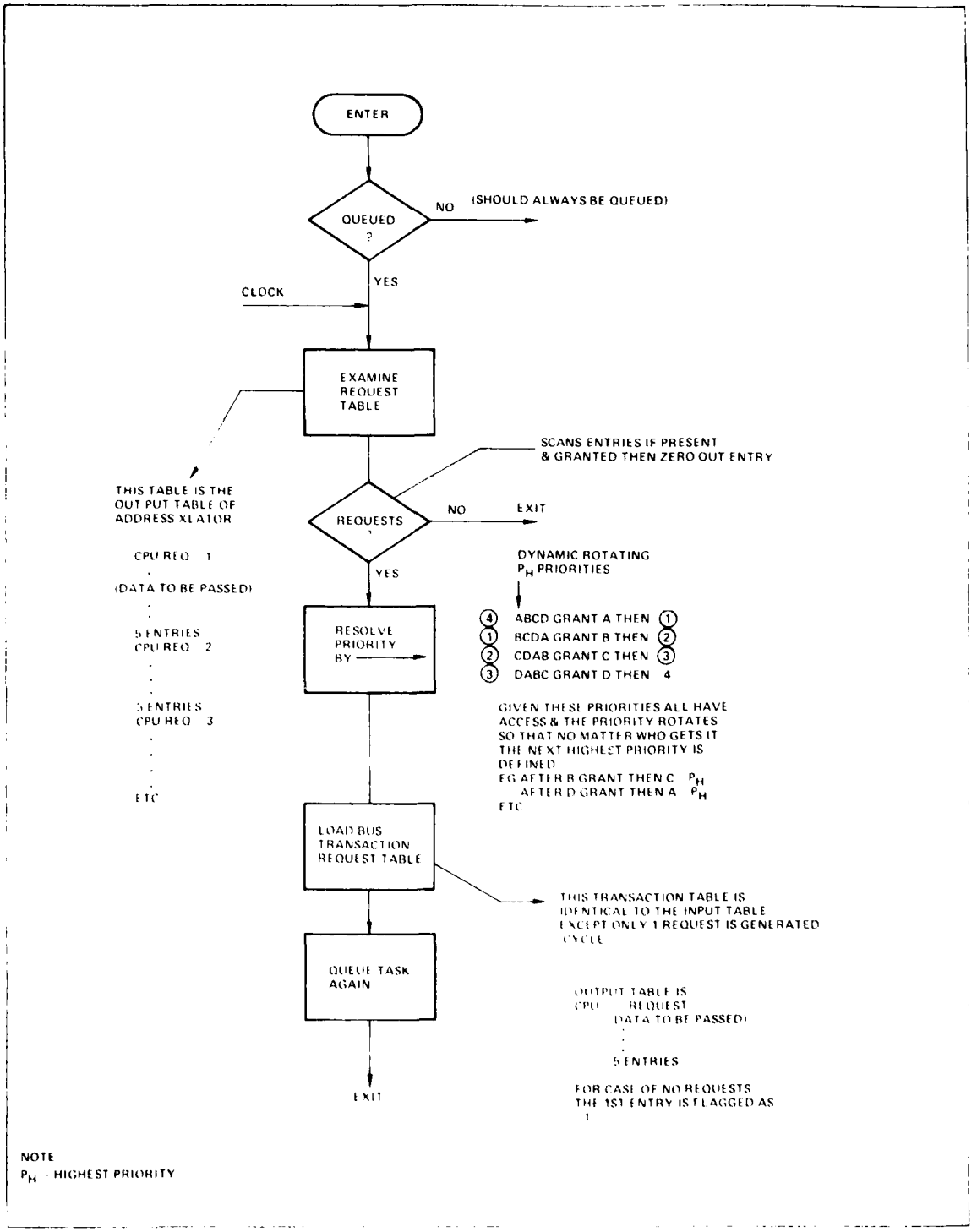


Figure 7-4 - Priority Resolution Model Flow Diagram

7.4 Microbus Model (Bus Model)

Given there are a number of buses in the system, a bus model is required. All bus models need not be the same. They are characterized by the number of devices connected to them and their cycle time, i.e., time to bus, time from bus. (Program Linkage Model) Table driven, interrupt driven (I/O Model).

This is a relatively simple model as shown in Figure 7-5.

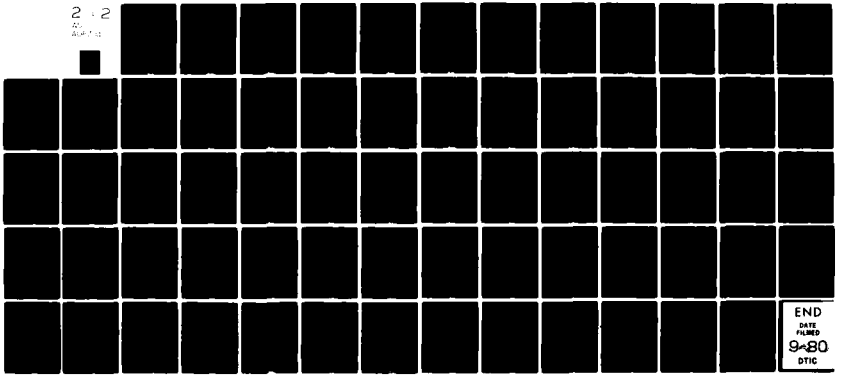
7.5 Memory Model

Each memory model has an access control determined by the number of users connected to it (Figure 7-6). Part of this access control is a priority network which can be identified as a priority model. The one shown is a dynamic rotating priority. Other priority schemes may be used such as fixed linear select.

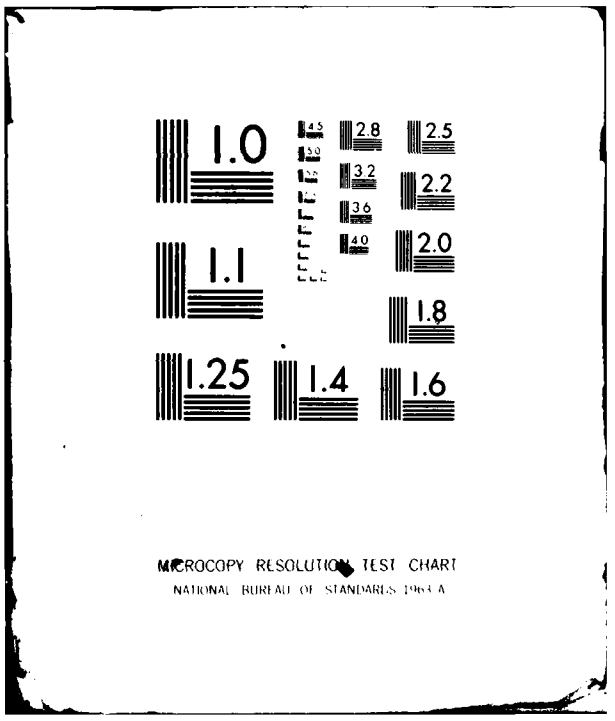
The priority module has a cycle time associated with it. The output of the priority network results in a request for a certain memory bank operation In the case of local memory this model will be a demand type i.e. since it is the only device using the memory no priorities are involved. In the case of Cache 4, devices may request service. In the case of Memory Bank 4, 9 devices may access this one. In general, the priority model should be a separate, detatched entry independent of any system configuration. The idea is to be able to attach different priority models to different memories.

AD-A087 041 RAYTHEON CO BEDFORD MA MISSILE SYSTEMS DIV F/G 17/7
MODULAR DIGITAL MISSILE GUIDANCE.(U)
JAN 80 F J LANGLEY, J DEMETRICK, F MARCHILENA N00014-75-C-0549
UNCLASSIFIED BR-11856 ONR-CR233-052-6 NL

2 : 2



END
DATE FILMED
9-80
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

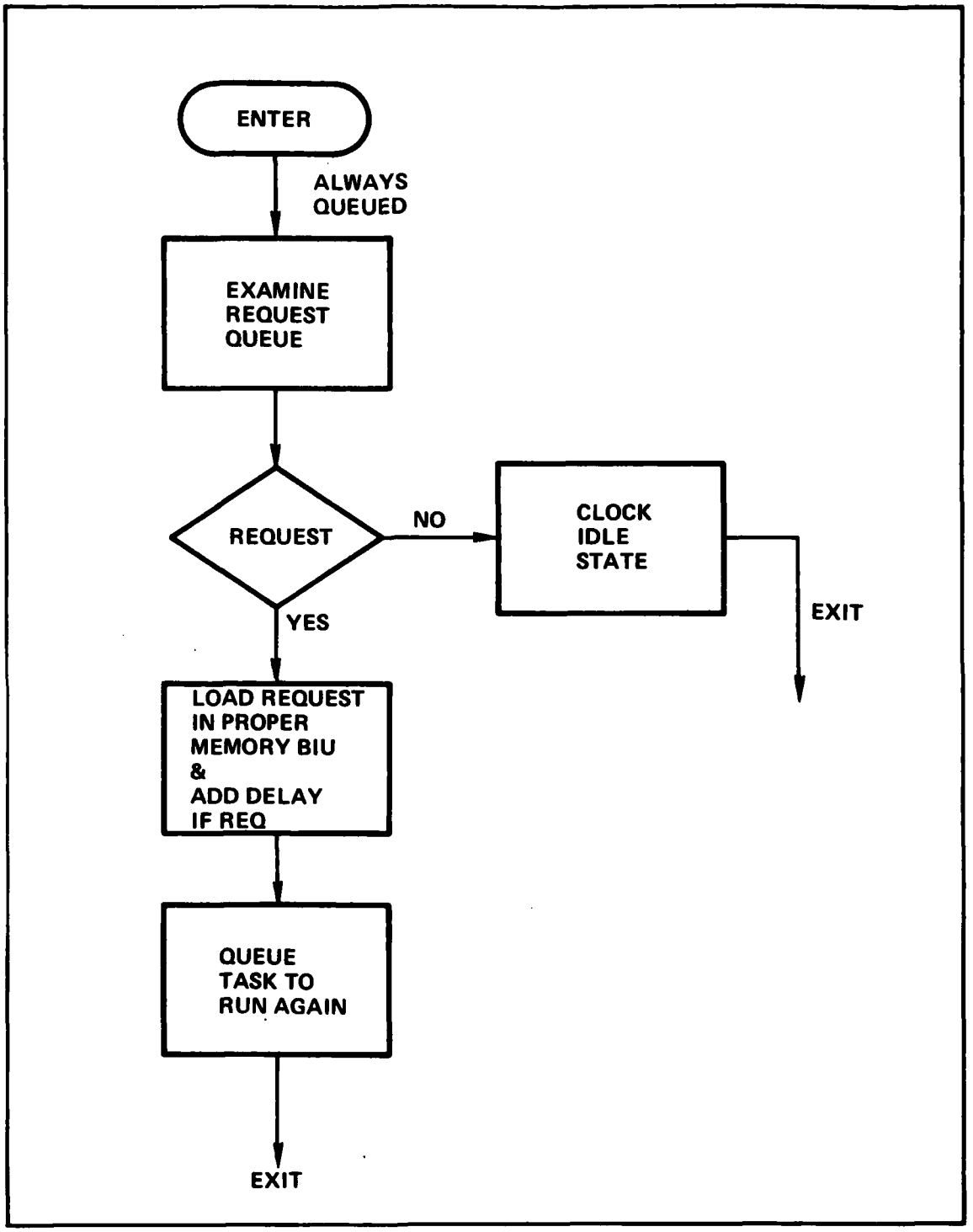


Figure 7-5 - Microbus Model - Flow Diagram

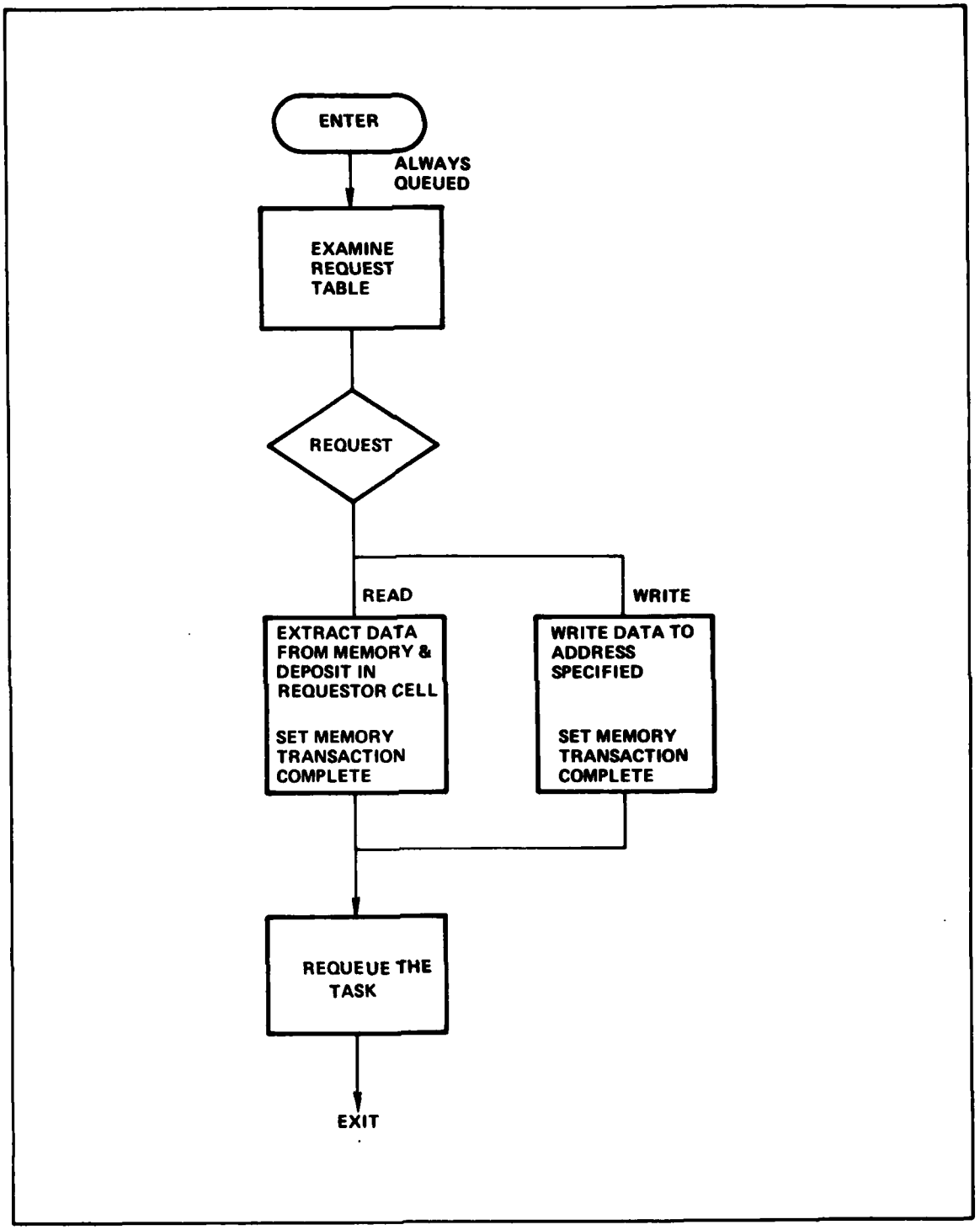


Figure 7-6 - Memory Model Flow Diagram

8. REFERENCES

The following is a list of all the reports and papers published as a result of the Navy Modular Digital Missile Guidance Program together with papers from other sources referenced during the course of the studies.

R-1 Hall, B.A., and Trainor, M.V., "Modular Digital Missile Guidance System Study," Ph. I Report, 30 June 1974, DDC-AD784-969/8GA.

R-2 Langley, F.J., and Cooney, J.J., "Synchronous Microcomputer System for On-Board Missile Guidance and Control," 1975 National Computer Conference, Anaheim, CA, 19-22 May 1975.

R-3 Hall, B.A., and Langley, F.J., "Modular Digital Missile Guidance," Ph. II Report, DDC-AD-B010399L, 28 January 1976.

R-4 Langley, F.J., "Macro-Modular Microcomputer Family for On-Board Missile Guidance and Control," IEEE/NADC Warminster, 22-24 June 1976.

R-5 Cole, B.C., "The Impact of LSI Microprocessors on the Military," Electronic Warfare Defense Electronics, July, 1978.

R-6 Hall, B.A., Langley, F.J. and Wefald, K.O., "Computer Design Requirements for Digital Air-to-Air Missiles," AIAA Guidance and Control Conf., San Diego, CA., 16-18 August 1976.

R-7 Langley, F.J., Kaplan, S., "Raytheon Macromodular Microcomputer Family", Proceedings of NASA/JPL Microprocessor Seminar, Caltech, April, 1977, JPL Pub. No. 77-39.

R-8 Langley, F.J., "Modular Digital Missile Guidance System Study." Ph. III Report, DDC-AD-A042466, 4 May 1977.

R-9 Langley, F.J., "Macro Modular Microcomputer Family for Digital Missile Guidance and Control" DDR&E/IDA Symposium on the Utilization of LSICs in Military Systems, Arlington, VA. 9 August 1977.

R-10 Nesline, F.W., and Langley, F.J., "Computer Architecture for Digital Control of Homing Missiles," SAE Aerospace Control and Guidance Systems Committee, Ft. Walton Beach, October 1977.

R-11 Langley, F.J., "The Application of Microcomputers to Digital Missile Guidance and Control," 1977 Mini/Micro Computer Conference and Exposition, Anaheim, CA, December 1977.

R-12 Langley, F.J., Sheehan, J. and LaGro, G.A., "Simulating Modular Microcomputers," 11th Annual Simulation Symposium, Tampa, FL, March 1978, and SCS Simulation Magazine, May, 1979.

R-13 Leventhal, L., "Design Tools for Microprocessor Systems," Digital Design, October 1979.

R-14 Langley, F.J., "Modular Digital Missile Guidance", Ph. IV Raytheon Report BR-11344.

R-15 Langley, F.J., and Cruikshanks, A., "Microcomputer-Based, On-Board Missile Guidance and Control," IEEE Guidance and Control Group, Boston Section, MITRE, Bedford, MA, May 1978.

R-16 Langley, F.J., and Demetrick, J., "An Adaptive Microprocessor/Microbus Interface Module," IEEE Annual Microprocessor Workshop, John Hopkins University, APL, Laurel, MD, June 1978.

R-17 Langley, F.J., Goldstein, D.S. and Mannion, A.J., "Real Time Signal Processing Devices for Missile Guidance and Control", 22nd SPIE Symposium, San Diego, CA, August 1978.

R-18 Langley, F.J., "Modular Digital Missile Guidance," Ph. V Report, DDC-AD-A072544. 30 November 1978.

R-19 Langley, F.J., "Application of Digital Control Techniques to Modular Ship-Launched Missiles," Ph.1 Report, Raytheon Report No. BR-10472, 8 January, 1979.

R-20 Langley, F.J., "Applications of Microprocessing in Distributed Systems", AIAA/DPMA Microprocessor/Microcomputer Applications Conference, Newport Beach, CA, October 1978, Washington, D.C., 3 April 1979 and Boston, MA, 1 May 1979.

R-21 Langley, F.J., "Federated Microcomputer Systems for On-Board Missile Guidance and Control", NATO AGARD Guidance Control Panel, Ottawa, Canada, May 8-11, 1979.

R-22 "MCS-86 User's Manual", Intel Corporation, February 1979.

R-23 "Z8001 CPU/Z8002 CPU Product Specification", Zilog Corporation, March 1979.

R-24 Zimmerman, T.A., and Barbe, D.F., "A New Role for Charge Coupled Devices: Digital Signal Processing", Electronics, 31 March 1977.

R-25 Rabiner, L.R., Schafer, R.W., and Rader, C.M., "The Chirp Z-Transform Algorithm," IEEE Trans. on Audio and Electroacoustics, Vol. AU-17, pp. 86-92, June 1969.

R-26 Claeys, C.L., et al, "Elimination of Stacking Faults for Charge-Coupled Device Processing," Proceedings of the Third International Symposium on Silicon Material Science and Technology, Electrochemical Society, May 1977.

R-27 "Military Standard Aircraft Internal Time Division Command/Response Multiplex Data Bus", MIL-STD-1553B 16 June, 1978.

R-28 "Modal Superdips" Data Sheet, Meret Inc.,

R-29 "HD 15531" Data Sheet, Harris Semiconductor.

R-30 "Proposed Military Standard Weapon Internal Time Division Multiplex Data Bus", AFATL/DLMM, 1 October, 1979.

R-31 Shaunfield, J.E., "EMI/EMP Resistant Data Bus", Final Report, DDC-AD-B014579, September, 1976.

R-32 "Modular Guidance System Simulation, User's and Programmer's Manuals", Raytheon Report BR-10438/39 June 1978, NSWC Contract N60921-78-C-A085.

R-33 Burks, A.W. Goldstone, H.H., Von Neuman, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", Part 1 and 2, Datamation, Sept. and Oct. 1962.

R-34 Amdahl, G.M., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", Proc. SJCC, pp. 483-485, 1967.

R-35 Keit, H.A., "The Polymorphic Principle in Data Processing", IRE WESCON, pp. 24-28, 1960.

R-36 Porter, R.E., "The RW-400, A New Polymorphic Data System", Datamation, Volume 6, No. 1, pp. 8-14, Jan. and Feb. 1960.

R-37 Anderson, J.P., et al., "D-825, A Multiple Computer System for Command and Control", Proc. FJCC, pp. 86-96, 1962.

R-38 Slotnik, D.L., et al., "The SOLOMON Computer", Proc. FJCC, pp. 97-107, 1962.

R-39 Stokes, R.A., "ILLIAC IV: Route to Parallel Computers", Electronic Design, pp. 64-69, Dec. 30, 1967.

R-40 Rudolph, J.A., "The Associative Processor - A New Computer Resource", IEEE Region 6 Conference, April 1969.

R-41 Huttenhoff, J.H., and Shively, R.R., "Arithmetic Unit of a Computing Element in a Global, Highly Parallel Computer", IEEE Trans. Computers, Volume C-18, No. 8, pp. 695-698, Aug. 1969.

R-42 Entner, R.S., Bersoff, E.H., "Operating System Reliability for the Navy Advanced Avionic Digital Computer" IEEE Trans. on Aerospace and Electronic Systems, Vol. AES-7, No. 1, pp. 67-72, Jan. 1971.

R-43 Cooley, J.W. and Tukey, J.W., "An Algorithm for the Machine Calculation of Complex Fourier Series," Math. Comp., V. 19, 1965 pp 297-301. MR 31#2843.

R-44 Berglund, G.D., "Fast Fourier Transform Hardware Implementations An Overview, "IEEE Trans. on Audio and Electroacoustics Vol. Au-17, No. 2, pp 104-108, June 1969.

R-45 Flynn, M.J., "Some Computer Organizations and Their Effectiveness", IEEE Transactions on Computers, Vol. C-21, No. 9, September 1972.

R-46 Lorin, H., "Parallelism in Hardware and Software, Real and Apparent Concurrency", Prentice Hall, 1972.

R-47 Flores, I., "Computer Programming", Prentice Hall, 1966.

APPENDIX A

SUPER-FEDERATED MICROCOMPUTER SYSTEM (SFMCS) TIMING AND THROUGHPUT ANALYSIS

A1 Introduction

In order to determine the throughput capability of the SFMCS architecture, a representative state-of-the-art commercial microprocessor was selected and its performance evaluated as a single processor using a realistic avionics instruction mix. The performance of the SFMCS quad multiprocessor was then determined using the above microprocessor and instruction mix, applied to four quad configurations viz:

1) Basic shared-memory multiprocessor (Figure 4-1);
2) Shared-memory multiprocessor using the time-phased ring technique.
3) Dedicated microprocessor program memories and shared data memory without time phasing.
4) Dedicated microprocessor program memories and shared data memory with time phasing.

In each of the above cases, the performance improvement of the quad versus the single processor was noted.

Lastly, the performance of Configuration 4 was determined using different avionic instruction mixes in each microprocessor.

A2 Calculation of Intel 8086 Throughput

There are several established avionic instruction mixes which are representive of various guidance and control-type algorithms. These mixes are shown in Table A-1.

TABLE A-1
CANDIDATE INSTRUCTION MIXES (%)

	STANDARD AIRBORNE	F4 FIRE CONTROL	F15 AUTO FLIGHT CONTROL	R.F4 INERTIAL NAV
MOVE	45	22	41	45
ADD/SUB	9	17	19	9
MULTIPLY	5	17	4	<1
DIVIDE	.2	4	-	<1
SHIFT	5	2	3	8
LOGICAL	5	4	10	13
TEST & BRANCH	30	32	21	24
I/O CONTROL	1	2	2	-

The Intel 8086 has over 100 basic instructions. Within the basic instructions, a variety of options may be used to perform the same basic operation. In addition, several different addressing modes may be used in the instruction.

The execution time of an instruction is therefore a sum of the contributions due to basic type, option selected, and address mode. A method of weighted averages will be used that considers not only the above mentioned factors but also includes a usage weight as well. An analysis of the move instruction will be used to illustrate the technique.

A2.1 Move Instruction

There are several types of move instruction which are as follows:

Mnemonic: MOV
Description: MOV performs a byte or word transfer from a specified source to a specified destination.

Encoding:
Memory or Register to/from Memory or Register:

1 0 0 0 1 0 d w mod reg r/m

Percent Usage

if d = 1 then SRC = EA, DEST = REG
else SRC = REG, DEST = EA

Timing (clocks):	register to register	2	10
	memory to register	8+EA	5
	register to memory	9+EA	5

Immediate Operand to Memory or Register:

1 1 0 0 0 1 1 w mod 0 0 0 r/m data data if w=1

SRC = data, DEST = EA

Timing (clocks):	Immediate to register	4	10
	Immediate to memory	10+EA	5

Immediate Operand to Register:

1 0 1 1 w reg data data if w=1

SRC = data, DEST = REG

Timing: 4 clocks		12	15

Memory Operand to Accumulator:

		Percent Usage	
1 0 1 0 0 0 0 w addr-low addr-high			
if w = 0 then SRC = addr, DEST = AL			
else SRC = addr+1:addr, DEST = AX			
Timing: 10 clocks		30	20

Accumulator Operand to Memory:

		Percent Usage	
1 0 1 0 0 0 1 w addr-low addr-high			
if w = 0 then SRC = AL, DEST = addr			
else SRC = AX, DEST = addr+1:addr			
Timing: 10 clocks		25	20

Memory or Register Operand to Segment Register:

		Percent Usage	
1 0 0 0 1 1 1 0 mod 0 reg r/m			
if reg = 01 then SRC = EA, DEST = REG			
else undefined operation			
Timing (clocks):	register to register	2	2
	memory to register	8+EA	3

Segment Register Operand to Memory or Register:

		Percent Usage	
1 0 0 0 1 1 0 0 mod 0 reg r/m			
SRC = REG, DEST = EA			
Timing (clocks):	memory to register	9+EA	2
	register to register	2	3

Operation:

(DEST) <== (SRC)

Flags Affected:

None

A number of addressing modes are available that are used to calculate the effective address (EA). These times are as follows:

Effective Address Timing

Addressing Mode			Percent Used
1)	No EA calc required	0 clocks	25
2)	Direct 16-bit offset address	6 clocks	50
3)	Indirect through base or index register (BX, BP, SI, DI)	5 clocks	10
4)	Indirect through base or index register with displacement constant	9 clocks	8
5)	Indirect through sum of index register plus base register	7 or 8 clocks	5
6)	Indirect through sum of base register plus index register with displacement constant	11 or 12 clocks	2

Now the weight average for EA is calculated as:

Address Mode	Clocks	Percent Used	100/ASTR	Clocks	Total
1	0	25	25	25	0
2	6	50	50	300	300
3	5	10	10	50	50
4	9	8	8	72	72
5	7.5	5	5	37.5	37.5
6	11.5	2	2	23	23
					482.5

$$\text{Time for EA calc} = \frac{482.5 \text{ clocks} \times 200 \text{ nsec/clock}}{100 \text{ EA calculations}}$$

EA time = .965 μsec or 4.825 clocks

In order to calculate the weighted average for the move instruction, the weighted average of the MOV instruction is determined. Here we have:

	Operation	Clocks	Percent Used	Total Clocks
Instruction #1	r-r	2	10	20
	m-r	8+4.82	5	64.1
	r-m	9+4.82	5	69.1
Instruction #2	I-r	4	10	40
	I-m	5+4.82	5	49.1
Instruction #3		4	15	60

	Operation	Clocks	Percent Used	Total Clocks
Instruction #4		10	20	200
Instruction #5		10	20	200
Instruction #6	r-r	2	2	4
	m-r	8+4.82	3	38.4
Instruction #7	m-r	9+4.82	2	27.64
	r-r	2	3	6
				778.34

$$\text{Move time} = \frac{778.34 \times 200}{100} = 1.556\ \mu\text{sec or } 7.78 \text{ clocks}$$

A2.2 Add/Subtract Instructions

The six types of these instructions are:

1) Memory or Register Operand with Register Operand

	Operation	Clocks	Percent Used	Total Clocks
(a)	r-r	3	10	30
(b)	m-r	9+4.82	5	69.1
(c)	r-m	16+4.82	5	104.1

2) Immediate Operand to Memory or Register Operand

(a)	I-M	17+4.82	10	218.2
(b)	I-r	4	20	80

	Operation	Clocks	Percent Used	Total Clocks
	3) Immediate Operand to Accumulator Operand			
(a)		4	10	40
	4) Add with Carry			
(a)	r-r	3	10	30
(b)	m-r	9+4.82	5	47.41
(c)	r-m	16+4.82	5	104.1
	5) Immediate Operand to Memory or Register Operand			
(a)	I-m	17+4.82	5	109.1
(b)	I-r	4	10	40
	6) Increment			
(a)	r	3	5	15
(b)	m	15+4.82	5	99.1
				986.11

$$\text{Add/Subtract time} = \frac{986.11 \times 200 \text{ nsec}}{100}$$

Add/Subract time = 1.972 μsec or 9.86 clocks

A2.3 Multiply Instruction

Multiply (unsigned) 30 percent (usage)

	Clocks	Clock	Percent	Total Clocks
8 bit register	(70-77)	73.5	5	367.5
8 bit memory	(76-83)+4.82	84.3	5	421.5
16 bit register	(118-133)	125.5	10	1,255.0
16 bit memory	(124-139)+4.82	136.3	10	1,363.0
Multiply (Integer) 70 percent (usage)				
8 bit register	(80-98)	89	10	890
8 bit memory	(86-104)+4.82	99.82	10	998.2
16 bit register	(128-154)	141	25	3,525.0
16 bit memory	(134-160)+4.82	151.82	25	3,795.5
				12,615.7

$$\text{Multiply time} = \frac{12{,}615.7 \times 200 \text{ nsec}}{100}$$

Multiply time = 25.231 μsec or 126.15 clocks

A2.4 Divide Instruction

Divide (unsigned) 30 percent (usage)

	Clocks	Clocks	Percent Used	Total Clocks
8 bit register	(80-90)	85	5	425
8 bit memory	(86-96)+4.82	95.8	5	479
16 bit register	(144-162)	153	10	1,530
16 bit memory	(150-168)+4.82	163.82	10	1,638.2

Divide (Integer) 70 percent usage

8 bit register	(101-112)	106.5	10	1,065.0
8 bit memory	(107-118)+4.82	117.3	10	1,173.0
16 bit register	(165-184)	174.5	25	4,362.5
16 bit memory	(171-190)+4.82	185.3	25	4,632.5
				15,305.2

$$\text{Divide time} = \frac{15{,}305.2 \times 200 \text{ nsec}}{100}$$

Divide time = 30.61 μsec or 153.05 clocks

A2.5 Shift Instruction

Shift logical left (25 percent) usage

Single bit reg	2	2	5	10
Single bit mem	15+EA	15+4.82	5	99.1
Var bit reg	8+4/bit	24	10	240
Var bit mem	20+EA+4/bit	65.64	10	328.2
				667.3

Shift logical right (25 percent) usage

Single bit reg.	2	2	5	10
Single bit mem.	15+EA	15+4.82	5	99.1
Var bit reg.	8+4/bit	24	10	240
Var bit mem	20+EA+4/bit	65.64	5	328.2
				667.3

Shift Arithmetical (25 percent) usage

Single bit reg	2	2	5	10
Single bit mem	15+EA	15+4.82	5	99.1
Var bit reg	8+4/bit	24	10	240
Var bit mem	20+EA+4/bit	65.64	5	328.2
				667.3

Rotate (25 percent) usage

Single bit reg	2	2	5	10
Single bit mem	15+EA	15+4.82	5	99.1
Var bit reg	8+4/bit	24	10	240
Var bit mem	20+EA+4/bit	65.64	5	328.2

		667.3 x 4 =		2709.2

$$\text{Shift time} = \frac{2709.2 \times 200 \text{ nsec}}{100} = 5.418 \ \mu\text{sec or } 27.09 \text{ clocks}$$

A2.6 Logical Instructions

Exclusive OR (25 percent) usage

			Percent	Total Clocks
(1)	r-r	3	15	45
(2)	m-r	9+4.82	2	18.96
(3)	r-m	16+4.82	2	41.64
(1)	I-r	4	2	8
	I-m	17+4.82	2	43.64
(1)	I-r	4	2	8

AND 50 percent usage

(1)	r-r	3	25	75
	m-r	9+4.82	10	138.2
	r-m	16+4.82	5	104.1
(2)	I-r	4	5	20
	I-m	17+4.82	5	109

OR 25 percent usage

	r-r	3	10	30
	m-r	9+4.82	3	41.46
	r-m	16+4.82	3	62.46
	I-r	4	3	12

	Percent	Total Clocks	
I-m	17+4.82	3	65.46
I-r	4	3	12
			835.024

$$\text{Logical time} = \frac{835.04 \times 200 \text{ nsec}}{100}$$

Logical time = 1.670 μsec or 8.35 clocks

A2.7 Test and Branch Instructions

Jump on, Less than

Jmp taken	8	25	200
Not taken	4	25	100

JMP

Int. segment	7	10	70
Int. segment	7	5	35
Int. segment	3	20	60
mem	7+EA	5	59.1
Int. segment	16+EA	10	208.2
			732.3

$$\text{Test and branch} = \frac{732.3 \times 200}{100}$$

Test and branch time = 1.464 μsec or 7.32 clocks

A2.8 I/O Control Instructions

Interrupt	25 Percent	Clocks	Percent	Total Clocks
	Type 3	51	12.5	637.5
	Not Type 3	50	12.5	625.0
INTO	7 % pass	52	7	364
	8 % fail	4	8	32
IRET	25 %	24	25	600
CLC	20 %	2	20	40
STC	15 %	2	15	30
				2328.5

$$\text{I/O control} = \frac{2328.5 \times 200}{100}$$

I/O Control time = 4.657 μsec or 23.28 clocks

A2.9 Average Instruction Execution Times

In summary, then the average execution time for each Intel 8086 instruction based on a 200 nsec clock is as follows:

Instruction	Execution Time (μsec)	Clocks
Move	1.556	7.78
Add/Sub	1.972	9.86
Multiply	25.23	126.15
Divide	30.61	153.05
Shift	5.481	27.09
Logical	1.670	8.35
Test and Branch	1.464	7.32
I/O Control	4.657	23.28

A2.10 Intel 8086 Throughput

The throughput of the 8086 may now be determined for the mixes cited in Table A-1. The results are tabulated in Table A-2. It is interesting to note how multiply/divide operations significantly affect throughput. In the F-4 fire control case, a 21 percent multiply/divide load diminishes the throughput by four times over inertial NAV mix where the load was only 0.5 percent.

TABLE A-2

INTEL 8086 THROUGHPUT FOR CANDIDATE INSTRUCTION MIXES

INSTRUCTION	AV TIME (μsec)	AV CLOCKS NO.	STD AIRBORNE	TIME	F-15 AUTO FLIGHT	TIME	R-F4 INERTIAL NAV		F4 FIRE CONTROL	
LOAD AND STORE	1.556	7.78	45%	70.02	41%	63.7	45	70.	22	34.2
ADD/SUB	1.972	9.86	9%	17.7	19%	37.4	9	17.7	17	33.5
MULTIPLY	25.23	126.15	5%	126.5	4%	100.9	0.3	7.5	17	428.9
DIVIDE	30.61	153.05	0.2%	6.1	-	0.0	0.2	6.1	4	122.4
SHIFT	5.418	27.09	5%	27.1	3%	16.2	8	43.3	2	10.8
LOGICAL	1.670	8.35	5%	8.35	10%	16.7	13	21.7	4	6.6
TEST AND BRANCH	1.464	7.32	30%	43.9	21	30.7	24	35.2	32	46.8
I/O CONTROL	4.657	23.28	1%	4.6	2	9.3	0.5	2.3	2	9.3
Execution Time (μsec):				304.05		275.1		204.0		692
Average Ex. Time (μsec):				3.04		2.75		2.04		6.92
Throughput (KOPS):				328		363		490		144

THROUGHPUT SUMMARY

STD AIRBORNE: 328 KOPS
F-15 AUTO FLIGHT: 363 KOPS
R-F4 INERTIAL NAV: 490 KOPS
F-4 FIRE CONTROL: 144 KOPS

AVG = 331.25 KOPS

A3 SFMCS Quad Throughput

The throughput for several architectural cases will now be examined. The cases will be developed to measure both single and multiple processor throughputs.

A3.1 Case 1, Four Processors Sharing a Common Memory (no time phasing)

The timing and arbitration rules for this case, (Figure A-1), are found in Figure A-2.

A3.1.1 With Memory Access Conflicts

In order to determine CPU waiting time, a 25 clock sample will be used. Referring to Figure A-2 $\mu p1$ encounters 14 wait states. The waiting time becomes:

$$\text{Waiting time } (\mu p1) = \frac{14 \text{ wait clock}}{25 \text{ clocks}} = 56 \text{ percent}$$

Similarly for $\mu p2$ we have

$$\text{Waiting time } (\mu p2) = \frac{14}{25} = 56 \text{ percent}$$

Similarly for $\mu p3$ we have

$$\text{Waiting time } (\mu p3) = \frac{14}{25} = 56 \text{ percent}$$

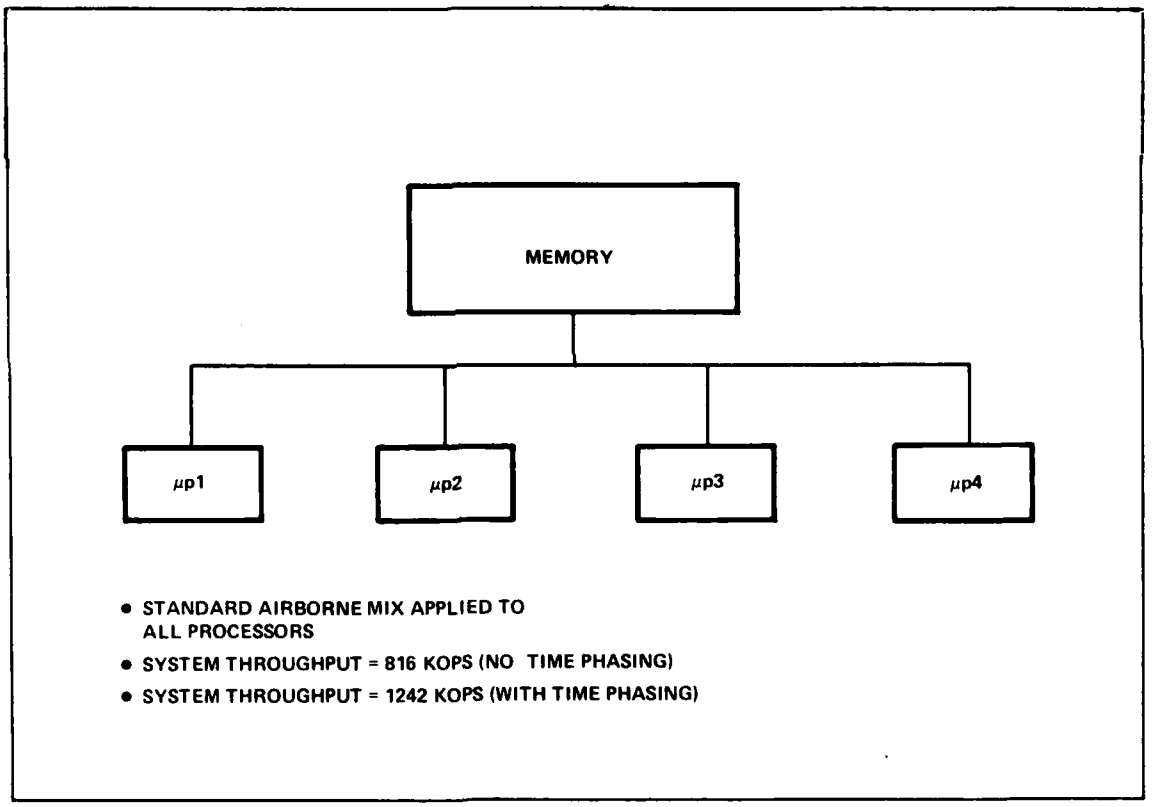


Figure A-1 - Case 1, Four Microprocessors Sharing a Common Memory

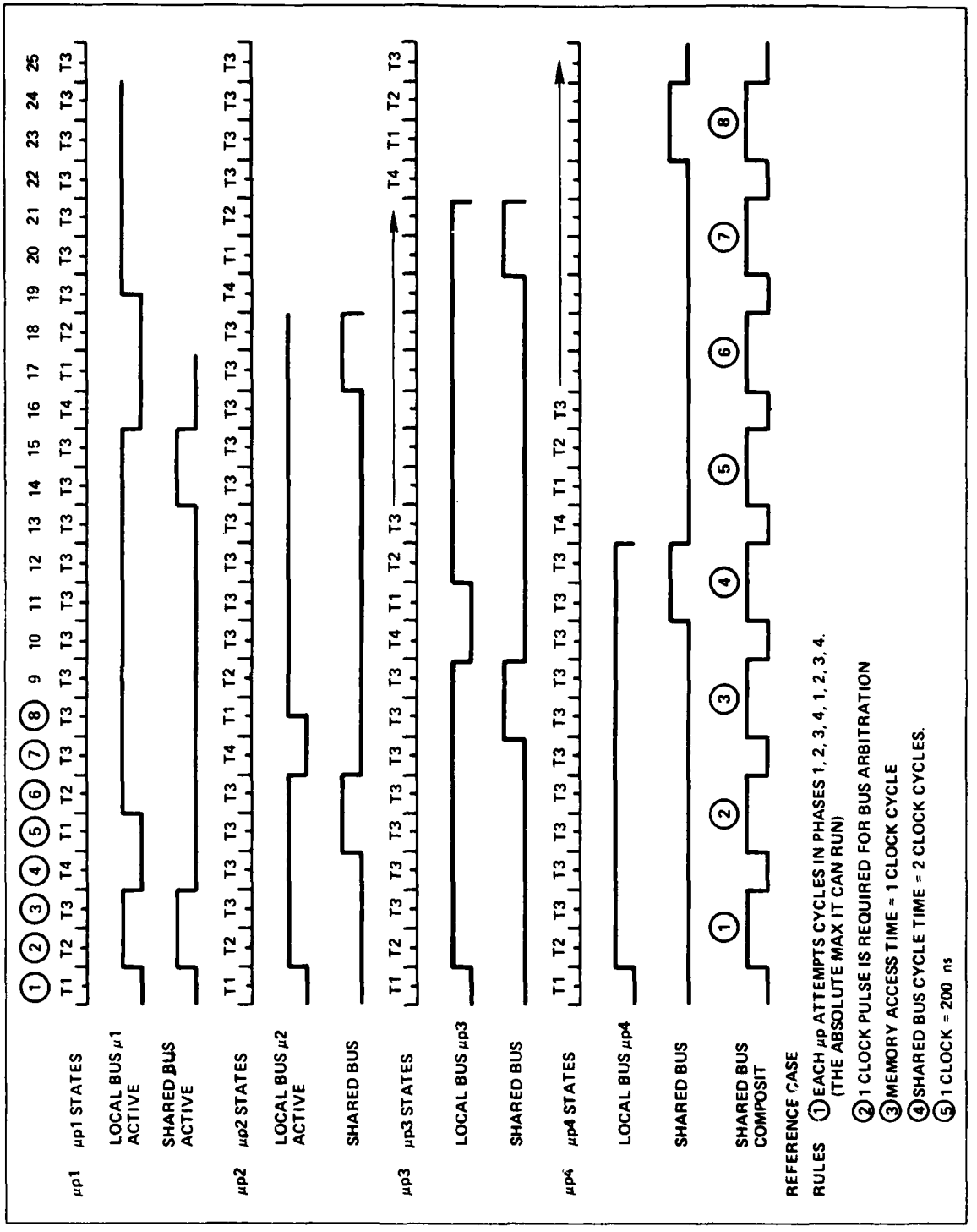


Figure A-2 - Case 1, Four Processors with Shared Memory, No Time Phasing

and for μp4

$$\text{Waiting time } (\mu p4) = \frac{19}{25} = 76 \text{ percent}$$

Applying the standard airborne mix to all processors we get:

μp1	= 304.5 μsec + 304.5 (0.56)	=	475	μsec	= 210 KOPS
μp2	= 304.5 μsec + 304.5 (0.56)	=	475	μsec	= 210 KOPS
μp3	= 304.5 μsec + 304.5 (0.56)	=	475	μsec	= 210 KOPS
μp4	= 304.5 μsec + 304.5 (0.76)	=	535	μsec	= 186 KOPS
		Quad Throughput			816 KOPS

A3.1.2 Without Memory Access Conflicts

If the system were to run without memory access conflicts, the max throughput would be:

Quad Throughput = 328 KOPS x 4 = 1312 KOPS

The gain of this system over the single processor case is:

$$\text{Gain over 1 processor} = \frac{816}{328} = 2.4$$

or

$$\text{The percent of throughput utilized is } \frac{816}{1312} = 62 \text{ percent}$$

A3.2 Case 2, Four Processors Sharing a Common Memory with Time Phasing

The timing diagram for this case together with arbitration rules is found in Figure A-3.

The waiting time for each processor becomes:

$$\text{Waiting time } (\mu p1) = \frac{0}{25} = 0 \text{ percent}$$

$$(\mu p2) = \frac{1}{25} = 4 \text{ percent}$$

$$(\mu p3) = \frac{2}{25} = 8 \text{ percent}$$

$$(\mu p4) = \frac{3}{25} = 12 \text{ percent}$$

Again applying the standard airborne mix to each processor we get:

$\mu p1 = 304.5\ \mu sec + 304.5\ (0)$	$= 304.5\ \mu sec$	$= 328$ KOPS
$\mu p2 = 304.5\ \mu sec + 304.5\ (0.04)$	$= 316.6\ \mu sec$	$= 315$ KOPS
$\mu p3 = 304.5\ \mu sec + 304.5\ (0.08)$	$= 328.8\ \mu sec$	$= 304$ KOPS
$\mu p4 = 304.5\ \mu sec + 304.5\ (0.12)$	$= 341.4\ \mu sec$	$= 293$ KOPS
	Quad Throughput	1242 KOPS

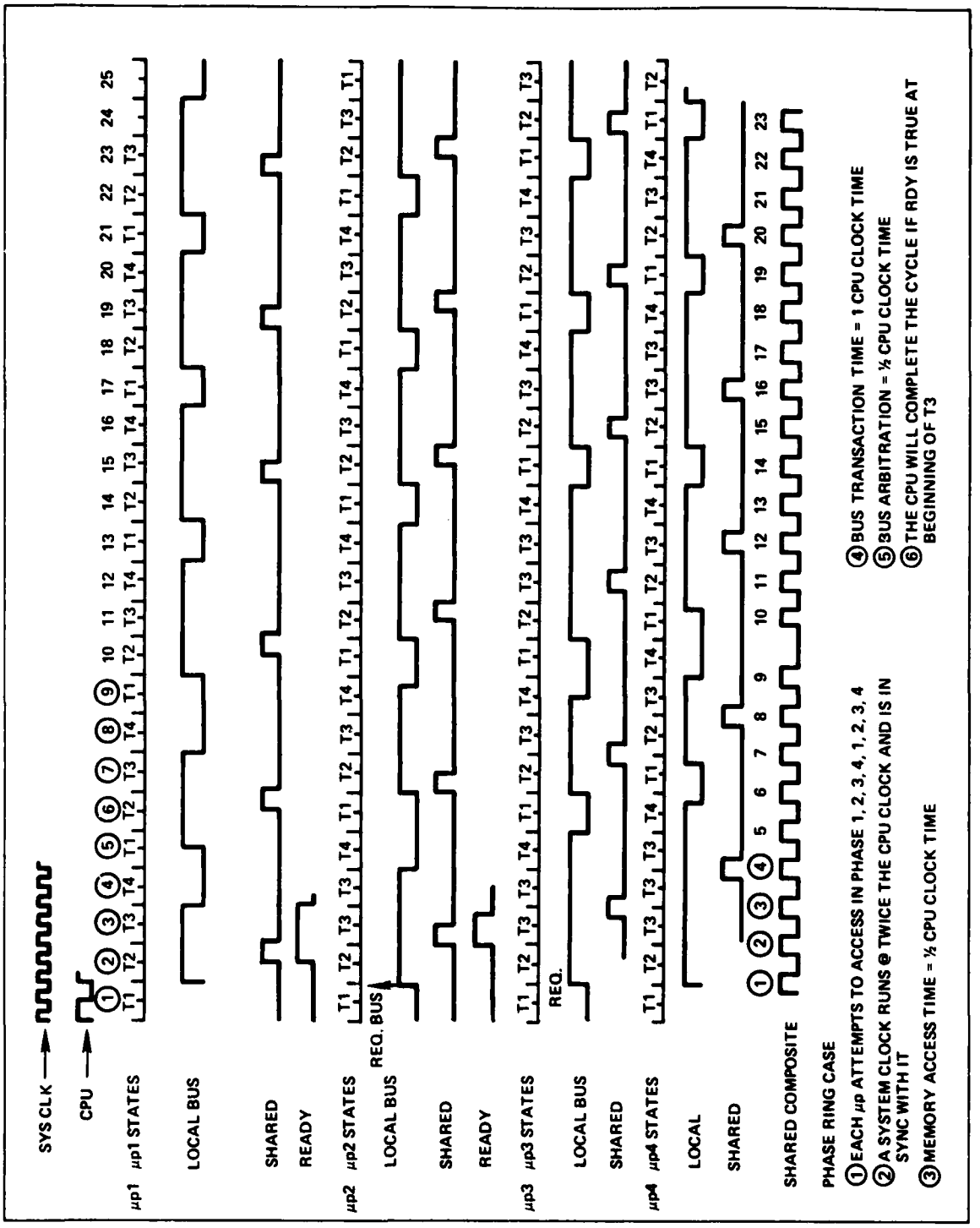


Figure A-3 - Case 2, Four Processors with Shared Memory using Time-Phased Ring Technique

The gain of this system over the single processor case becomes:

$$\text{Gain over 1 processor} = \frac{1242}{328} = 3.78$$

or

$$\text{Percent of throughput utilized} = \frac{1242}{1312} = 94 \text{ percent}$$

The effect of time phasing can be seen by comparing the two cases. This increase in throughput becomes:

Gain by using time phasing = 94 - 62 = 32 percent

A3.3 Case 3, Four Processors with Dedicated Program Memories Sharing a Common Data Memory Without Time Phasing

In this configuration, Figure A-4 the program is split so that the program resides in a local memory with shared data between processors in common memory. The split for memory access used is 75 percent to local for instructions and 25 percent for shared data.

It is noted that the shared memory is utilized 100 percent of the time. Again applying the standard airborne mix, the throughput becomes for μp1:

Access time from local	= 304 x 75 percent	= 228 μsec
Access time from common	= 304 x 25 percent	= 76 μsec

The access time to common memory involves wait states, and since memory is 100 percent utilized the same percent wait time that was determined in Case 1 applies.

Access time to common = 76 μsec + 76 (55 percent) = 117.8 μsec

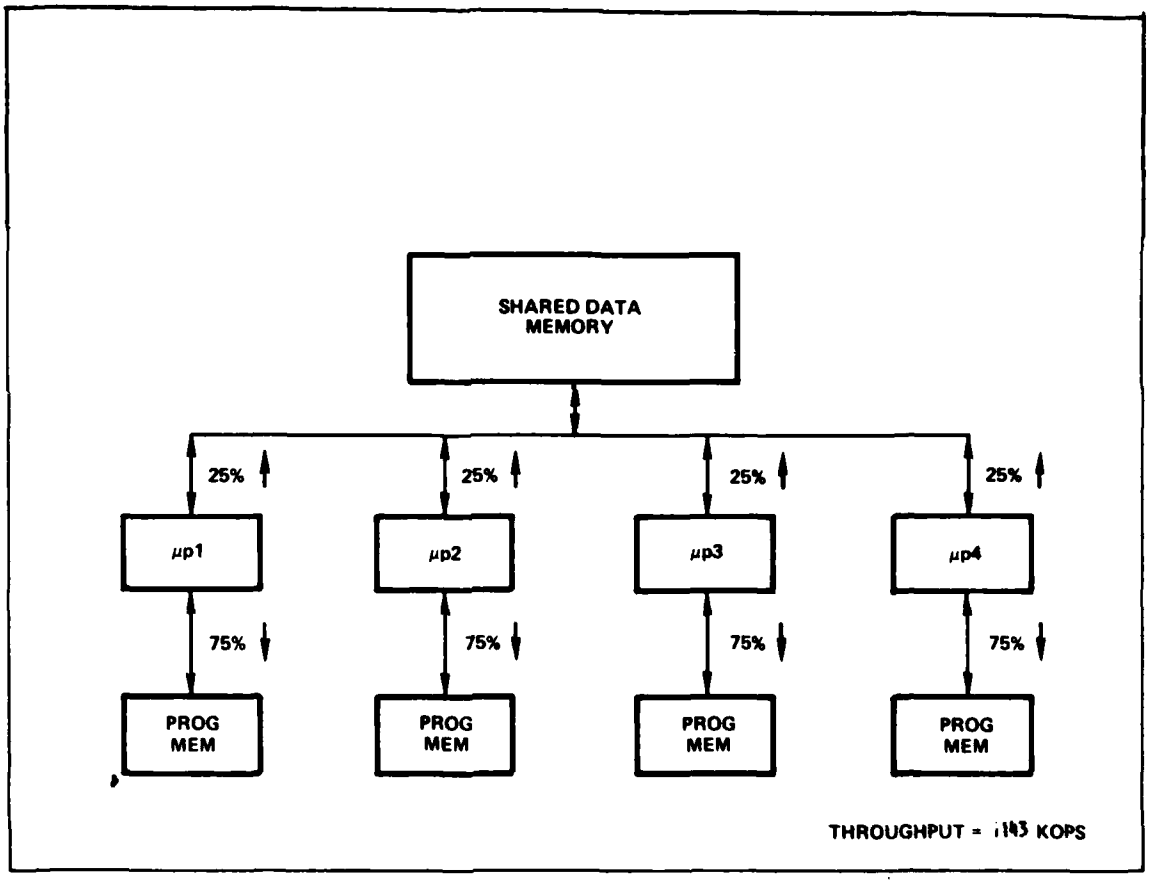


Figure A-4 - Case 3, Four Processors with Dedicated Program Memories and Shared Data Memory Without Time Phasing

The total time for the 100 standard airborne mix instructions becomes:

$$\text{Total time} = 228 + 117.8 = 345.8\ \mu\text{sec}$$

$$\mu p1 \text{ throughput} = \frac{1}{3.458\ \mu\text{sec}} = 289 \text{ KOPS}$$

$\mu p2 = 289$ KOPS

$\mu p3 = 289$ KOPS

$\mu p4 = 304\ (0.75) + 76 + 76\ (0.76) = 276$ KOPS

Quad Throughput: 1143 KOPS

The gain over the single processor case is

$$\text{Gain over 1 processor} = \frac{1143}{328} = 3.48$$

A3.4 Case 4, Four Processors with Dedicated Program Memories and Shared Data Memory (with Time Phasing)

Using the same techniques as in Case 3 the throughput becomes:

$\mu p1 = 328$ KOPS

$\mu p2 = 326$ KOPS

$\mu p3 = 323$ KOPS

$\mu p4 = 320$ KOPS

Quad Throughput: 1297 KOPS

$$\text{Gain over 1 processor} = \frac{1297}{328} = 3.955$$

A3.5 Case 5, Four Processors with Dedicated Program Memories, Different Instruction Mixes, Shared Data Memory, Without Time Phasing

In order to determine system throughput sensitivity on algorithms, a different instruction mix is applied to each processor:

µp1 =	Standard Airborne Mix	=	304 µsec
µp2 =	F-15 Auto Flight	=	275 µsec
µp3 =	R-F4 Inertial NAV	=	204 µsec
µp4 =	F-4 Fire Control	=	692 µsec

µp1 Access for instruction	= 304 x 75 percent	= 228 µsec
µp1 Access for data	= 304 x 25 percent	= 76 µsec

The common memory wait in time is:

76 + 76 (55 percent) = 117.8 µsec

µp1 total time = 228 + 117.8 = 345.8 µsec	= 289 KOPS
similarly,	
µp2	= 319 KOPS
µp3	= 430 KOPS
µp4	= 122 KOPS
Quad Throughput:	1160 KOPS

The average throughput for the single processor case is 331.25 KOPS, (Table A-2).

The gain over the single processor case is:

$$\text{Gain over 1 Processor} = \frac{1160}{331.25} = 3.5$$

In comparision to the single algorithm of Case 3, the above result shows that executing different instruction mixes in each microprocessor does not have a significant effect on the overall throughput of the quad.

APPENDIX B

REAL-TIME MISSILE SIMULATION WITH SFMCS

B1 Introduction

Given a high-speed modular processing system such as the SFMCS, it became apparent that the potential existed to significantly reduce the present high cost of missile simulations using large-scale, time-shared computing facilities, by providing instead a low-cost dedicated system using standard industry microcomputers.

High throughput and software modularity could be addressed through "super-federation", i.e., assigning one microcomputer per major airframe component or functional element. The net result of this approach would therefore be aimed at rapidly adapting any given airframe model to a new or improved version, as is typically the case during the course of specific missile development, by the substitution of alternate preprogrammed microcomputers. Further, the time-sharing and delays associated with a single large-scale computing facility would be eliminated through the replication of small dedicated microcomputer systems.

The following paragraphs describe the analysis of missile simulation functions and resulting computer performance requirements based upon the expanded SFMCS as shown in Figure B-1. Figure B-1 shows functions which immediately come to mind in terms of the missile airframe application. The following paragraphs outline the nature of missile airframe functions for simulation on the SFMCS.

B1.1 Missile Aero Model

The functional aerodata has been linearized to stability coefficients which are a function of Mach number. Since these terms are relatively, slowly variable, and they only act as multipliers in the dynamic control loops (Subsection B1.6.5), they are computed at the low frequency rate in the simulation model.

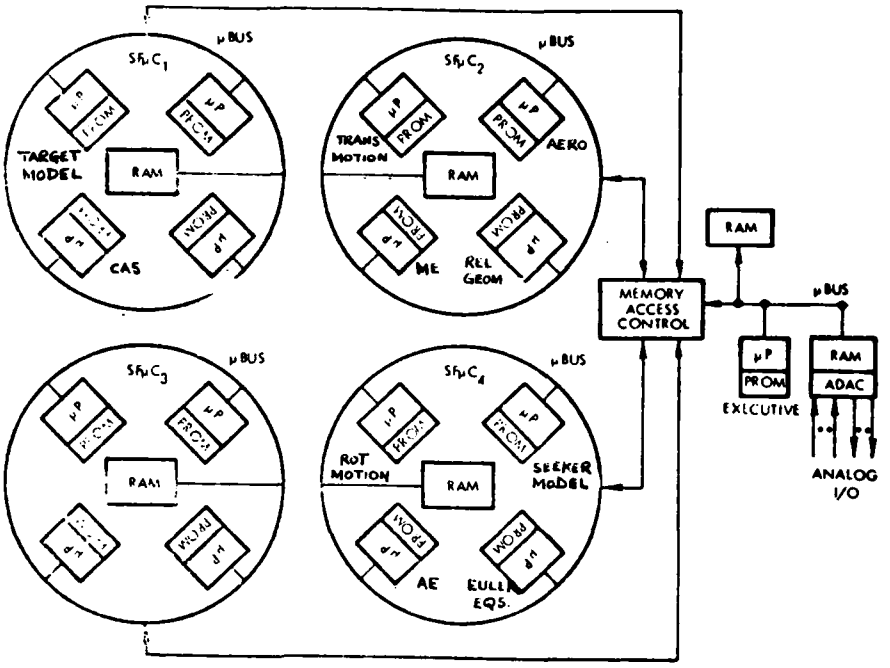


Figure B-1 - High-Performance Super-Federated Microcomputer System For Missile Simulation

The aero derivatives are as follows:

Roll

$Cl_{\delta e}$ is the aerodynamic roll effectiveness coefficient, f(Mach)

Clp is the roll damping coefficient, f(Mach)

Pitch/Yaw

$Cm\,\delta_o$ is the lateral moment effectiveness coefficient of the control surfaces at the aerodynamic reference point, -f(Mach)

$Cm\alpha_o$ is the lateral moment partial with respect to inplane angle-of attack at the reference point, -f(Mach)

$Cm_{\dot{\theta}}$ is the aerodynamic damping derivative, f(Mach)

Cn_{α} is the aerodynamic force derivative with respect to angle-of-attack, f(Mach)

Cn_{δ} is the aerodynamic force derivative with respect to control surface deflection, f(Mach)

Other aerodynamic parameters are computed in order to develop the forces and moments acting on the missile. These terms depend on certain missile states, velocity, altitude, and orientation.

Velocity ($\underline{VM}$) and altitude (RM_2) are obtained from the missile translational motion model. Altitudes (ϕ_M, θ_M, ψ_M) are obtained by snapshooting the high frequency states developed in the missile dynamics model.

The earth (inertial reference frame) to missile transformation matrix is then computed.

$$(ME) = (\phi_M)(\theta_M)(\psi_M)$$

Then the velocity vector is transformed to the missile coordinate frame.

$$\overline{V_{M_M}} = (ME)\ \bar{V}_M$$

from which the two components of angle-of-attack are generated.

$$\alpha_{yo} = \tan^{-1}(-FM_{M3}/VM_{M1})$$

$$\alpha_{zo} = \tan^{-1}(VM_{M2}/VM_{M1})$$

where the subscript refers to the value of these quantities at the start of the low frequency calculation cycle. Note that in the missile dynamics model that these quantities are updated between the low frequency calculations at the high frequency data rate.

Dynamic pressure and Mach number are also computed as part of this model. Air density and velocity of sound are table look-ups,

$$\rho = f(RM_2)$$

$$V_s = f(RM_2)$$

and Mach number is developed as

$$MACH = |VM|/V_S$$

and dynamic pressure is given by

$$Q = 1/2\ \rho|V_M|^2$$

Note that this model neglects the effect of wind.

B1.2 Missile Physical Properties

The characteristics of the missile that are categorized as missile physical properties are the thrust, mass, inertia, etc. This model generates those quantities which, when combined with the aerodynamic forces and moments, produce the rotational and translational acceleration of the missile.

The basis of the model is the defined time history of the motor thrust, and the relationship of this to the other physical properties of the missile. Thus, the thrust profile is determined from a table of thrust level specified at arbitrary time points, with linear interpolation for intermediate values. That is;

$$THRUST = f(t)$$

There is a single state variable associated with this model which is the total impulse, or energy expended while thrusting. That is obtained via the integration of the rate of energy expenditure, thrust.

$$IMP = {}_0\int^t THRUST \cdot dt$$

The other physical properties are directly proportional to energy (fuel) expended, so that they may be drived from impulse.

$$\text{MASS} = \text{MASS}_{IC} + (\partial \text{MASS}/\partial_{IMP})\ \text{IMP}$$

$$I_{xx} = I_{xx_{IC}} + (\partial I_{xx}/\partial_{IMP})\ \text{IMP}$$

$$I_{yy} = I_{yy_{IC}} + (\partial I_{yy}/\partial_{IMP})\ \text{IMP}$$

$$\Delta X_{CG} + \Delta X_{CG_{IC}} + (\partial \Delta X_{CG}/\partial_{IMP})\ \text{IMP}$$

These are missile mass (MASS), roll moment of inertial (I_{xx}), pitch-yaw moment of inertia (I_{yy}), and center of gravity displacement (X_{CG}) from the longitudinal reference point (at the missile nose).

The partial derivatives are assumed constant over the entire range of thrust, with an average value used to insure the proper parameters in the missile glide condition.

Other properties of the missile necessary to scale the aerodynamic quantities to force and moment are as follows:

X_o = Reference point at which moment data is taken
S = Aerodynamic reference area
$\bar{C}$ = Reference dimension, pitch and yaw
b = Reference dimension, roll

B1.3 Missile Translational Motion

Missile translational motion is described by integration of Newton's equations in an inertial reference frame. The acceleration vector developed from aerodynamic force, missile thrust, and other forces applied to the missile (such as launcher constraints) is converted to an Earth fixed reference frame (E-frame) and combined with gravity to produce the three-component acceleration vector which is successively integrated to velocity and position.

The acceleration in the missile fixed axes is given by,

$$\overline{NM} = (NM_1, NM_2, NM_3)^T$$

The conversion to the E-frame is achieved through the transformation matrix describing the relative orientation of the missile coordinates and the inertial reference. When gravity is added, the total inertial acceleration results.

$$\overline{AM} = \overline{G} + (ME)^T \overline{NM}$$

This is then integrated to obtain the velocity and position states

$$\overline{VM} = \overline{VM}_{IC} + \int \overline{AM} \cdot dt$$

$$\overline{RM} = \overline{RM}_{IC} + \int \overline{VM} \cdot dt$$

These states are typically assigned to the low frequency regime of state variables in the continuous subsystem. Because these equations have been decoupled from the rotational modes of the missile (via ME^T), there is no significant component of the higher frequency motion present in these equations. The missile linear acceleration due to the aerodynamic and internal forces (NM) is generated from the aerodynamic parameters and the physical properties of the missile.

$$NM_1 = \left(THRUST + qS(C_{Do} + C_{XTHR})\right)/MASS$$

where

C_{XTHR} = A drag correction coefficient to account for drag increase when motor burns out (switched to zero prior to burnout)

C_{Do} = The base drag coefficient, f(Mach)

$$NM_2 = \frac{qS}{MASS} (C_{N\alpha} \cdot \alpha_P + CN_\delta \cdot \delta_P)$$

$$NM_3 = \frac{qS}{MASS} (C_{N\alpha} \cdot \alpha_Y + C_{N\delta} \cdot \delta_Y)$$

B1.4 Relative Geometry Model

The relative geometry model combines the missile and target states to produce several quantities of prime importance in the simulation. The driving terms to the guidance of the missile are the relative position and velocity of the target with respect to the missile.

The relative range is merely the difference between the target and missile position vectors,

$$\overline{RTM} = \overline{RT} - \overline{RM}$$

while the relative velocity is given by,

$$\overline{VTM} = \overline{VT} - \overline{VM}$$

The line-of-sight to the target (as seen from the missile) is obtained by unitizing the relative range vector,

$$\overline{LOS} = u\ (\overline{RTM})$$

The closing velocity is the relative velocity along the line-of-sight, that is

$$V_C = -(\overline{VTM} \cdot \overline{LOS})$$

and projected time-to-go is given by,

$$t_{go} = V_C \cdot |\overline{RTM}| \ / \ |\overline{VTM}|^2$$

B1.5 Miss Distance Calculation

When a simulated flight is terminated due to the time-to-go to the target becoming negative, a miss distance iteration is made. This calculation is based on the missile and target states near intercept, and assumes that over the short iteration time missile and target acceleration is constant.

The range between the missile and target can be expressed as a quadratic in time-to-go.

$$R_F + R + \dot{R} \cdot t_{go} + \ddot{R} \cdot t_{go}^2/2$$

where

R, $\dot{R}$ and $\ddot{R}$ are the relative range, velocity, and acceleration that exist between the target and missile at the time the simulation run is terminated, but prior to the miss distance iteration.

The relative velocity is given by,

$$V = \dot{R} + \ddot{R} \cdot t_{go}$$

The minimum approach to the target by the missile is defined by the condition,

$$\bar{R} \cdot \bar{V} = 0$$

This requires the solution of a cubic equation which is solved by a Newton-Raphson iteration method. The iteration is performed until the time-to-go has settled to within 1 μsec of the solution, or if 100 passes have been made through the iteration cycle. The result is the amount of time the extrapolation covered, the magnitude of the miss distance vector, and the three components of the miss distance vector.

B1.6 High Frequency Models

The high frequency regime contains the rotational motion of the missile and the models of the missile subsystems, the control actuator section, inertial instruments, the seeker gimbal system, and the receiver.

B1.6.1 Control Actuator Section (CAS)

The control actuator modeled as a first order transfer function with rate and position limits. There are four such actuators, one for each control surface. The block diagram for a single actuator is illustrated in Figure B-2. Table B-1 lists the inputs, outputs, and parameters of the CAS model.

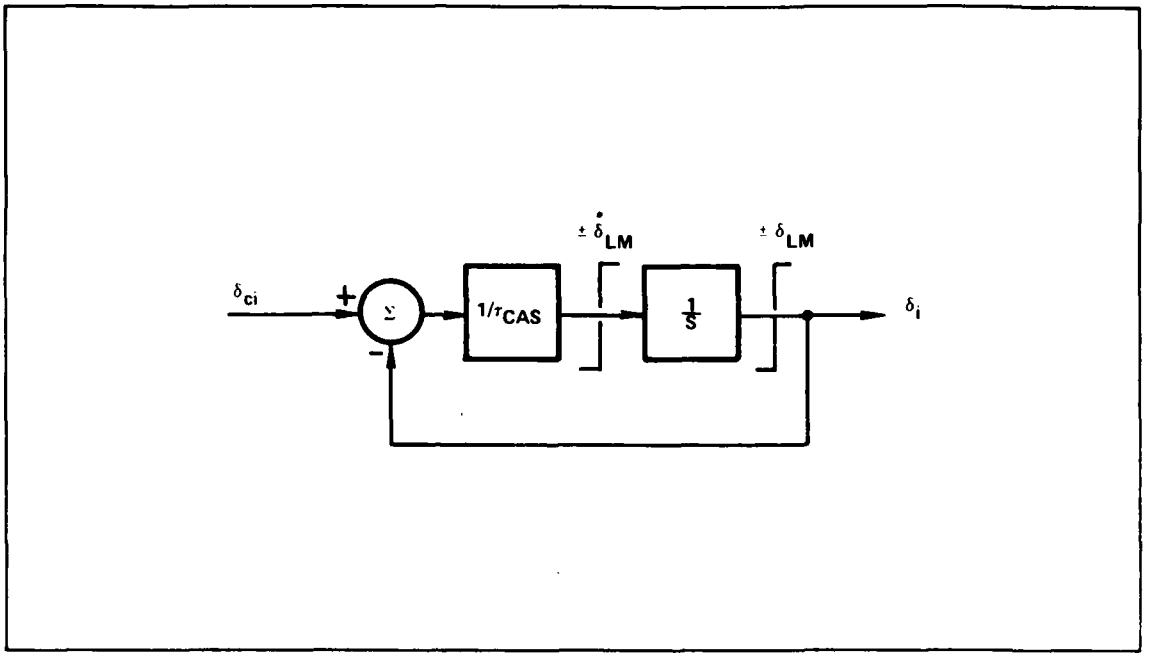


Figure B-2 - Control Actuator Section

TABLE B-1
CAS MODEL QUANTITIES

Type	Quantity	Nominal Value	Definition
Input	δc; (1-4)	-	Control surface angle command
Output	δ ; (1-4)	-	Control surface angle
Parameter	τ CAS $\dot{\delta}$ LM δ LM	0.01 sec 5.236 rad/sec 0.5236 rad	Response time constant Rate limit Angle limit

B1.6.2 Inertial Instrument Models

The Level-1 model of the instruments represents the more significant errors present in the measurement process.

These errors are added to the quantities to be measured as determined by the kinematics. The errors include the effects of gyro drift and accelerometer bias, but not the dynamics of response of the measurement devices. Also included are the output limits representing the dynamic range of the devices.

The gyro measurements are modeled by the following expressions:

$$\underset{\sim}{\hat{W}}_P = \text{LIM}\left\{ \underline{WM}' \begin{bmatrix} BGY1LM \\ BGY2LM \\ BGY3LM \end{bmatrix} \right\}$$

where

$$WM'_1 = WM_1 + DDGIN1B$$

$$WM'_2 = WM_2 + DDGIN2B$$

$$WM'_3 = WM_3 + DDGIN3B$$

and

DDGIN1B, DDGIN2B, DDGIN3B are the zero-gee drift rates of the respective gyros. These are nominally zero but may be used for error sensitivity studies.

Each of the error sources is determined from statistical distributions with specified mean and variance. For Monte-Carlo analysis a new value of each parameter is chosen for each flight in a sample set, to represent the errors typical of random missiles.

The accelerometer measurements are modelled by the following expressions:

$$\underset{\sim}{\hat{N}}_P = \text{LIM}\left\{ \underline{NP}' \begin{bmatrix} BAC1LM \\ BAC2LM \\ BAC3LM \end{bmatrix} \right\}$$

where

$$NP'_1 = NM_1 + NP_{1o}$$

$$NP'_2 = NM_2 + NP_{2o}$$

$$NP'_3 = NM_3 + NP_{3o}$$

B1.6.3 Head Control and Stabilization Model

The model of the missile seeker is defined to represent head control and stabilization. It is configured to operate in the inertially stabilized model, so as to represent the operation of the missile seeker. The inertially stabilized mode uses gyros mounted on the antenna to achieve the desired rate stabilization.

The head control and stabilization system is pictured in block diagram form in Figure B-3. There are two control axes, an outer gimbal which is the pitch axis, and an inner gimbal which is the yaw axis. Head control is achieved by rate commanding the seeker in the inertially stabilized mode. Base motion, which is developed by the missile rotation and for the inner gimbal by outer gimbal motion, is decoupled from the seeker track loop by the inherent inertial stabilization of the electric motor drive, and supplemented by gyro feedback.

The stabilization loop has been simplified to a first order response, but improved low frequency stabilization could be achieved with a more complex representation of the stabilization loop dynamics. The amount of coupling through the motor and gearing is defined by the parameters KGR2 and KGR3. If these are zero, then no base motion is coupled into the antenna drive, while when they are 1, the base motion is directly coupled.

B1.6.4 Receiver Measurement Model

The receiver measurement model produces the equivalent boresight errors from the geometric tracking errors and the various noise sources present in the measurement process.

The geometric error is derived from the line-of-sight to the apparent (glinting) target and the seeker orientation with idealized antenna patterns included in the model. The line-of-sight vector is transformed to the antenna coordinates by:

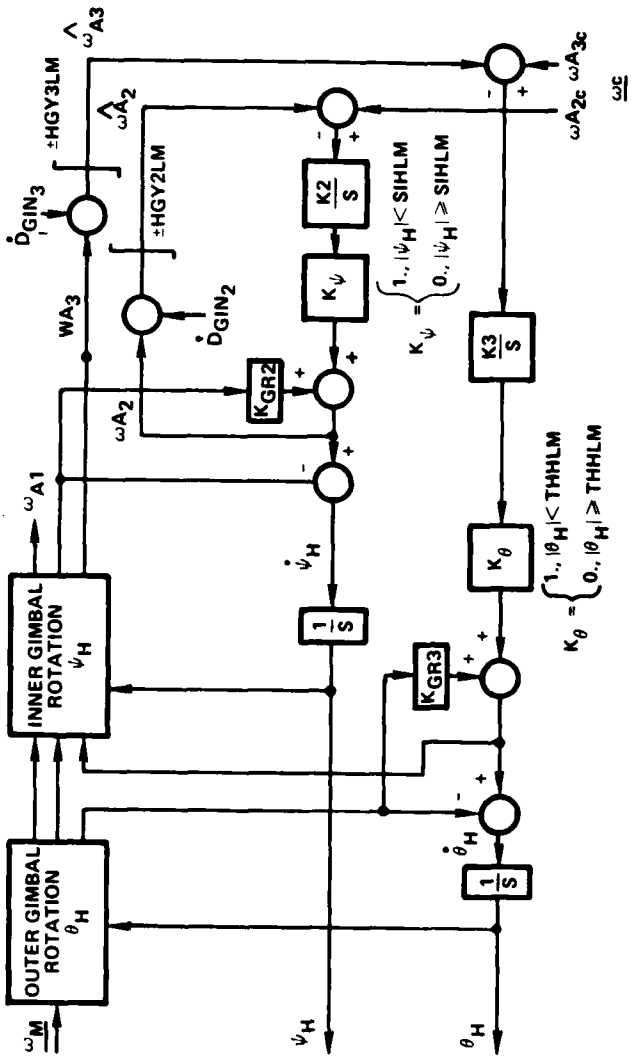


Figure B-3 - Head Control and Stabilization Model

$$\underline{u}'_A = (AE)\ \underline{u}'_E$$

where

$$(AE) = (\psi_M)(\theta_M)(ME)$$

The idealized antenna pattern model then produces the monopulse channel signals as follows,

$$\Sigma v = 1.0$$

$$\Delta_{pv} = u'_{A2}$$

$$\Delta_{yv} = -u'_{A3}$$

and the boresight error (prior to adding other errors) is

$$\epsilon_2 = \Delta_y v/\Sigma v$$

$$\epsilon_3 = \Delta_p v/\Sigma v$$

The range and range rate (prior to adding errors) and taken directly from the geometry. That is,

$$\dot{R} = V_c$$

$$R_{TM} = |\underline{R_{TM}}|$$

The measurement noise is added to produce angle errors and the doppler velocity to be used for target tracking and guidance.

Receiver thermal noise and range independent noise are generated by adding band limited Gaussian noise to the quantities that have been derived from the geometry. The bandwidth is assumed constant as it is representative of the receiver hardware. The noise variance due to thermal noise is dependent upon the receiver signal-to-noise ratio, which changes with the effective radar cross section of the target and the power loss due to range. The signal-to-noise ratio (S/N) is computed from,

$$S/N = \frac{R_O^4 \ (S/N_O) \ \sigma_T}{R_{TM}^4 \ \sigma_O}$$

where

R_O, S/N_O, and σ_O = Normalizing parameters to represent the illuminator power and missile receiver gain

σ_T = The effective bistatic cross section of the target as determined by the fading target model

R_{TM} = The magnitude of the range vector between missile and target

The noise variance of the signals (which is considered white noise) is given by,

$$V_{RN} = (\sigma^2_{RN})/(1+S/N) + \sigma^2_{SN}$$

where

$$\sigma^2_{RN} = \phi RN/ \Delta t_H$$

$$\sigma^2_{SN} = \phi SN/ \Delta t_H$$

and

ϕ_{RN}, ϕ_{SN} = The spectral densities of receiver and servo noise (range independent)

Δt_H = The integration step size for the high frequency states in the simulation

The standard deviation of a Gaussian distribution is determined from the square root of the noise variance, VR_N, and then filtered so as to represent the noise bandwidth as a filter with a time constant τ_N. The resultant noise is added to the geometric error along with the other error sources modeled to produce the guidance errors as indicated in Figure B-4.

The measurement of closing rate is assumed to be the ideal (geometric) closing velocity, VC.

B1.6.5 Missile Dynamics Model

The aerodynamic forces and moments are developed by combining the aerodynamic coefficients with the missile physical properties. However, in the case of the moment generation, care must be taken to preserve the dynamic integrity of these calculations. Since the rotational modes of the missile are of significantly wider bandwidth than the translational motion, an effectively smaller computation cycle is required. This is achieved, while maintaining computational efficiency, through hybridization of the airframe model.

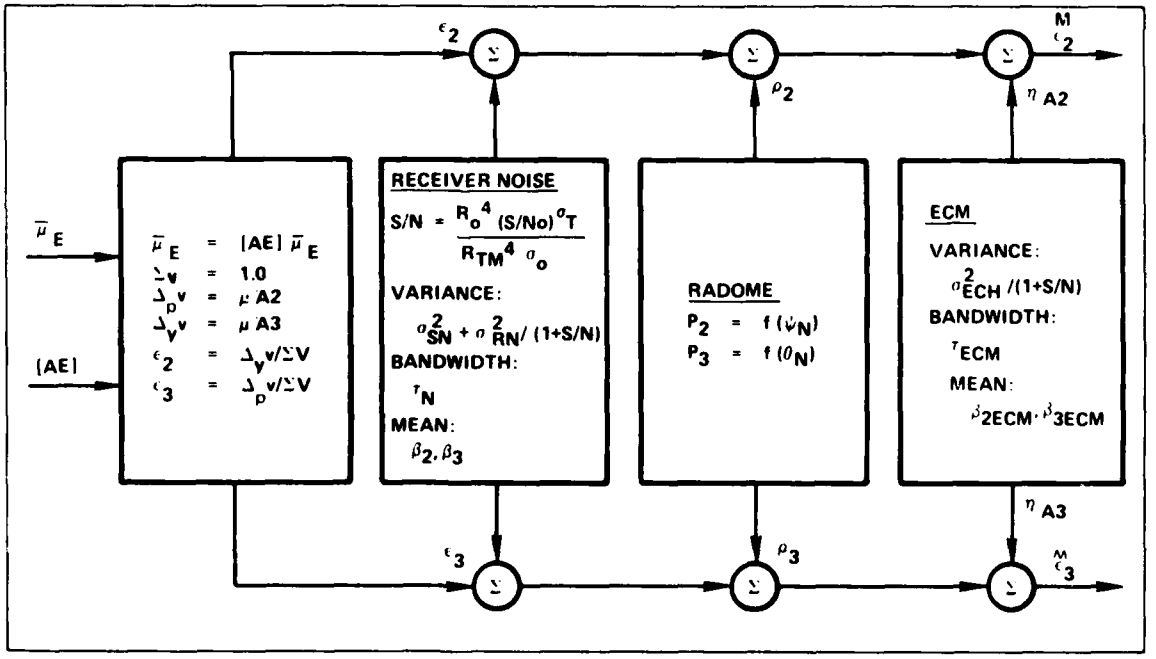


Figure B-4 - Receiver/Error Source Model

B1.6.6 Moment Generation

Moments are not a direct output of this model; rather, the torque/inertia ratio, which is the inertial acceleration (rotational), is the quantity sent to the rotational motion model.

These terms are developed in the manner indicated in Figure B-5. The model is "hybrid" in nature, in that some terms are computed at the high data rate and others at the lower data rate. The low data rate terms are typically multipliers to the high data rate variables. The multipliers themselves are slowly varying quantities. In this role they do not contribute phase errors to the high bandwidth loops. The simulation computational savings gained by this form of implementation is that the function generation (aerodynamic coefficients, missile physical parameters) is performed at a relatively low data rate (see missile kinematics model).

This provides adequate compensation to make the quantities α_Y and α_P effectively computed at the high data rate.

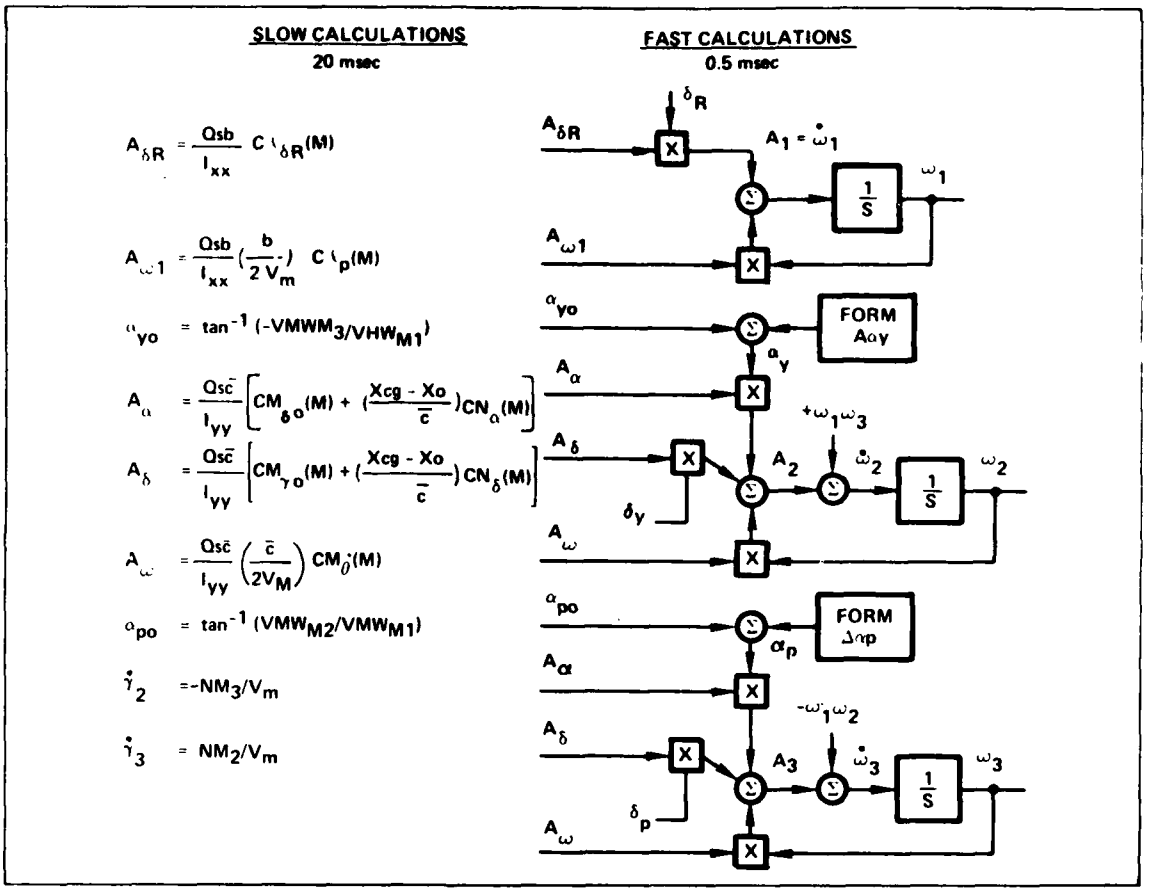


Figure B-5 - Aeromodel Partitioning

Missile rotational motion is obtained by integration of a modified set of Euler equations. The missile body-fixed axes system is chosen in the direction of the principal axes of the missile, thus making the products of inertia vanish. There remains the Euler form of the rotational equations of motion.

$$I_{xx}\,\dot{W}_{M1} + (I_{zz} - I_{yy})\,W_{M2}\,W_{M3} = T_1$$

$$I_{yy}\,\dot{W}_{M2} + (I_{xx} - I_{zz})\,W_{M1}\,W_{M3} = T_2$$

$$I_{zz}\,\dot{W}_{M3} + (I_{yy} - I_{xx})\,W_{M1}\,W_{M2} = T_3$$

where

T_1, T_2, and T_3 = The components of torque applied to axes 1, 2, and 3

I_{xx}, I_{yy}, and I_{zz} = The moments of inertia about axes 1, 2, and 3

W_{M1}, W_{M2}, and W_{M3} = The rotational rates about these axes

Since missiles are nearly symmetrical in pitch and yaw,

$$I_{yy} \approx I_{zz}$$

and since roll moment of inertia is typically only 1 or 2 percent of the lateral inertia

$$I_{xx} << I_{yy}$$

so that the equations can be reasonably simplified to

$$\dot{W}_{M1} = \frac{T_1}{I_{xx}}$$

$$\dot{W}_{M2} = \frac{T_2}{I_{yy}} + W_{M1}\, W_{M3}$$

$$\dot{W}_{M3} = \frac{T_3}{I_{yy}} - W_{M1}\, W_{M2}$$

These terms are the derivatives of the body rate vector, whose components are the state variables of this model. Since these states are in the high bandwidth control loop of the autopilot, it is necessary to perform this state integration at a sufficiently small step size so that no adverse lag is imparted by the simulation.

Missile orientation is developed by integrating a set of Euler rates driven by the missile body rate vector.

These rates are integrated to produce the Euler angles 0_M, ψ_M, and θ_M which define the orientation state (relative to the inertial reference) of the missile. From these angles we compute the earth-to-missile transformation matrix.

B1.7 Approximate Computer Requirements

The computer requirements are estimated based on the speed of execution and the accuracy required to implement the control functions and provide an accurate simulation of the physical system. These requirements are defined for each of the four function groups described earlier.

B1.7.1 Guidance Functions

The computers implementing the guidance function must execute the algorithms so that the autopilot commands and seeker tracking commands are generated in NGT 10 msec. The filter prediction and inertial reference algorithms must be complete prior to the beginning of the next guidance cycle (20 msec). The result is that the computer can support a guidance cycle rate of 50 Hz and the guidance and tracking commands suffer from no greater than a 10 msec computer time delay (transport lag).

The computer should have at least 16 bits of dynamic range with double precision (or equivalent) to maintain the inertial states to the equivalent of 1 m/sec^2 acceleration resolution over a dynamic range of 20,000 m.

The input/output functions should be supported by modules of NLT 10 bits.

B1.7.2 Autopilot Functions

The computers implementing the autopilot function must execute the algorithms so that the control actuator commands are generated in NGT 1 msec. The Euler angle integration for the altitude reference must be complete prior to the beginning of the next autopilot cycle (2 msec). The result is that the computer can support an autopilot cycle rate of 500 Hz and the autopilot stability path suffers from no greater than a 1 msec compute time delay (transport lag).

The computer should have at least 16 bits of dynamic range.

The input/output functions should be supported by modules of NLT 10 bits.

B1.7.3 Low Frequency Physical Models

The computers implementing the low frequency physical models must execute the algorithms in a period NGT 20 msec.

The computers should have a word size of NLT 16 bits, with double precision (or equivalent) for the velocity and position states of the missile and target in order to achieve trajectory accuracy of 0.1 m with a 20,000 m dynamic range.

The input/output functions should be supported by modules of NLT 12 bits.

B1.7.4 High Frequency Physical Models

The computers implementing the high frequency physical models must execute the algorithms in a period NGT 0.5 msec.

The computer should have a word size of NLT 16 bits.

The input/output functions should be supported by modules of NLT 12 bits.

B1.8 Conclusion

In conclusion, it appears quite feasible to perform real-time digital missile simulations using the SFMCS. Further, the degree of configuration/flexibility and independence from time-shared facilities are two unique advantages.

APPENDIX C

SFMCS FUNCTIONAL BLOCK DIAGRAMS

This appendix contains functional block diagrams of the basic quad and expanded SFMCS architecture, which employs four quads and one host processor. These designs formed the basis of the detailed timing analyses and simulation models.

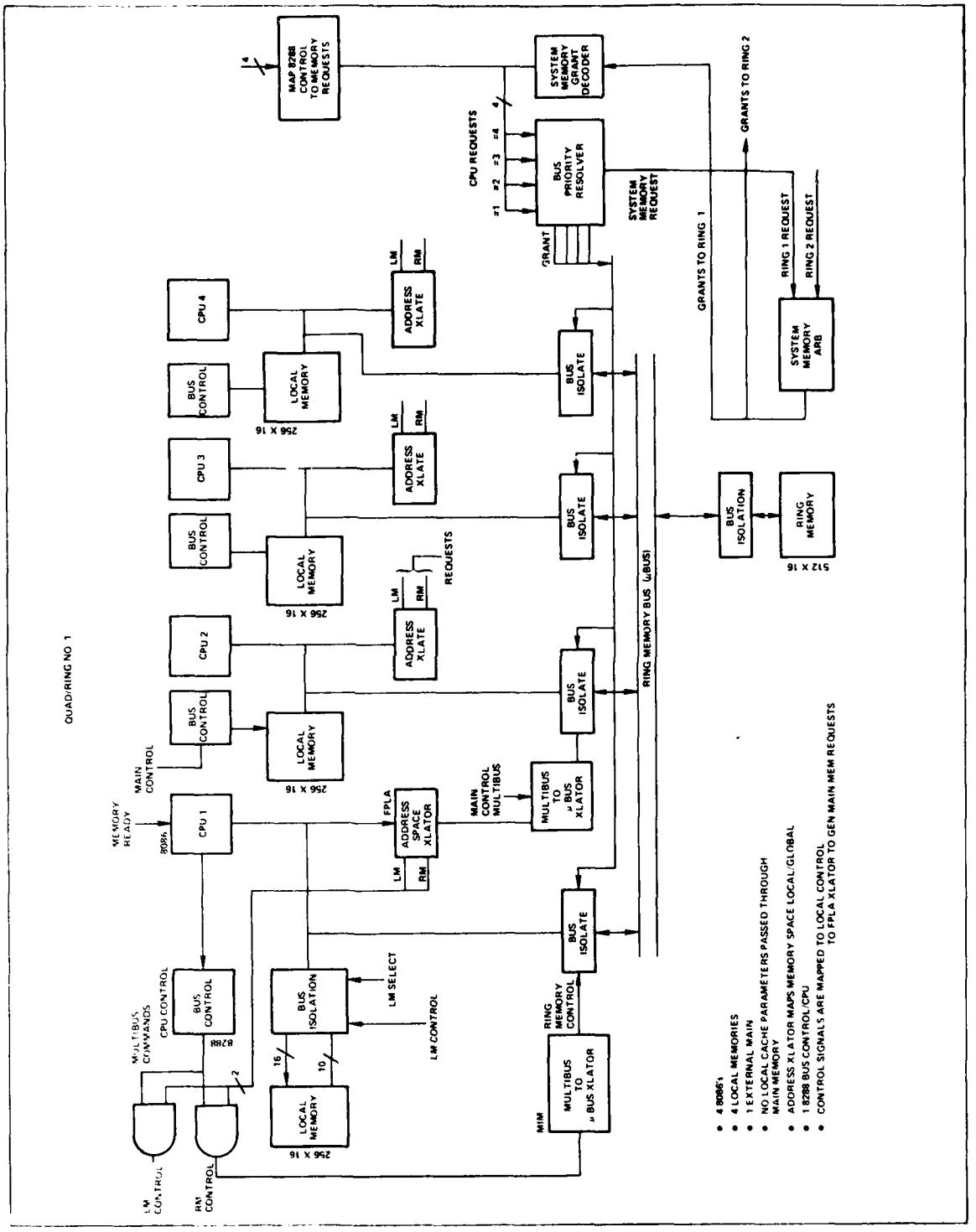
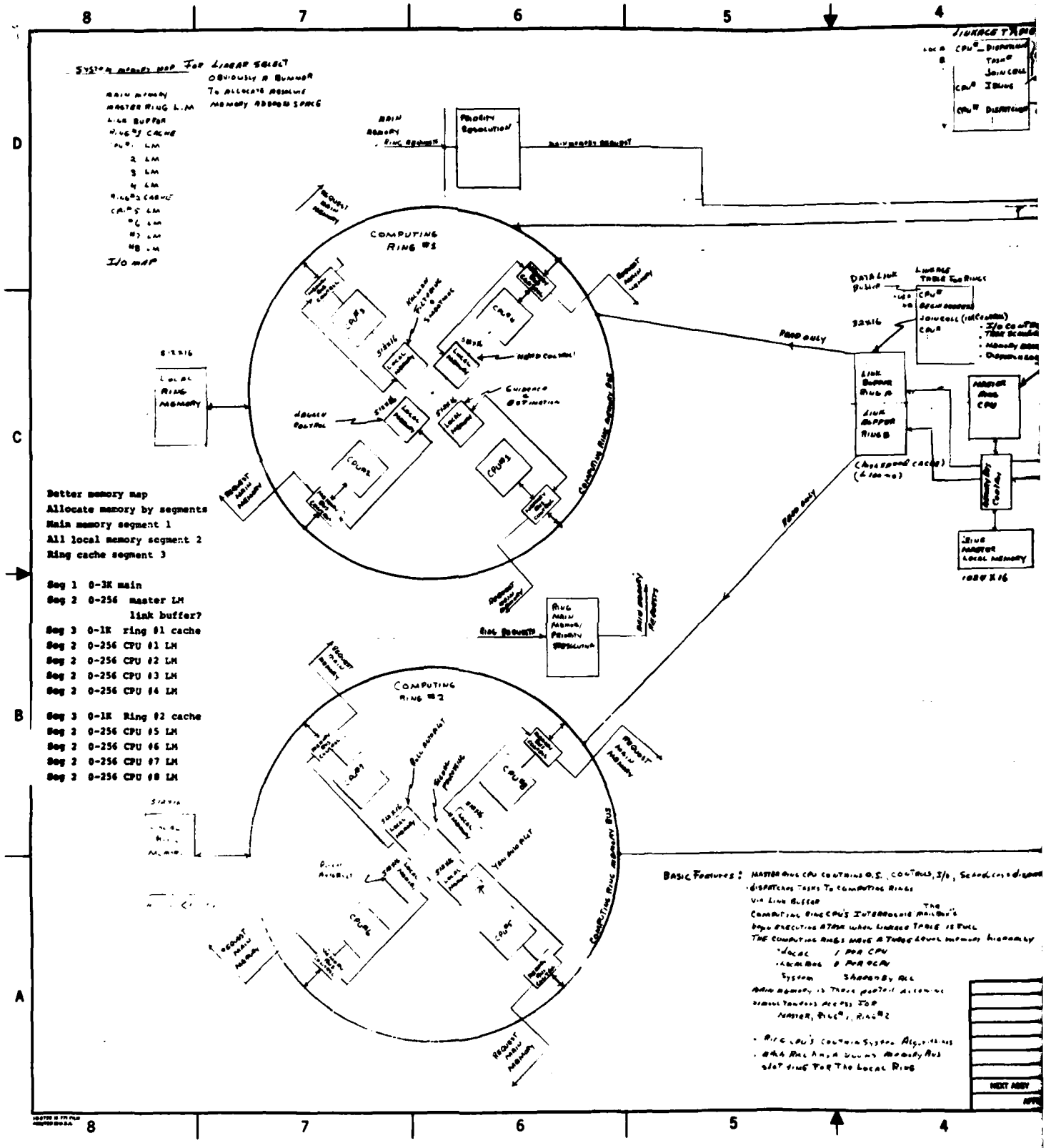


Figure C1 SFMCS Quad, Block Diagram


COMPUTING RING #1
COMPUTING RING #2
Better memory map
Allocate memory by segments
Main memory segment 1
All local memory segment 2
Ring cache segment 3
Seg 1 0-3K main
Seg 2 0-256 master LM
link buffer?
Seg 3 0-1K ring #1 cache
Seg 2 0-256 CPU #1 LM
Seg 2 0-256 CPU #2 LM
Seg 2 0-256 CPU #3 LM
Seg 2 0-256 CPU #4 LM
Seg 3 0-1K Ring #2 cache
Seg 2 0-256 CPU #5 LM
Seg 2 0-256 CPU #6 LM
Seg 2 0-256 CPU #7 LM
Seg 2 0-256 CPU #8 LM
I/O MAP
Basic Features:
NEXT ASSY

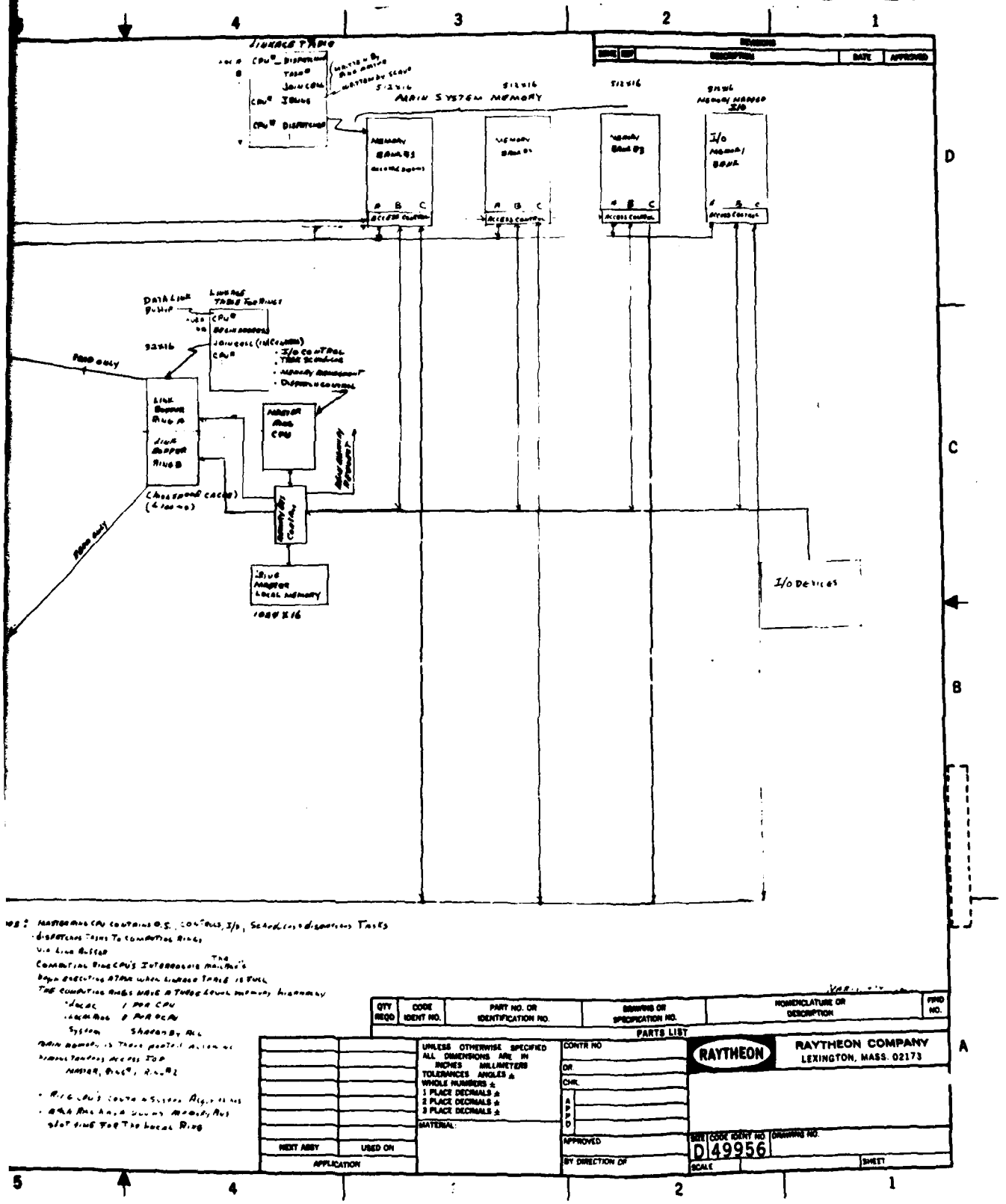
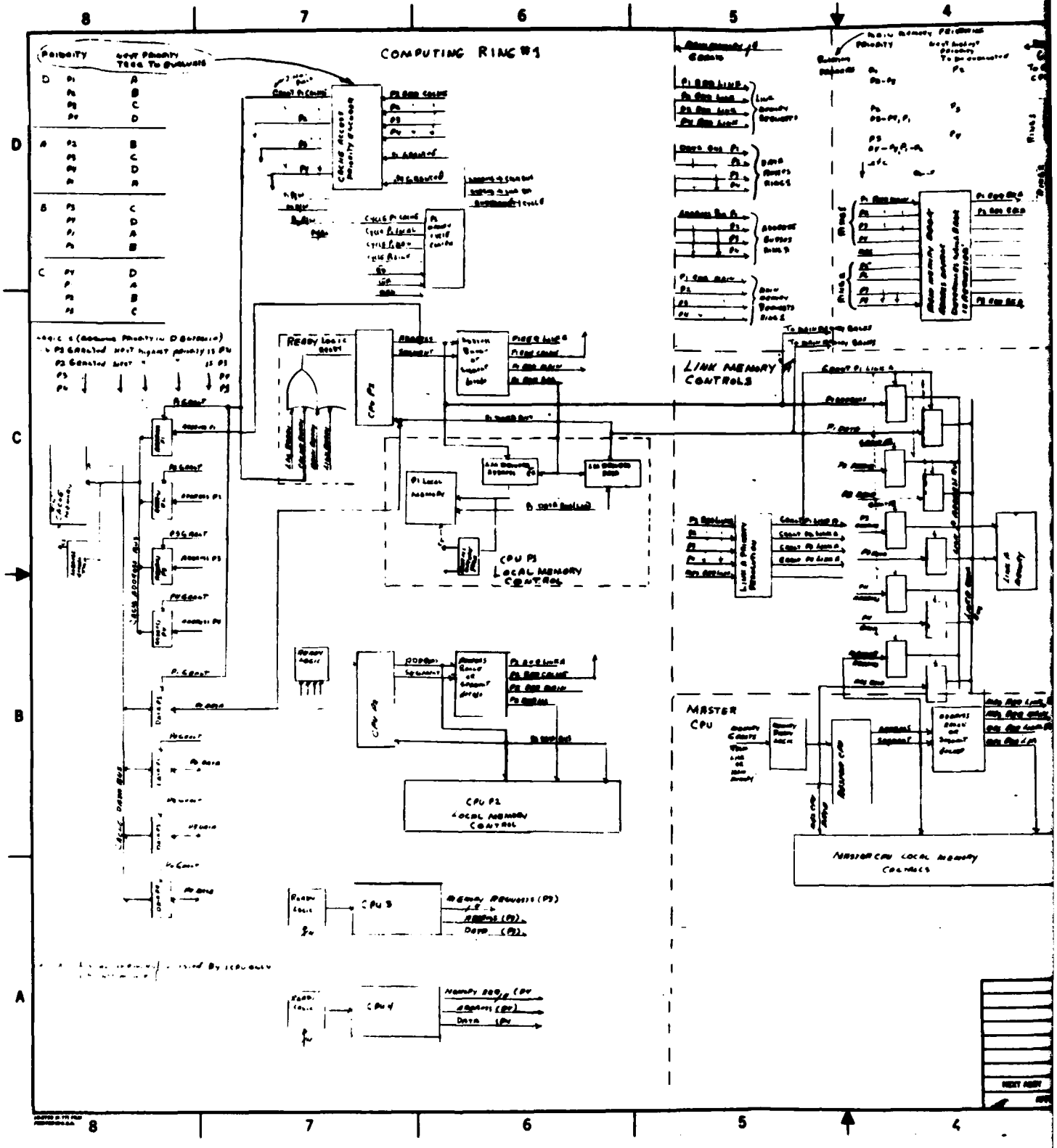


Figure C2 SFMCS - Expanded System, Dual Quad Configuration, Block Diagram


COMPUTING RING #1
LINK MEMORY CONTROLS
CPU P3 LOCAL MEMORY CONTROL
CPU P2 LOCAL MEMORY CONTROL
MASTER CPU
CPU 3
CPU 4

Figure

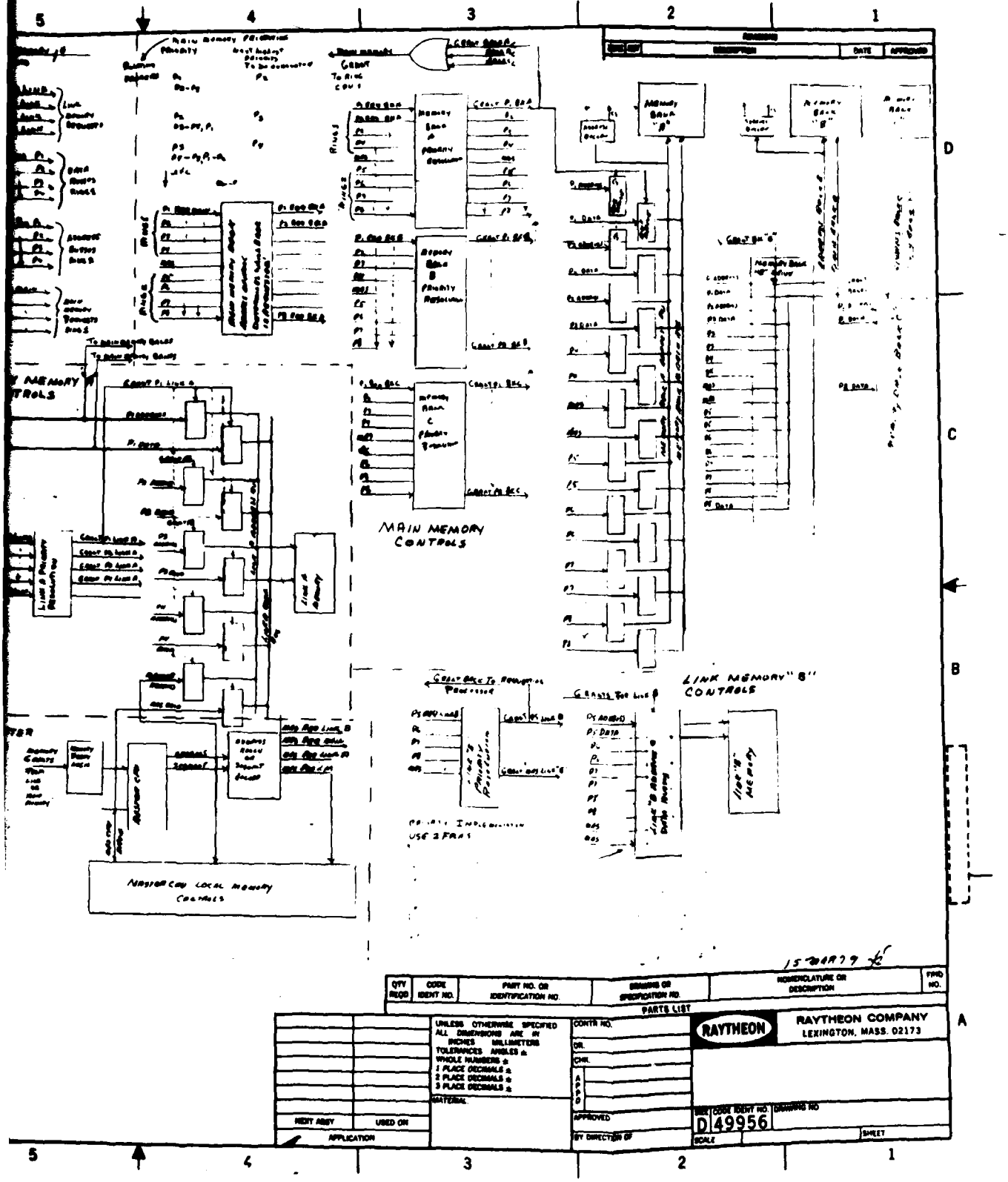


Figure C3 - SFMCS - Expanded System, Detailed Block Diagram

DISTRIBUTION LIST

Air Force Avionics Laboratory
Wright Patterson AFB, Ohio 45433

AFAL/DHM (Mr. R. E. Conklin) (1)

Air Force Systems Command
Andrews AFB, Md 20334

AFSC/DLCA (Mr. P. Sandler) (1)
AFSC/SDZ (COL R. Loukota) (1)
AFSC/XRL (COL F. Smith) (1)

Armament Development and Test Center
Eglin AFB, Florida 32542

ADTC/DL (COL A. D. Brown) (1)
ADTC/SD7 (Mr. G. Kirby) (1)
ADTC/ADDE (Dr. J. W. Jones) (1)
ADTC/DLMI (LCOL Britton) (1)
ADTC/DLMT (MAJ E. Halprin) (1)
AFATL/DLMM (Mr. M. Henne) (1)

Chief of Naval Operations (1)
The Pentagon
Washington, D. C. 20350

NOP-982E1 (1)
NOP-987P3 (1)

DCASR
Raytheon Company
Wayside Ave.
Burlington, Mass 01803 (1)

Defense Technical Information Center
Bldg. 5, Cameron Station
Alexandria, VA 22314 (1)

Director of Defense Research and Engineering
The Pentagon
Washington, D. C. 20350
Mr. M. Keller (Rm. 3E1081) (1)
Mr. G. Kopcsak (Rm. 3D1089) (1)

Harry Diamond Laboratory
2800 Powder Mill Road
Aldephi, MD 20783
Dr. Carter (1)

Institute for Defense Analysis
400 Army Navy Drive
Washington, D. C. 22202
Dr. T. C. Bartee (1)
Mr. G. W. Preston (1)

National Aeronautic and Space Administration
Headquarters
Washington, D. C. 20546
NASA-KC (Mr. W. E. McInnis) (1)

National Aeronautic and Space Administration
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91103
NASAJPL-114-122 (Mr. P. E. Lecoq) (1)
NASAJPL-114-122 (Mr. M. Ebersole) (1)
NASAJPL-158-205 (Mr. R. E. Covey) (1)
NASAJPL-/T1180 (Mr. R. W. Scott) (1)

Naval Air Development Center
Warminster, PA 18974
NADC-2031 (Mr. C. M. Nowicki) (1)

Naval Air Systems Command
Washington, D. C. 20361
NAIR-03P2 (Mr. J. W. Malloy) (1)
NAIR-03P3 (Mr. R. H. Krida) (1)
NAIR-350F (Mr. R. J. Wasneski) (1)
NAIR-360B (Mr. B. A. Zempolich) (1)
NAIR-360 (Mr. C. Thomas) (2)
NAIR-52022B1 (Mr. G. B. Cudd) (1)
NAIR-533DX (Mr. R. S. Entner) (1)
PMA-263 (CAPT. W. M. Locke) (1)

Naval Avionics Facility
Indianapolis, Ind 46218
NAFI-072.9 (Mr. R. H. Huss) (1)
NAFI-D072 (Mr. W. Bridges) (1)
NAFI-D824 (Mr. C. W. Hagen) (1)
NAFI-D824 (Mr. J. Carr) (1)
NAFI-D900 (Mr. R. Barnett) (1)
NAFI-D924 (Mr. G. Forsee) (1)

Naval Electronic Systems Command
Washington, D. C. 20360
ELEX-304 (Mr. J. Cauffman) (1)

Naval Material Command
Washington, D. C. 20362
NMAT-0321 (1)

Naval Ocean Systems Center
San Diego, CA 92152

NOSC-19 (Mr. S. Maynard)	(1)
NOSC-722 (Ms. N. S. Mathis)	(1)
NOSC-8222 (Mr. D. Wilcox)	(1)
NOSC-9102 (Mr. W. Dejka)	(1)
NOSC-923 (Mr. E. C. Urban)	(1)
NOSC-923 (Mr. J. F. Poulson)	(1)
NOSC-923 (Mr. E. S. Holland)	(1)
NOSC-9232 (Mr. R. D. Peterson)	(1)
NOSC-923 (Mr. A. Johnson)	(1)

Naval Post Graduate School
Monterey, CA 93940

Dr. T. F. Tao 62TV	

Naval Sea Systems Command
Washington, D. C. 20362

NSEA-03132 (Mr. R. Cuddy)	(1)
NSEA-0341 (Mr. T. Tasaka)	(1)
NSEA-393M2	(1)
PMS-406	(1)

Naval Surface Weapons Center
Dahlgren Laboratory
Dahlgren, VA 22448

DF-34 (Mr. R. Holden)	(6)
DF-34 (Mr. R. Dorsey)	(1)

Naval Surface Weapons Center
White Oak Laboratory
Silver Spring, MD 20910

WA-32 (Mr. A. Kolodzinski)	(1)
(Mr. R. Bost)	(1)

Naval Weapons Center
China Lake, CA 93555

NWC-01 (Mr. R. M. Hillyer)	(1)
NWC-015 (Dr. R. Cartwright)	(1)
NWC-033 (Mr. C. Neal)	(1)
NWC-395 (Mr. W. G. Hueber)	(1)
NWC-3130 (Mr. L. Lakin)	(1)
NWC-3132 (Mr. R. Hawkins)	(2)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

ONR 212 (LCDR W. Savage)	(6)
ONR 221 (Mr. J. Trimble)	(1)

Office of Naval Research Branch Office
495 Summer Street
Boston, Mass 02210 (1)

U.S. Army Electronics Command
Fort Monmouth, N.J. 07703

NL-BP (Mr. D. Haratz)	(1)

U.S. Army Missile R&D Command
Redstone Arsenal, Ala 35809

DRDMI-TG (Mr. J. B. Huff)	(1)
DRDMI-TGG (Mr. D. Ciliax)	(1)
DRDMI-TE (Mr. D. Holter)	(1)
DRDMI-EA (Mr. C. Riley)	(1)
DRCPM-PE-X (Mr. W. Jann)	(1)
DRCPM-MD (Mr. E. Wood)	(1)

U.S. Naval Research Laboratory
Washington, D.C. 20375

NRL 2627	(6)
NRL 5216 (Mr. L. Palkuti)	(1)
NRL 5260 (Dr. D. Barbe)	(1)


DATE
FILME
8